UNIVERSITY OF CALIFORNIA,
IRVINE

Cyber-Physical-System-On-Chip (CPSoC): An Exemplar Self-Aware SoC and Smart Computing Platform

THESIS

submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Computer Science

by

Santanu Sarma

2016

To my family

Contents

List of Figures

List of Tables

Acknowledgments

During the last five years at the University of California Irvine, I have had the privileged to interact with several fine individual who directly or indirectly contributed to the success of my dissertation work, and more importantly provided me the opportunity to grow and expand my knowledge. I would like to express my utmost gratitude to everyone who has guided, supported and helped me achieve my academic and professional goals. Especially, my advisors: Professor Nikil D. Dutt and Professor Alex Nicolau, who gave me the opportunity to pursue my dream under their tutelage. I am extremely grateful to them for giving me the freedom to explore the ever growing landscape of computer science and pursuing my own interests in an exciting and stimulating environment. Their mentorship has helped me grow as a person and learn the necessary skills to grow as a successful individual in the community.

I take this opportunity to express my gratitude to Prof. Nalini Venkatasubramanian, for her guidance and support throughout my Ph.D. and course work. I have enjoyed every opportunity to collaborate and interact with her during various group meetings, brainstorming for ideas, as well as writing research papers. I thank her for introducing me to several promising research directions such as cross-layer resilience issues in cyber-physical systems, adaptive and reflexive middleware, and Internet-of-things.

I am really fortunate that I had the opportunity to work with Prof. Axel Jantsch of TU Vienna, Austria and Prof. Puneet Gupta of UCLA on many emerging areas, especially computational self-awareness, and hardware variability respectively. The interaction with them greatly influenced my understanding of several aspects of the thesis. I was fortunate to have been part of the Variability Expedition project where I got the opportunity to learn from Prof. Subhashish Mitra, Prof. Rajesh Gupta, Prof. Mani Srivastava, Prof. Tajana Simunic Rosing, Prof. Steve Swanson, Prof. Rakesh Kumar, Prof. Andrew B. Kahng, and Prof. Dennis Sylvester. They have all at some point in my life made an enormous impact, encouraged me to follow my interest, and think big. I would like to thank Prof. Elaheh (Eli) Bozorgzadeh, Prof. Marco Levorato, Prof. Tony Givargies and Prof. Sharad Mehrotra for their guidance and help at various stages of my research. I would like to specially mention Sudeep Pasricha and Houman Homayoun, who always encouraged my work and gave me the words I needed to hear to keep moving forward.

I would also like to express my gratitude to all my present and past mentors : Dr. V.K. Agrawal, Dr. J.K. Kishore, Subramaniyam Udupa, Dr. Venkateswarlu Andra, Prof.

Sudeep Pasricha, Prof. Houman Homayoun, Prof. Sung-Soo Lim, Prof. Alfonso Avila, and Prof. Krishnendu Chakrabarty. I would like to thank Dr. Ruddy Beraha and Dr. Karam S. Chatha of Qualcomm Research Center for the opportunity to present my work and the kind support to pursue my research.

I would like to thank my friends and colleagues at UCI, who made a great impact in my life. In particular, I thank Dr. Luis Angel D. Bathen, Dr. Rosario (Ro) Cammarota, Arup Chakraborty, JurnGyu Park, Majid Namaki Shoushtari, Michael Beyeler, Bryan Donyanavard, Roger Chen-Ying Hsieh, Tiago Rogerio Muck, Kasra Moazzemi, Hirak Kashyap, Kazuyuki Tanimura, Abbas Banaiyan, Hossein Tajik, Dr. Jun Yong Shin, Dr. Juan Gonzalez, Dr. Janmartin Jahn, Dr. Codrut Stancu, Prof. Gustavo Giro, and Dr. Trent (Chen-Kang) Lo. Special thanks to Dr. Yuko Hara-Azumi, Prof. Yukio Mitsuyama, Prof. Hoyoung Hwang, Prof. Preeti Panda, Prof. Gu-Min Jeong for all the stimulating discussions and insights into their research work. I would also like to take this opportunity to thank my colleagues in NSF Variability Expedition and Cypress project : Lucas Wanner, Mark Gottscho, Abbas Rahimi, Nga Dang, Kyle Benson, Zhijing Qin, Leila Jalali, M. Reza Rahimi, Liyan Zhang, Dr. Minyoung Kim, and Dr. Grit Denker of SRI for giving me the opportunity to know about their research work. Similarly, I would like to thank my former colleagues in ISRO Satellite Centre, who gave me the opportunity to collaborate on several exciting projects that resulted in successful publications. I am especially thankful to my friends and colleagues Ajay Andhiwal, Rahul Mishra, Partha Bandyopadhyay for their immense help and support in numerous situations. Special thanks to my friends Arunima Bhattacharjee, Rupsa Datta, Hrishi Chandanpurkar, Rohit Athavale, Zonglin Guo, Sky Faber, Sukomal Paul, Sirishkumar Jayine, Harsh Sahu, and Pramit Choudhary for all their help, support, and memorable time we spent together. I extend my gratitude and thanks to all those friends and colleagues I could not personally mention here.

I would like to thank UC Irvine's graduate division for their help and various sources of funding. I especially thank Melanie Sander and Grace Wu for making CECS such an enjoyable place to work and for all the help and advice on so many different subjects. I would like to thank National Science Foundation (NSF) for partially supporting my work with NSF award CCF-1029783 (Variability Expedition).

Lastly, I am thankful to my wife who had been of immense help and my core support system during the course of this dissertation work. Without her support and help, I would have never been able to complete this thesis work. I am greatly indebted to my parents and in-laws, my sister and brother-in-law, and all my relatives for their encouragement and support during this course of study. My younger brother performed many of my responsibilities in my absence on several occasions, and my deepest appreciation for all the hardship he took on my behalf. I would like to thank all of them for making my stay at Irvine enjoyable and memorable during this course of study.

CURRICULUM VITAE

SANTANU SARMA

EDUCATION

Ph.D. in Information and Computer Science 2016
University of California, Irvine Irvine, California

M.S in Computer Science 2015
University of California, Irvine Irvine, California

M.Tech in ECE, Specialization Control System Technology 2002
Indian Institute of Technology Guwahati Guwahati, India

B.E in Electrical Engineering 1999
National Institute of Technology Agartala Agartala, India

RESEARCH EXPERIENCE

Graduate Student Research Assistant 2013-2016
University of California, Irvine Irvine, California

Reader / Teaching Assistant 2011-2013
University of California, Irvine Irvine, California

Research Scientist (C/D/E) 2002-2011
Indian Space Research Organization, Bangalore Bangalore, India

AWARDS AND HONORS

Chair's Fellowship 2011-2016
Donald Bren School of Information and Computer Science,
University of California, Irvine

Satellite Technology Development Award 2011
Indian Space Research Organization, ISRO Satellite Centre,
Bangalore

Satellite Technology Development Award 2010

Indian Space Research Organization, ISRO Satellite Centre, Bangalore

Young Engineer Award 2009
Indian National Academy of Engineering (INAE)

GATE Scholarship 2000-2002
Indian Institute of Technology, Guwahati

University Gold Medal 1999
NIT Agartala (Tripura University), India

SELECTED PUBLICATIONS

Toward Smart Embedded Systems: A Self-aware System-On-Chip (SoC) Perspective 2016
ACM Transection on Embedded Computing Systems, TECS, 2016.
(Invited Keynote Paper)

Cross-layer Virtual/Physical Sensing and Actuation for Resilient Heterogeneous Many-core SoCs 2016
ASPDAC'16, Macau, 2016.

SmartBalance: A Sensing-Driven Linux Load Balancer for Energy Efficiency of Heterogeneous MPSoCs 2015
DAC'15, SF, USA, 2015.

Cyber-physical System-on-Chip (CPSoC): A Self-Aware MPSoC Paradigm with Cross-Layer Virtual Sensors and Actuators 2015
DATE'15, Grenoble, France, 2015

Run-DMC: runtime dynamic heterogeneous multicore performance and power estimation for energy efficiency 2015
In Proceedings of the 10th International Conference on Hardware/Software Codesign and System Synthesis (CODES '15), Amsterdam, 2015

Self-Aware Cyber-physical Systems-on-Chip 2015
In Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, ICCAD '15, Austin, USA, 2015

Cross-Layer Exploration of Heterogeneous Multicore Processor Configurations VLSI Design, India, Jan 2015	2015
Minimal Sparse Observability of Complex Networks: Application to MPSoC Sensor Placement and Run-time Thermal Estimation & Tracking Design, Automation and Test in Europe Conference and Exhibition, DATE'14, Dresden, Germany, 2014	2014
On-chip Self-Awareness Using Cyber-physical-Systems-on-Chip (CPSoC) Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis. CODES'14, New Delhi, 2014.	2014
Sense-making from Distributed and Mobile Sensing Data: A Middleware Perspective Proceedings of the The 51st Annual Design Automation Conference on Design Automation Conference. DAC'14, SF, USA, 2014	2014
FPGA Emulation and Prototyping of a Cyber-physical-System-On-Chip (CPSoC) Rapid System Prototyping, RSP'14, 25th IEEE International Symposium on. IEEE, 2014	2014
Cross-Layer Virtual Observers for Embedded Multi-processor System-on-Chip (MPSoC) Adaptive and Reflective Middleware, ARM'12, Montreal, Canada, Dec 3-7, 2012	2012

BOOK CHAPTER

Architecture and Cross-layer Design Space Exploration of Heterogeneous Muti-core Processors *Handbook of Hardware/Software Codesign*, Springer, to appear in 2016	2016

TECHNICAL REPORTS

Cyber-physical-System-On-Chip (CPSoC): Sensor-Actuator Rich Self-Aware Computational Platform 2014
UC Irvine, Technical Report, CECS TR 13-06. Revised June 26, 2014.

Strength of Diversity: Exploiting Cheap Heterogeneous Noisy Sensors for Accurate Full-Chip Thermal Estimation and Prediction 2014
UC Irvine, Technical Report, CECS TR 14-01, 2014

SOFTWARE

SmartBalance 2015
https://github.com/santanusarma/smartbalance

CPSoCSim: CPSoC, HMP Simulator. 2014
https://github.com/santanusarma/CPSoCSim

INVITED TALKS AND PRESENTATIONS

Cyber-Physical-System-On-Chip (CPSoC): A Self-Aware SoC Paradigm for Smart Embedded Systems 2016
ECE Department, George Mason University, March, 2016.

Essence: A Machine Learning Approach to Sense-making for Internet-of-Things 2016
IoT Workshop, ESWEEK, 2016

Energy and Performance-aware task scheduling in heterogeneous parallel platforms 2012
ACM SRC, Design Automation Conference, DAC'12, San Francisco, June, 2012

PROFESSIONAL MEMBERSHIPS

ACM Student Member	2012
IEEE Member	2003

ABSTRACT OF THE THESIS

Cyber-Physical-System-On-Chip (CPSoC): An Exemplar Self-Aware SoC and Smart Computing Platform

by

Santanu Sarma

Doctor of Philosophy in Computer Science

University of California, Irvine, 2016

Professor Nikil Dutt, Chair

Embedded systems are increasingly seeing the need for self-awareness to operate autonomously in the face of uncertainty and unpredictability in the environment, the applications they execute, and in the manufactured hardware. The notion of self-awareness enables a system to monitor its own state and behavior such that it is capable of making judicious decisions and adapt intelligently. However, emerging Multiprocessor Systems-on-chip (MPSoCs), used by these embedded systems and devices, still treat the elements of intelligence, specifically self-awareness, as a second-class design requirement, supporting them with ad hoc and poorly-developed awareness mechanisms, architectural supports, and system software. This dissertation overcomes these limitations by providing the foundation for a new class of self-aware adaptive MPSoCs called a Cyber-Physical-System-on-Chip (CPSoC). Unlike traditional MPSoCs, CPSoCs are distinguished by an intelligent co-design of the control, communication, and computing (C3) infrastructure while considering both the cyber and physical aspects together so as to adaptively achieve desired objectives and goals. CPSoC's sensor-actuator rich scalable architecture intrinsically couples on-chip and cross-layer sensing and actuation to enable self-awareness in a principled way. The thesis corroborates, through experiments and FPGA prototypes, the key idea that giving the SoC the freedom to opportunistically adapt the software and the hardware stack by infusing self-awareness mechanisms and steerable knobs across the stack can open up new and otherwise untapped opportunities in energy efficiency, performance, and thermal resilience.

Chapter 1

Introduction

The primary driver for computing systems innovation has been phenomenal scalability of manufacturing process that has allowed us to build computing systems at exponentially growing capacity over the last three decades. Transistor scaling improved integration density, speed and energy exponentially following Moore's law [13]. As sequential processor performance peaked in the early 2000s and faced several improvement barriers [350], computer architects were forced to turn to parallelism to sustain performance growth. This shift was primarily motivated by power concerns, as the end of Dennard scaling meant it was no longer practical to increase core frequency with each technology generation [58, 94, 107]. Although core frequency and complexity have since plateaued and the properties of CMOS technology scaling have changed dramatically, the number of cores in emerging SoCs has increased exponentially following Moore's law.

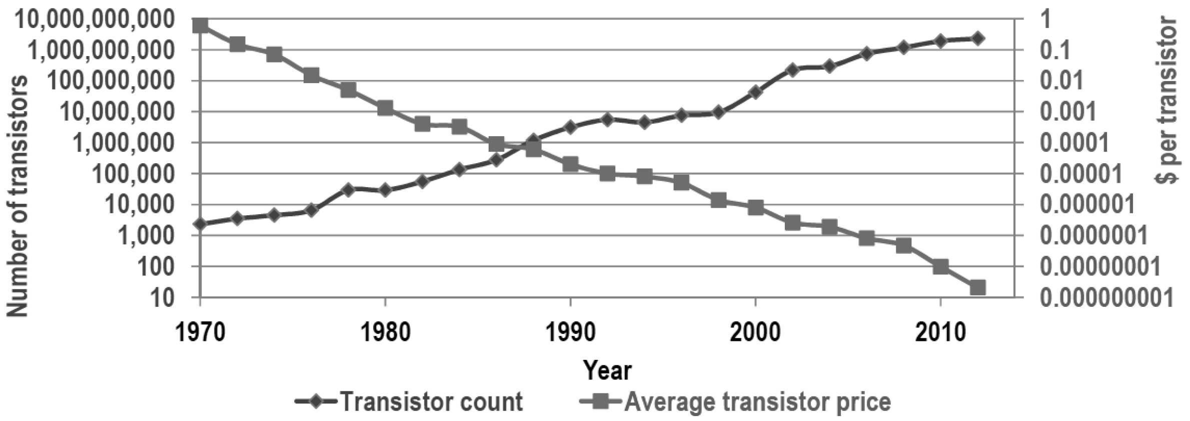


Figure 1.1: Number of transistors integrated into a singe chip and average cost per transistor over time.

As integrated circuit technology dives deeper into the nanoscale era, a multitude of challenges stemming from the end of perfect Dennard scaling [107, 280, 111] and worsening process variations [54] are introduced. Several studies indicate [42, 174] that radical approaches including major shift in architecture, software, and operating system are needed in order to address these new and emerging challenges.

1.1 Challenges in Emerging SoC

1.1.1 Increasing Heterogeneity and the Era of Dark Silicon

Due to the breakdown of Dennard scaling, the percentage of a silicon chip that can switch at full frequency is dropping exponentially with each process generation as shown in Fig.1.2b. "Dark silicon" will be a defining feature of future SoCs, where only small portions of a chip may be powered on at a time in order to manage power density and heat [107]. This utilization wall forces designers to ensure that, at any point in time, large fractions of their chips are effectively dark or dim silicon, i.e., either idle or significantly underclocked. As exponentially larger fractions of a chip's transistors become dark, silicon area becomes an exponentially cheaper resource relative to power and energy consumption. This shift is driving a new class of architectural techniques that spend area to improve energy efficiency and seek to introduce new forms of heterogeneity into the computational stack. The dark silicon phenomenon and the need for greater energy proportionality and efficiency are major driving forces behind research and development in many-core SoCs and heterogeneous devices, architectures, and systems.

Widespread use of specialized architectures that leverage these techniques in order to attain orders-of-magnitude improvements in energy efficiency is expected to increase. However, many of these approaches also suffer from massive increases in complexity and the need for developing heterogeneous architectures that insulate the hardware designer and the programmer from the underlying complexity of such systems. Careful understanding of the underlying tradeoffs and benefits are crucial as several application domains and diverse heterogeneous cores increasingly converge to a single multicore SoC platform [53], which must address a multitude of potentially conflicting design and runtime constraints such as resiliency, energy, heat, cost, performance, security, etc., all in the face of highly dynamic operational behaviors and environmental conditions.

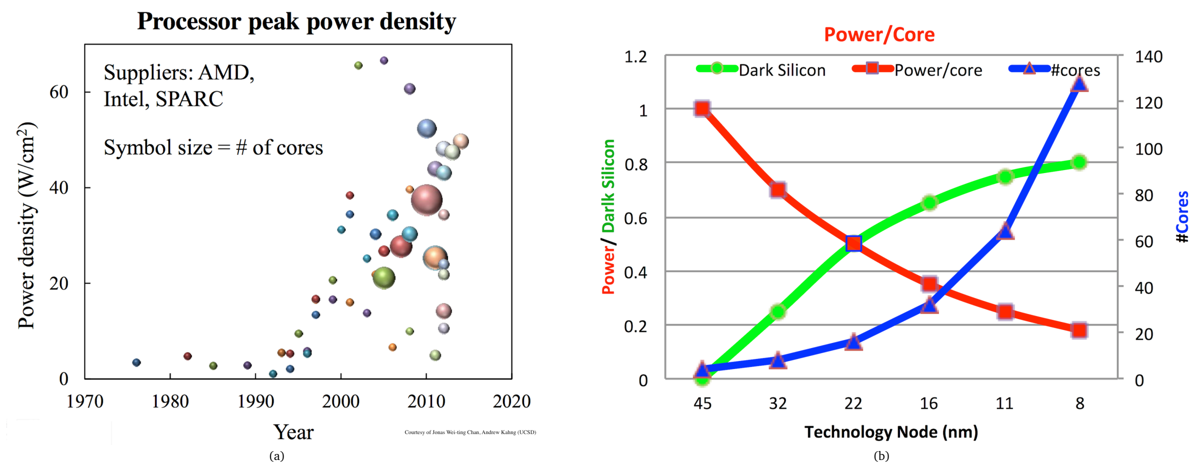


Figure 1.2: (a) Power density in multi-core trend (b) dark silicon in emerging SoC.

1.1.2 Increasing Variability

Emerging computing systems exhibit increasing variation in performance, power consumption, and reliability parameters across the manufactured parts as well as across the use of these parts over time in the field from three fronts as illustrated in Fig. 1.3. This variability has led to increasing use of over design and guard-bands in design and test to ensure yield and reliability to a rigid set of specifications. Variability in supply voltage, chip temperature, manufacturing process, and transistor aging have imposed a large amount of pessimistic margins in clock frequency, voltage, and device size, which has severely undermined gains from various advancements in technology scaling.

In particular, many-core computational platforms already face significant resiliency challenges with errors resulting from manufacturing process variability, exponentially increasing power dissipation and heating, environmental effects (e.g., radiation induced soft errors [35], and aging/wear-out [43]). These problems are exacerbated in the nanometer era with exploding core counts and on-chip resources. The combined systematic and random effects in nanoscale technologies results in high variability (and thus higher error rates) manifested at the circuit level all the way to the architecture and system levels [54], requiring new strategies for ensuring application resilience when executing on these computing platforms.

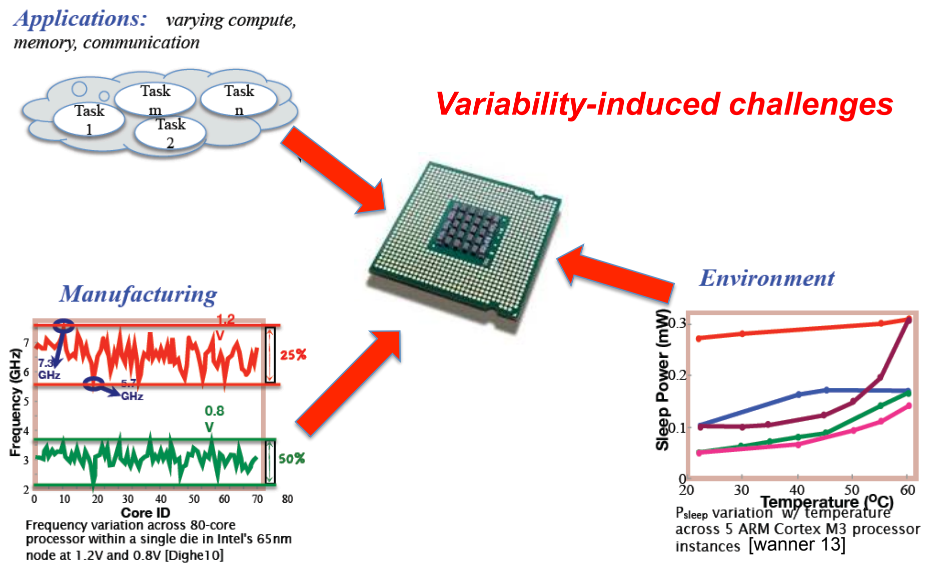
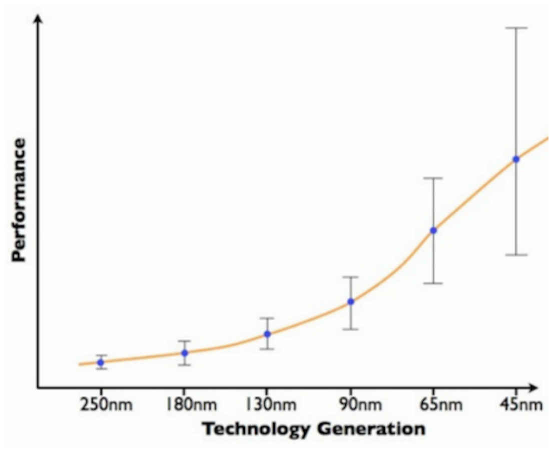

Applications: varying compute, memory, communication
Task 1
Task m
Task 2
Task n
Variability-induced challenges
Manufacturing
Frequency (GHz)
1.2 V
7.3 GHz
5.7 GHz
25%
0.8 V
50%
Core ID
Frequency variation across 80-core processor within a single die in Intel's 65nm node at 1.2V and 0.8V [Dighe10]
Environment
Sleep Power (mW)
Temperature (°C)
Psleep variation w/ temperature across 5 ARM Cortex M3 processor instances [wanner 13]

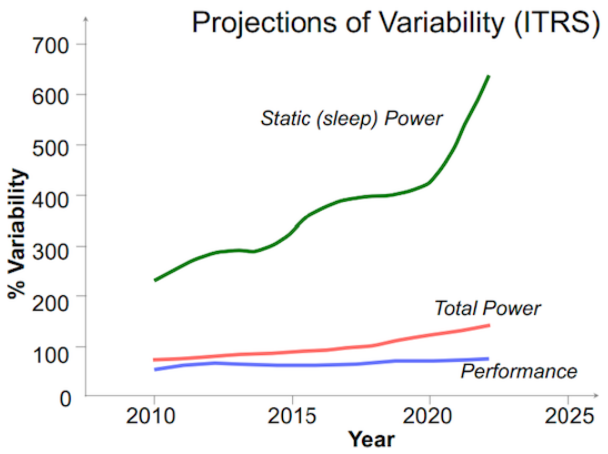
Figure 1.3: Variability induced challenges from three fronts in emerging SoCs.

1.1.3 Manufacturing Process Variations

The relentless scaling of CMOS technology has accelerated in recent years and will arguably continue toward the 10 nm regime. In the nanometer era, physical factors that previously had little or no impact on circuit performance are now becoming increasingly significant as predicted in Fig. 1.4 by ITRS [159, 158]. Particular examples include process variations, transistor mobility degradation, and power consumption. These new effects pose dramatic challenges to robust circuit design and system integration.



(a)

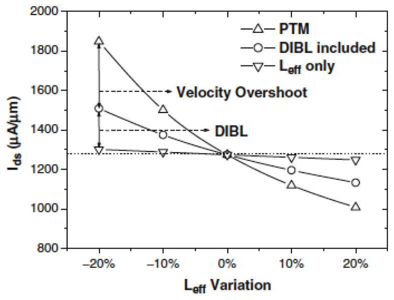


(b)

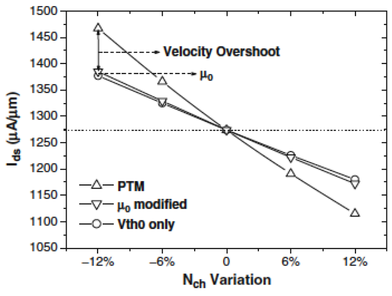
Figure 1.4: Variability predictions by ITRS (a) performance variability and (b) leakage and total power variability [159, 158].

1.1.3.1 Intrinsic Parametric Variability

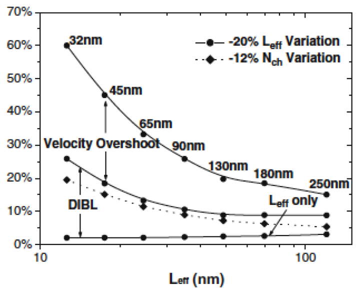
Spatial parameter variations in the device geometries in conjunction with temporal degradation and undesirable fluctuations in the operating condition may prevent the circuit from meeting desired performance and power constraints as discussed below:



(a)



(b)



(c)

Figure 1.5: Effect of variations on technology parameters (a) effective channel length L_{eff} on drain to source current I_{ds} (b) channel doping concentration N_{ch} on I_{ds}(c) overall impact on I_{ds} due to variation in L_{eff} at different technology nodes[65].

- **Channel Length Variation** L_{eff} is a form of *critical dimension* (CD) variation that dramatically affects performance (in terms of both delay and power). Since digital

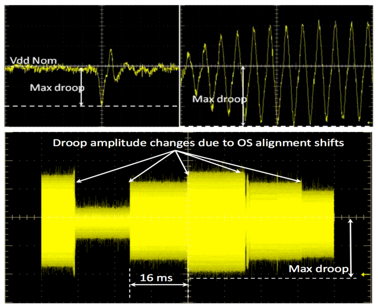
ICs typically utilize the minimum gate length allowed for a device, gate length is especially susceptible to variation consisting of a probabilistic component as well as spatially correlated (systematic) component [267]. Variation in L_{eff} affects designs in numerous ways including changes in the drain current I_D in all operating regimes (subthreshold, triode, and saturation) of the MOSFET characteristics and the V_{th} through drain-induced barrier lowering (DIBL) as shown in Fig. 1.5a. It also affects the gate-to-channel capacitance (C_{gc}), which loads the previous logic stage (modulating the previous stage's delay and dynamic power consumption) altering the propagation and rise/fall delays, leakage power consumption, and the delays and power consumption of its fan-in core. CD variations also impact the interconnect geometries modifying the capacitance and resistance of a given net. Variable interconnect capacitance affects both the coupling between nets, as well as the dynamic power consumption and delay of the gates driving those nets.

- **Threshold Voltage Variations :** The threshold voltage V_{th}, variation impacts fundamental MOSFET device behavior due to a probabilistic phenomenon (which is independent of other types of variation) known as *random dopant fluctuation* (RDF) caused by the random nature of ion implantation [17, 18]. With process scaling, the number of dopants has decreased dramatically and is only on the order of hundreds in modern-day devices [17, 237]. This fluctuation in channel dopants typically results in $\sim$50mV of V_{th} variation in today's MOSFETs [17, 237]. Unlike CD variation, the main component of the threshold voltage variations due to RDF is probabilistic and random in nature which is typically modeled as a Gaussian random variable characterized by its mean, μ, and standard deviation, σ [17, 237]. V_{th} variations influences number of MOSFET parameters including both circuit delay and leakage power. While the circuit delay is usually a linear or slightly super-linear function of V_{th}, leakage power is exponentially dependent on threshold voltage [68]. This exponential relationship between subthreshold current (and hence, leakage power) and V_{th} has become a major concern as leakage power is expected to surpass dynamic power as projected by ITRS in Fig. 1.4a.

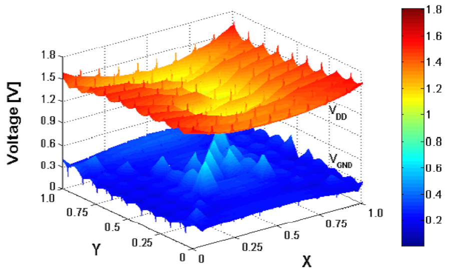
- **Gate Oxide Thickness Variations**: In state-of-the-art (sub-65nm) process nodes, the equivalent gate oxide thickness, t_{ox}, is less than *five silicon atoms thick* (and is in the order of $1nm$ for a silicon atom diameter of $\sim 0.2nm$ [19]). Thus, gate-to-oxide and oxide-to-silicon interfaces can exhibit significant amounts of oxide thickness variation (OTV) due to *atomic scale roughness* [19]. These variations are probabilistic in nature and can lead to variability in mobility, gate tunneling leakage current, and threshold voltage, among other parameters [19].

1.1.3.2 Voltage and Power Variations

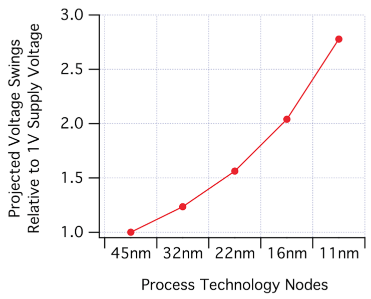
Voltage droops result from abrupt changes in the switching activity, inducing large current transients in the power delivery system (dI/dt voltage drops), and contain high-frequency and low-frequency components which occur locally as well as globally across the die as shown in Fig. 1.6a [191]. In addition to transient voltage droops, spatial variation in both the supply voltage and ground plane voltage is observed for a IBM power grid benchmark [359]. On the other hand, Fig. 1.7a shows transient power variation in each functional unit of the Alpha processor executing SPEC2k benchmarks with fast temporal variations. Power variability is also challenging due to spatial variation among instances of the same processor, for instance 13× variation in the sleep power across five instances of ARM Cortex M3 core was observed over a temperature range of $22 - 60\ ^oC$ [370].



(a)

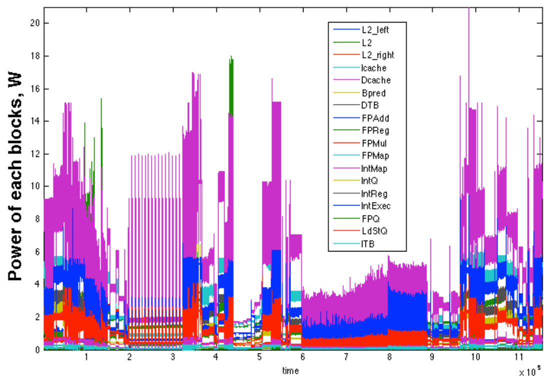


(b)

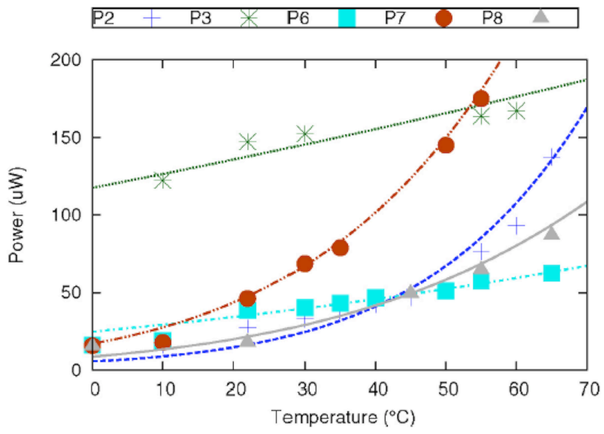


(c)

Figure 1.6: Voltage droops and variations (a) voltage droops in multi-core processors [191] (b) supply voltage and ground plane voltage variations [359] (c) voltage swings in emerging technologies [288].



(a)

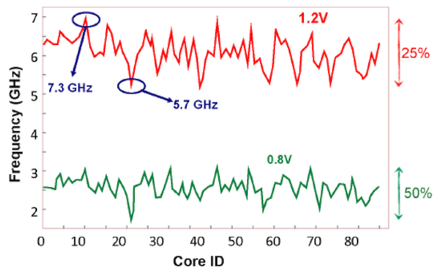


(b)

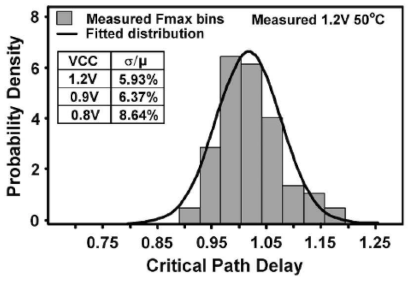
Figure 1.7: Power variations (a) transient power variation in each functional units of Alpha processor executing SPEC2k benchmarks (alphabetically) (b) power variability in ARM cortex M3 processor cores [370].

1.1.3.3 Gate, Path Delay, and Slack Variations

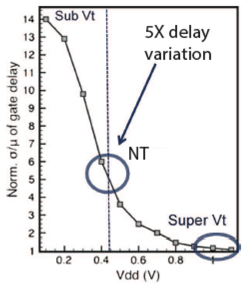
With-in-Die (WID) core-to-core maximum frequency ($Fmax$) variations as well as exaggerated variation in delay at near-threshold voltages are accepted reality in modern microelectronic manufacturing processes with geometries in nanometer scales. For an Intel 80-core processor in 65nm, Fig. 1.8a shows the WID core-to-core maximum frequency ($Fmax$) variations for each of the 80 cores. The measurements have been done at a fixed operating temperature of $50\ ^oC$ with three operating voltages: 1.2V, 0.9V, and 0.8V. At the nominal voltage of 1.2V, the fastest core displays the $Fmax$ of 7.3GHz while in the same die the slowest core can work with the $Fmax$ of 5.7GHz resulting in 28% WID clock frequency variation. Fig. 1.8b illustrates the delay distribution of the 80 cores for the same operating conditions where the single die exhibits an increasing value of σ/μ for lower voltages. Specifically, lowering the voltage from the nominal 1.2V to 0.8V, increases the critical paths variability (σ/μ) by 45% [96].



(a)



(b)



(c)

Figure 1.8: Variation in Emerging SoC Architecture (a) With-in-Die (WID) core-to-core maximum clock frequency variation for 80 cores on a single chip at 65nm [96] (b) Critical path delay distribution and its coefficient of variation (s/m) for 80 cores on a single chip[96] (c) Impact of voltage scaling on gate delay variation due to process variation [103].

For circuits working at near-threshold voltages, the statistical WID variation in the voltage threshold (V_{th}) plays an important role in determining the path delay. For example, Fig. 1.8c shows the normalized gate delay variation due to process variations as a function of V_{DD} where working at near threshold voltage of 400mV increases the performance variability by 5× compared to 1.3× at the nominal operating voltage [103]. Considering dynamic sources of variations, including temperature fluctuations, and voltage droops result in a total performance variability of 20× [103]. Similarly, variability is observed in relative delay and worst case runtime delay for different instances of the same chip running the same benchmarks, as well as different benchmarks as shown in Fig. 1.9 [286]. Wang et al. [367] reports that certain combinational logic can have a $5\times$increase in delay degradation depending on their input vectors.

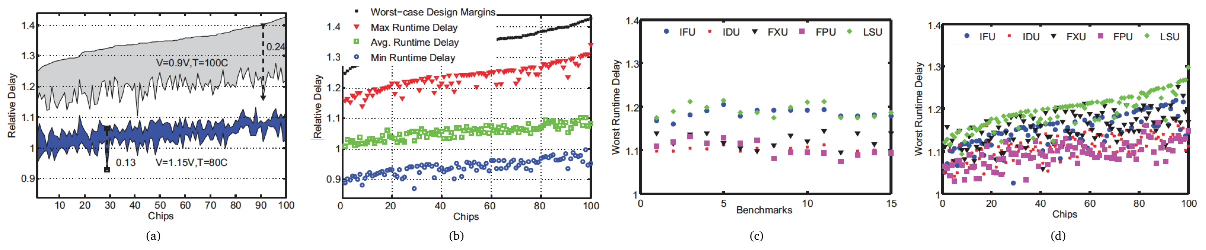


Figure 1.9: Timing Slack and Delay Variability (a) Spatial slack across processor units (b) Temporal slack between application phases (c) worst case circuit delay variations across benchmarks (d) worst case circuit delay variations across chips for same benchmarks [286].

1.1.4 Ambient and On-chip Thermal Variability

Environmental variations in ambient conditions are caused by fluctuations in operating temperature and supply voltage droops.

- **Thermal Variations:** Depending on the thermal conductivity, the dissipated power affects the temperature of a chip and devices as illustrated in Fig. 1.10. Power dissipation hence leads to global temperature variations as well as local fluctuations in regions of high-activity, so-called hot-spots. Additionally, ambient temperature variations lead to global shifts in chip temperature. Temperature fluctuations typically have time constants in the range of milliseconds to seconds [286]. An increase in temperature typically causes a circuit to slow down due to reduced carrier mobility and increased interconnect resistance, see Fig. 1.11. However, for low V_{DD} the circuit is operated in temperature inversion. Here, the effect of decreasing threshold voltage with temperature exceeds the mobility degradation, see Fig. 1.11a. Consequently, the circuit exhibits an inverted temperature characteristic, as it speeds up with increased temperature and vice versa, see Fig. 1.11b.

- **Variation in Hotspots:** Hot spots are a major concern in high-end processors as they constrain performance and limit the lifetime of semiconductor chips. To illustrate the magnitude of thermal variations, Fig. 1.10 shows infrared imaging captures thermal traces of a dual-core AMD Athlon II 240 and a dual-core Intel Core 2 Duo processor while running various CPU SPEC2006. The traces demonstrate that with-in-die thermal gradients can reach up to 16 $^{\circ}C$, and that differences in workloads can lead to strong variations in hotspot locations [284]. Variability in the hotspot locations requires smart sensor allocation techniques and various full thermal map characterization methods.

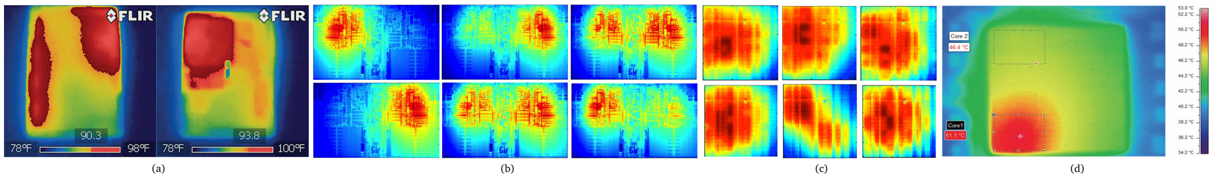


Figure 1.10: Thermal variations (a) mobile processor [380](b) AMD dual-core processor [251](c) AMD quad-core [284] (d) Xilinx Virtex 5 FPGA [14].

1.1.4.1 Thermal Effects on Failure, Aging, and Material Degradation

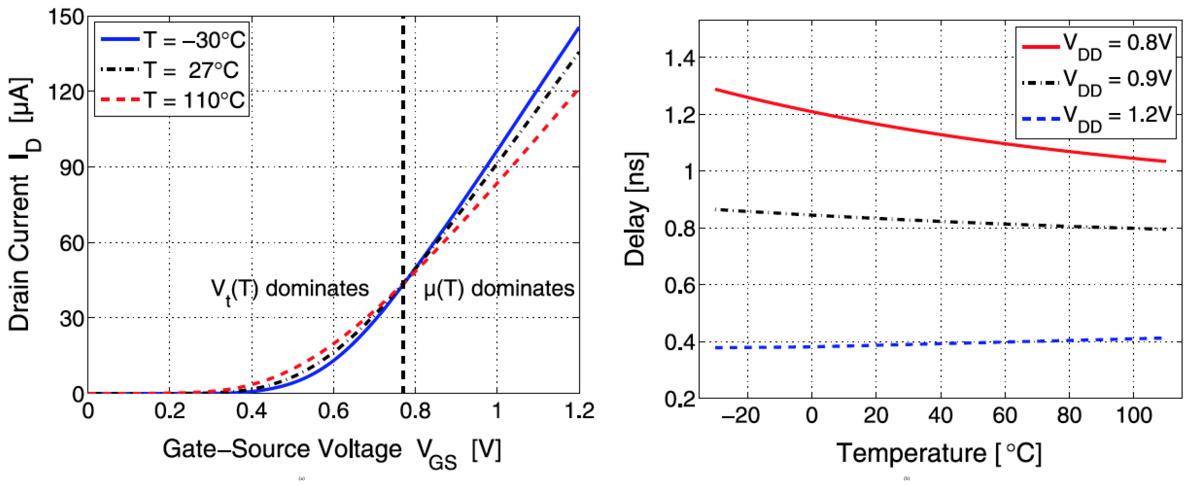


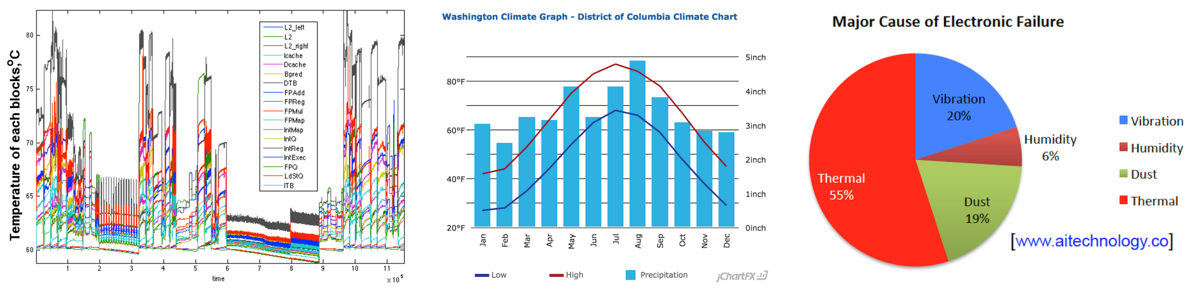
Figure 1.11: Influence of temperature on device characteristic and path delay (a) transfer characteristic of MOSFET (b) path delay for different voltages.

Effective thermal solutions are essential for reasons ranging from prevention of thermal runaway to maintenance of low skin temperature for mobile and wearable systems. However, due to increasing electrical fields and new materials, transistor wear-out is of increasing concern in recent technologies. BTI- as well as HCI-effects degrade the speed of transistors during their lifetime and demand for additional safety margin :

- Hot Carrier Injection (HCI) : Hot Carrier Injection mainly occurs during switching of logic gates. Carriers are accelerated in the lateral field under the oxide and gain sufficient kinetic energy to be injected into the gate dielectric. The trapped charge increases the threshold voltage of the device and reduces its current drivability.

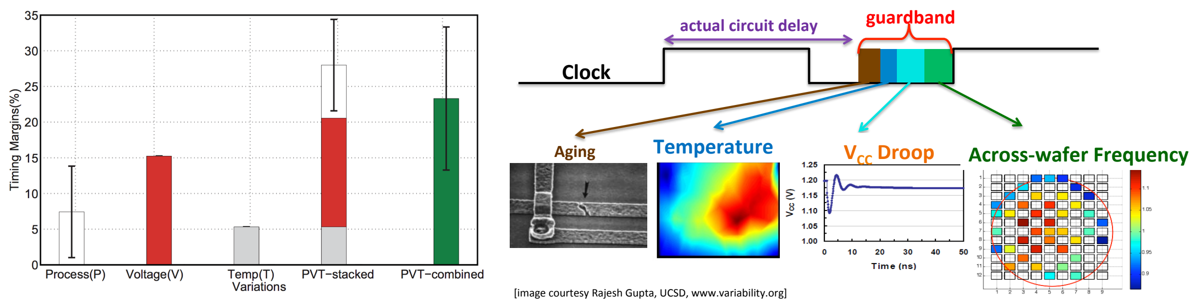
- Bias Temperature Instability (BTI): BTI results from high vertical fields and thus mainly occurs when a transistor is operated in triode mode (linear region), i.e., high V_{GS} and low V_{DS} is applied. Charge trapping by BTI for an n-FET is referred to as Positive Bias Temperature Instability (PBTI), while the term Negative Bias

Temperature Instability (NBTI) is used in the case of a p-FET. PBTI and NBTI increase the threshold voltage of the device and slow down the switching speed. BTI-aging is caused by charge trapping and detrapping with a wide range of capture and emission times. Therefore, small V_{th} -shifts can be observed already after very short stress times down to microseconds. However, due to the distribution of the capture and emission time constants, considerable V_{th} -shifts arise only after days, weeks or even years. NBTI alone can degrade circuit speed by upwards of 20% over a ten year period [366].



(a) (b) (c)

Figure 1.12: On-chip and ambient thermal variations (a) on-chip temporal thermal variations of Alpha processor functional units executing SPEC2k benchmarks (alphabetically) (b) yearly weather and ambient temperature variations (c) major cause of electric circuit failure.



(a) (b)

Figure 1.13: Impact of different sources of variations on (a) timing margins [286](b) guardband.

1.1.5 Application and Workload Variability

The level of Thread-Level Parallelism (TLP), Instruction-Level Parallelism (ILP), and Memory-Level Parallelism (MLP) varies across programs and across program phases due to workload phases [324, 325]. Within a single program TLP is defined as the number of concurrently active threads. A thread is active when it is not waiting for a synchronization event. TLP varies at run-time because of software requirements but also due to inter-thread synchronization. ILP is defined as the number of instructions executed in parallel, and MLP is defined as the number of memory requests issued in parallel. Within a single thread of execution, ILP and MLP often vary across programs and program phases because of the inherent structure of the programs and input set dependencies. Due to these workload variability, every application requires different underlying core microarchitectural resources for high performance and/or energy efficiency.

1.1.6 Complexity of Resource Management and Programming

With increasing integration of different heterogeneous core types and increasing number of cores per chip, while satisfying increasingly diverse and stringent cost, power, and performance constraints along with the key features of today's general-purpose programmability, hardware resource utilization and sharing will become a more complex issue. Better resource utilization (such as costly microarchitectural resources, I/O, and off-chip bandwidth) can improve aggregate system performance and enable lower-cost design alternatives, such as smaller die area or less exotic battery technology. Concurrently executing threads with greater hardware resource sharing in presence of diverse sources of variability, not only amplifies the programming complexity with system size, but also exaggerates the programmer burden by posing additional responsibility of computation organization and optimization of complex applications [141, 180].

1.2 Tackling Emerging SoC Design Challenges

Computing systems face unprecedented levels of system dynamics exaggerated by different sources of variability and dramatic changes in CMOS technology scaling. As a result, computing systems must respond to the new realities both in architecture and software. Radical approaches are needed that call for a major shift in architecture, software, and whole-stack design. We consider the following radical approaches that may revolutionize how computers are designed in order to address the emerging challenges and limitations.

1.2.1 Whole-Stack Co-Design and Optimization

The recent generation of transistor device scaling have produced limited benefits in transistor speed and power. Aggressively scaled technologies face more stringent design constraints and introduce worst-case design margins or guardbands to tackle increasing process variability. State-of-the-art guardband design methodologies assume worst-case environmental factors and minimum feature size that result in overly conservative decisions and inevitable deterioration in the yield. For example, in today's 32nm node, a 20% voltage margin translates to a 33% frequency degradation, and at future technology nodes the situation gets much worse. Currently, the state-of-the art guardbands methodology tends to accumulate margins (i.e., overheads) as design closure is performed using a multi-corner analysis, with an increasing number of corners [20]. The impact of guardbanding on the key design metrics (power, performance, and area) has been steadily increasing with technology scaling [158], leading to loss of operational efficiency and increased costs due to over-design. Consequently, existing state-of-the-art guardbanding system technologies have hit major obstacles.

In addition, the future of computing faces formidable challenges, especially with the slow-down of traditional integrated circuit scaling. The slowdown of silicon CMOS (Dennard) scaling has prompted comprehensive research on faster, more reliable and energy-efficient switches. However, better switches alone will not deliver the necessary leaps in performance. Instead of focusing solely on improving transistors or memory cells, an integrated approach that shifts from transistor-scaling-driven performance improvements to a new post-scaling whole-stack co-design would be the key to improved efficiency and promises of significant benefits in performance, energy efficiency, and cost. An end-to-end approach is essential, as substantial improvements are generally rare, and cannot be achieved with uncoordinated improvements in architectures, transistors, or memory cells alone but through symbiotic relations among components to enhance key performance metrics. Over-optimizing a single component without considering the full-system may not improve the system margin, as components from other layers may be the bottleneck. Improving each component/layer of the system stack where each component/layer shows comparable improvement can result in synergies. Careful co-design and co-optimization of software and hardware can enable higher efficiency and performance. For example, faster memory accesses cut core idle times, reducing energy consumption and overall execution time [89]. When combined with increased memory bandwidth and improved memory locality that enable many concurrent memory accesses and reduce memory access contention significantly, performance speedup and energy efficiency is improve appreciably [37]. On the other hand, comprehensive improvement in the layers of the stack through (scale-out and scale-up) parallelism and near-threshold voltage scaling with tolerance of variability across the layer can provide the necessary leaps in performance and energy efficiency [103]. This thesis advances and promotes an integrated approach spanning emerging logic devices and memories, computer architecture, thermal solutions, and synergistic runtime software.

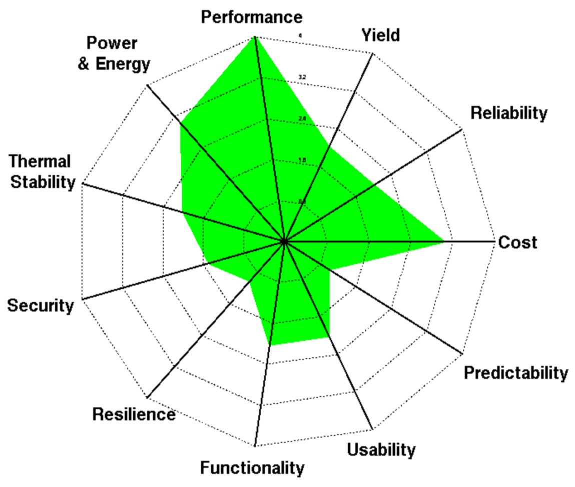
1.2.2 Principled Approach to Cross-Layer Awareness

In order to advance whole-stack co-design and optimization at a reasonable development cost and time, cross-layer awareness both at design time as well as run-time is key to enabling this objective. Awareness is critical because an un-aware computing system can never respond to user and system goals by being oblivious to system state, behavior, and environment. For instance, because of the lack of accurate awareness of the design margins, pessimistic assumptions incurring significant overheads are made across the layer during hardware design time. With the availability of certain system/architecture-level awareness information, overhead and margin can be reduced by relaxing some of these assumptions. Additionally, system design margins may be dominated by the existence of certain pathological scenarios caused by behaviors from different layers of system abstraction, for example, certain applications are found to be faster in simple cores that are design for energy efficiency than complex faster cores that are design for performance [187]. Contrary to the conventional belief, complex cores do not enhance the performance for all applications, neither do the simple cores save energy for all applications. Consequently, cross-layer awareness is required as awareness of a single-layer may be inadequate to avoid these scenarios during design optimization approach. Furthermore, some hardware-related phenomena are difficult to monitor by hardware monitors alone. For example, timing and soft error rate are extremely difficult to monitor due to their rare occurrence nature and dependence on ambient characteristics. Cross-layer awareness approaches can use system and cross-layer information for better monitoring and awareness to investigate and address several limitations of current SoC design.

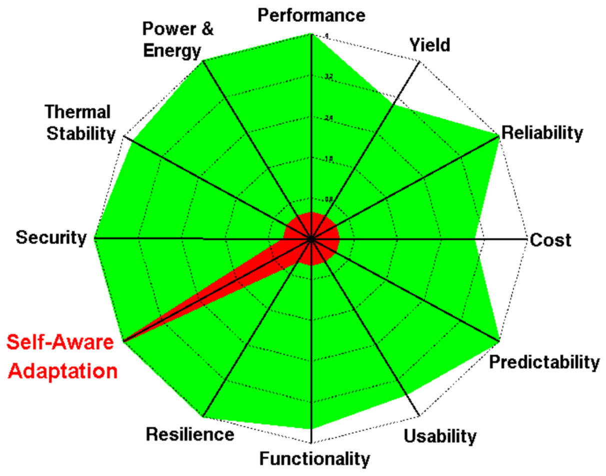
A fundamental step in the awareness of the whole-stack of the system is cross-layer monitoring and cross-layer modeling in a principled way. Cross-layer analytical models predict chip key metrics by capturing key technology trends, design constraints, workload characteristics and microarchitectural variables into a mathematical framework. Models that are accurate, insightful, and easy to implement will become crucial in evaluating many-core chip designs of the future. Principled approaches to tractable, accurate and insightful analytical cross-layer modeling are crucial in order to understand, evaluate, and optimize the whole-stack key system design metrics of many-core chip designs of the future.

1.2.3 Transitioning to Closed-loop Adaptive SoC Design

It is a well established fact [125] that worst case massive over-design through guardbands in response to increasing levels of random and systematic variations undercut the gains from process and device scaling. This design paradigm is steadily approaching the point of diminishing return. There is an increasing incentive for a transition from statically configured crash-and-recover design to adaptive design where the adaptive hardware provides the capability to alter itself to dynamically match software and user de-



(a)



(b)

Figure 1.14: Dimensions of Awareness for emerging SoC.

mands, environmental characteristics, and physical defects. The fundamental premise behind adaptive SoC design is the recognition that massive variations in manufacturing and environment cause a *statically configured operating point* to be far too inefficient. Inefficient designs are costly, waste power and performance, and will quickly be surpassed by more adaptive designs, just as it happens in the biological realm. Organisms must adapt to survive, and a similar trend is seen with processors and SoCs – those that are enabled to adapt to their environment, will be far more competitive. However, to achieve this objective, the adaptive SoC needs to be made aware of its environment and operating conditions through the use of various sensors and then have the ability to respond meaningfully to the sensor stimuli.

Even though the capability to adapt is enticing, it is extremely hard to achieve a generalized scheme with low overhead at the system level. This is partly because of the often unknown interdependencies and interactions, wide operating range of inputs and design parameters, and the computational cost of determining the adaptive behavior at the system level with extremely large number of parameter configurations. However, by devising assisted and online dynamic adaptation, emerging paradigms of the SoC, such as Multiprocessor-Systems-on-Chip (MPSoC), can sense and learn their own physical parameters, adapt with fine / coarse granularity in the field, as well as adjust and re-implement their design goals / intentions on-the-fly using on-line or off-line learning information for higher efficiency of a design metric. Traditionally, SoCs were designed and dynamically optimized for a single design metric, for instance, performance or power efficiency while compromising and/or trading-off other dimensions as illustrated in Fig. 1.14(a). The radar chart in Fig. 1.14(a) shows a specific template of the design that is tuned for performance. It is not designed or capable of reaching higher value of design metric for other dimensions of the design (for example, resilience). On the other hand, with increasing need for power/energy efficiency, thermal stability, and resilience, multiple design dimensions and metrics are required to be optimized and adapted across the system layers. It is not enough for emerging SoCs to be performance efficient, but need have the capability to be efficient in other design dimensions (for example, energy efficiency and resilience). In order to achieve these objectives and optimize multidimensional design metrics, Fig. 1.14(b) shows the need for awareness of multiple dimensions in emerging SoCs and transitioning to self-aware adaptation as a key design dimension to achieve multiple, often conflicting, design goals. This transition enables changes in computing model, architecture, and the runtime system to achieve multiple conflicting goals and leverage the trade-off space.

1.2.4 Rethinking MPSoCs as Cyber-Physical Systems

Engineering variability challenges and the struggle to control variations in MPSoCs with exploding number of computing cores [51, 165] calls for a rethinking of these computational devices and their architectural design paradigm. With an increasing tilt towards adaptive SoC design and influence of physical parameters dynamics, MPSoCs bear several similarities with cyber-physical systems that are real-time controlled and coordinated systems relying on computational infrastructure. For example, both these types of systems have complex functional specifications, demanding non-functional specifications, multi-modal behaviors, as well as use of networks in their design. Even though the operational life-time and timeline of these designs are very different, design of both involves a control/computing co-design of closed-loop plant with computer systems in the loop. The characteristic of the plant specially varies, the former being computation and the later being a physical system, with disproportionate scale (e.g., one distributed in the nano-scale and the other geographically distributed). However, MPSoC's physical characteristics such as thermal- power dynamic behaviors, the cooling system, and the modeling approach can greatly benefit from techniques and design approaches in cyber-physical systems. Consequently, this thesis proposes the foundations for a new kind of SoC paradigm that unifies and emphasize the need for a rethinking of the whole computing stack of the future.

1.3 Thesis Contributions

The theme of this dissertation provides the foundation for a new class of *self-aware adaptive SoC* called Cyber-Physical-Systems-on-Chip (CPSoC) that contributes toward the whole-stack co-design of emerging MPSoCs by using the notion of self-aware adaptation through a tightly coupled optimization of control, communication, and computing in order to achieve competing design and runtime goals (e.g., boosting energy and power efficiency, improving thermal resilience, and delivering greater performance). The dissertation presents a new paradigm through the principled design and implementation of awareness and adaptation mechanisms in emerging MPSoCs (e.g., mobile SoCs and heterogeneous MPSoCs) to deliver greater performance, energy efficiency, and resilience by adopting *self-aware control theoretic cross-layer mechanisms*. The dissertation provides the foundation for CPSoC through the development of closed loop modeling, architecture, algorithm, system software, and platform that makes it possible to adapt the system stack. The dissertation outlines research spanning five research themes and contributes to the respective areas by bringing fresh perspective and opening new design space as described below:

- **CPSoC Paradigm.** The thesis presents and builds the foundation of a new class of self-aware SoC called Cyber-Physical-Systems-on-Chip (CPSoC) and contributes toward the principled design and implementation of awareness and adaptation mechanisms for whole-stack co-design and optimization. The thesis demonstrates that giving the SoC the freedom to opportunistically adapt the software and the hardware stack by infusing steering knobs and awareness mechanism across the stack can open up new and otherwise untapped opportunities in energy efficiency, performance, and thermal resilience.

- **Architecture:** The thesis focuses on a radically new architecture for efficient self-awareness and control of emerging heterogeneous MPSoCs in presence of increasingly unreliable manufacturing process by transitioning from a run-and-crash worst case design paradigm to sense-predict-adapt paradigm for multiple design metrics. The proposed architecture is similar in spirit as the UnO machines [125] but distinctly differs by incorporating a self-aware execution model, distributed sensing and actuation infrastructure, and heterogeneity of computing elements. The dissertation develops critical architectural support such as dedicated multi-sensor and actuator networks-on-chip, suitable abstractions to expose awareness properties to the system software and other layers, and architectural techniques that navigate these complex trade-offs to reduced design overheads. An efficient *cross-layer* design space exploration of these emerging heterogeneous architectural configuration and compositions with system level design goals is formulated to reduce exploration time by three orders of magnitude [307, 308].

- **Models:** Emerging MPSoC architectures are difficult to model precisely because they are non-deterministic, interactive, and expected to scale to thousands of cores in the presence of an inherently unreliable silicon substrate [53, 51]. The dissertation work builds predictive machine learning models of performance, power, and thermal dynamics for capturing online interactions of these different goals considering the cross-layer (software and hardware) characteristics [307, 313, 244]. Use of such an integrated and end-to-end approach provides the opportunity to trade model complexity with accuracy by using dimensionality reduction and approximation techniques [341, 304] which ensures that the model fidelity is general enough to cover all the possible inputs and complex enough to express required features and behaviors. On the other hand, by using control theoretic dynamical approach, these models are generalized and exploited to identify suitable actuation knobs by using unique features at individual layers of the stack [306, 310]. The approach is extendable to incorporate environmental models such as on-chip noise and variability while providing formal guarantees on convergence, stability, optimality, and robustness [310, 306]. This control theoretic approach assures formal reasoning about system dynamics of the CPS plant even when the response is not fully understood in the closed loop context which is uniquely suitable for the challenges of emerging SoCs.

- **Algorithms:** The dissertation presents a new class of scheduling and balancing algorithm that incorporates the dynamics of the physical plant in the scheduling and balancing scheme to optimize multiple objectives and goals of the platform (e.g., performance, power, and energy efficiency) [313, 244]. Unlike traditional approach, this bridges scheduling / balancing theory with control systems dynamics of the MPSoC plants. The dissertation contributes to the design and synthesis of efficient multi-sensor placement and heterogeneous sensor fusion algorithms respectively using sparse and compressive techniques that reduced the area-power overhead of monitoring scheme by an order of magnitude and improves accuracy by two orders of magnitude [341].

- **System Software:** The thesis contributes toward the design and extension of the Linux operating system with self-awareness abstractions, adaptive reflexive middleware, with predictive models to achieve system level goals such as energy efficiency [313, 244]. The approach can be generalized to support multiple mix of goals. By reusing most of the Linux kernel source with minimal changes along with a light-weight run-time scalable global optimization engine, the energy efficiency of both application and operating system (OS) workloads can be improved by $\sim$50% for quad core heterogeneous multicore processors (HMP) [313, 244].

- **Platform and Testbeds:** This research creates a new CPSoC prototyping platform using FPGAs [305], develops full-system simulation framework and tools for heterogeneous MPSoC architecture in order to validate and demonstrate on-line monitoring, modeling, adaptation, and system software efforts [313, 341].

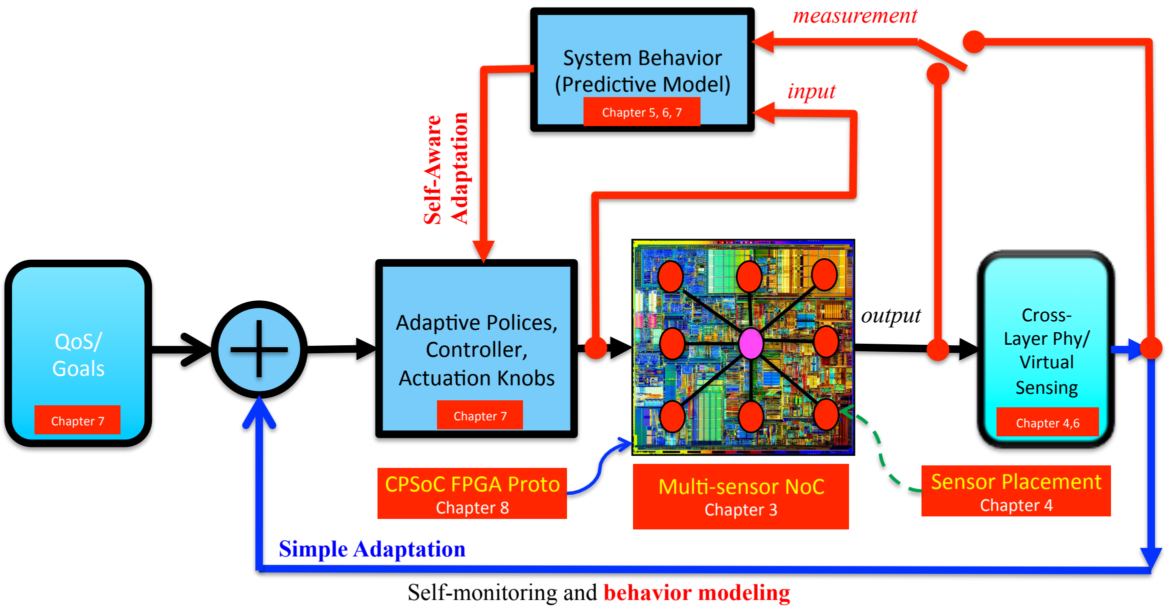


Figure 1.15: Thesis overview and scope.

1.3.1 Thesis Organization

The thesis is organized in eight chapters describing the key aspect of the CPSoC features as illustrated in Fig. 1.15. Chapter 1 gives a brief introduction and overview of the thesis. Chapter 2 provides an overview of the related work and the background of the thesis followed by Chapter 3 introducing and illustrating the concept of CPSoCs. It also discusses the rationale behind the CPSoC paradigm and architectural features and supports that distinguish CPSoCs from traditional MPSoCs. Chapter 4 presents on-chip sensor networks, placement, and fusion as an architectural support for self-awareness features in CPSoC supported by predictive models and model building capabilities in Chapter 5. Chapter 6 describes a minimal sensing approach to runtime system state estimation and prediction. The use of sensor-NoC (sNoC) architectural support along with predictive models for awareness is demonstrated by a smart system software and runtime to make intelligent decisions using cross-layer actuation mechanisms (for example, smart balancing) in Chapter 7. In order to achieve a realistic implementation of the CPSoC paradigm, prototyping and emulation is performed in FPGA platforms in Chapter 8. Chapter 9 presents the future research directions along with dissertation summary and conclusions.

Chapter 2

Background and Related Work

This chapter presents the relevant background, prior work, and trends in emerging SoC that falls into several categories: classification of microarchitecture, smart SoC architecture, self-aware, self-adaptive, autonomic systems, on-chip sensing for self-awareness, awareness modeling, and self-aware architectures.

2.1 Taxonomy and Classification of Microarchitectures

Processor microarchitectures can be classified along multiple orthogonal dimensions. Here we present the most common ones and then distinguish them with respect to emerging self-aware architectures.

- **Pipelined/Non-pipelined Processors** : Pipelined processors split the execution of each instruction into multiple phases and allow different instructions to be processed in different phases simultaneously. Pipelining increases instruction level parallelism (ILP), and due to its cost-effectiveness, it practically is used by all processors nowadays.

- **In-Order/Out-of-Order Processors** : An in-order processor processes the instructions in the order that they appear in the binary (according to the sequential semantics of the instructions), whereas an out-of-order processor processes the instructions in an order that can be different (and usually is) from the one in the binary. The purpose of executing instructions out of order is to increase the amount of ILP by providing more freedom to the hardware for choosing which instructions to process in each cycle. Obviously, out-of-order processors require more complex hardware than in-order ones.

- **Scalar/Superscalar Processors** : A scalar processor [166] is a processor that cannot execute more than one instruction in at least one of its pipeline stages. In other words, a scalar processor cannot achieve a throughput greater than one instruction per cycle for any code. A processor that is not scalar is called superscalar. On the contrary, a superscalar processor can execute more than one instruction at the same time in all pipeline stages and therefore can achieve a throughput higher than one instruction per cycle for some codes. Very-long-instruction-word (VLIW) processors are a particular case of superscalar processors, which can process multiple instructions in all pipeline stages, so they meet the definition of superscalar. What makes a superscalar processor to be VLIW are the following features: (a) it is an in-order processor, (b) the binary code indicates which instructions will be executed in parallel, and (c) many execution latencies are exposed to the programmer and become part of the instruction-set architecture, so the code has to respect some constraints regarding the distance between particular types of instructions to guarantee correct execution. These constraints have the purpose of simplifying the hardware design since they avoid the inclusion of hardware mechanisms to check for the availability of some operands at run time and to decide which instructions are issued in every cycle. For instance, in a VLIW processor that executes 4 instructions per cycle, the code consists of packets of 4 instructions, each of them having to be of certain types. Besides, if a given operation takes three cycles, it is the responsibility of the code generator to guarantee that the next two packets do not use this result. In other words, in a non-VLIW processor the semantics of a code are determined just by the order of the instructions, whereas in a VLIW processor, one cannot totally derive the semantics of a code without knowing some particular features of the hardware (typically the latency of the functional units). By exposing some hardware features as part of the definition of the architecture, a VLIW processor can have a simpler design but, on the other hand, makes the code dependent on the implementation, and thus, it may not be compatible from one implementation to another.

- **Vector Processors** : A vector processor is a processor that includes a significant number of instructions in its ISA (instruction set architecture) that are able to operate on vectors [196]. Traditionally, vector processors had instructions that operated on relatively long vectors. More recently, most microprocessors include a rich set of instructions that operate on relatively small vectors (e.g., up to 8 single-precision floating point (FP) elements in the Intel AVX extensions [149]). These instructions are often referred to as SIMD (single instruction, multiple data) instructions. According to this definition, many processors nowadays are vector processors, although their support for vector instructions varies significantly among them.

- **Multiprocessor** : Multiprocessor consists of multiple central processing units (CPUs) tightly coupled enough to cooperate on a single problem. A multiprocessor is a computer system having two or more processing units (multiple CPUs) each sharing main memory and peripherals, in order to simultaneously process programs. At the operating system level, *multiprocessing* is sometimes used to refer to the execution of multiple concurrent processes in a system, with each process running on a separate CPU or core, as opposed to a single process at any one instant. When used with this definition, *multiprocessing* is sometimes contrasted with *multitasking*, which may use just a single processor but switches it in time slices between tasks (i.e., a time-sharing system). Multiprocessing, however, means true parallel execution of multiple processes using more than one processor. Multiprocessing doesn't necessarily mean that a single process or task uses more than one processor simultaneously; the term *parallel processing* is generally used to denote that scenario. *Multitasking*, on the other hand, is a concept of performing multiple tasks (also known as processes) over a certain period of time by executing them concurrently. To avoid confusion, operating system techniques are preferably referred as *multiprogramming* and reserve the term *multiprocessing* for the hardware aspect of having more than one processor. *Multitasking* does not necessarily mean that multiple tasks are executing at exactly the same time (concurrently). In other words, multitasking does not imply parallel execution, but it does mean that more than one task can be part-way through execution at the same time, and that more than one task is advancing over a given period of time.

- **Multicore Processors** : Also known as *chip multiprocessor* (CMP) [256], is a processor that may consist of one or multiple cores on a single processor chip. A core is a unit that can process a sequential piece of code (usually referred to as a thread). Traditional processors used to have a single core, but most processors nowadays have multiple cores. A multicore processor can process multiple threads simultaneously using different hardware resources for each one and includes support to allow these threads to synchronize and communicate under the control of the programmer. This support normally includes some type of interconnect among the cores and some primitives to communicate through this interconnect and often to share data and maintain them coherently.

- **Multithreaded Processors** : A multithreaded processor is a processor that can execute simultaneously more than one thread on some of its cores. Both multicore and multithreaded processors can execute multiple threads simultaneously, but the key distinguishing feature is that the threads use mostly different hardware resources in the case of a multicore, whereas they share most of the hardware resources in a multithreaded processor. Fig. 2.1 depicts the difference between a conventional versus a multithread process. Multicore and multithreading are two orthogonal concepts, so they can be used simultaneously. For instance, the Intel Core i7 processor has multiple cores, and each core is two-way multithreaded. Fig. 2.2 illustrates different approaches of multithreading in contemporary architectures.

- **GPGPU** : A general-purpose GPU (GPGPU) is a graphics processing unit (GPU) that performs data parallel calculations that would typically be conducted by the CPU (central processing unit). In other words, General-purpose computing on graphics processing units (GPGPU), the use of a graphics processing unit (GPU), which typically handles computation only for computer graphics, to perform data parallel computations in applications traditionally handled by the central processing unit (CPU) [114, 113]. The use of multiple graphics cards in one computer, or large numbers of graphics chips, further parallelizes the already parallel nature of graphics processing. In addition, even a single GPU-CPU framework provides advantages that multiple CPUs on their own do not offer due to the specialization in each chip [239]. While GPUs operate at lower frequencies, they typically have

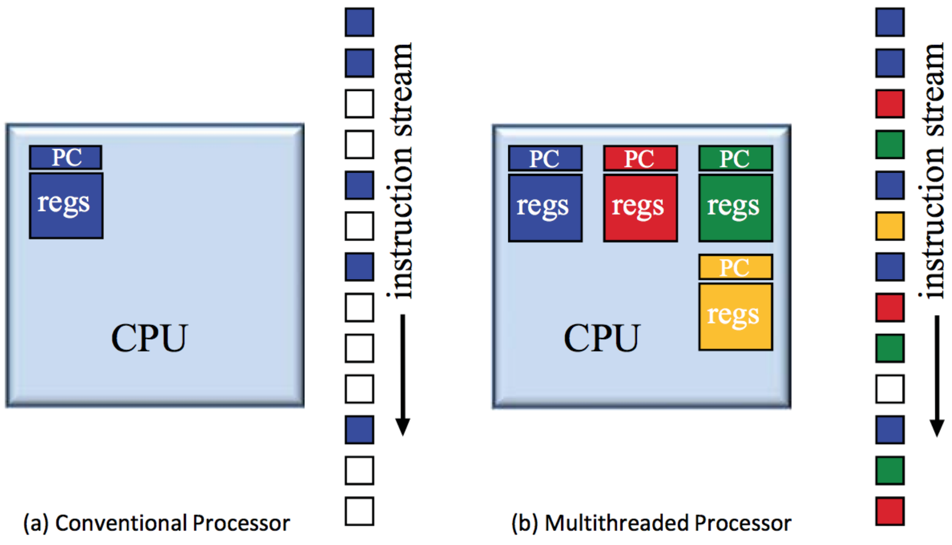


Figure 2.1: A conventional processor compared with a multithreaded processor [249].

many times the number of cores and operate effectively on data parallel applications much faster than a traditional CPU.

- **Manycore Processors** : Multicore architectures integrate multiple processor cores in a System-on-Chip (SoC) [256]. The terms *manycore* and massively *multicore* are often used to refer to multicore architectures with tens to hundreds of cores [51]. The term manycore distinguishes scalable architectures designed specifically to improve application performance and efficiency by distributing parallel computations across large numbers of highly integrated cores. Manycore architectures are specifically designed to exploit data-level and task-level parallelism within applications. They focus on delivering massive numbers of functional units efficiently, and support massive amounts of parallelism. Unlike the communication primitives found in multicore processors, which descend from architectures designed to support multi-chip multiprocessors typically comprising from 2 to 16 processors, manycore architectures integrate scalable communication systems that exploit the low communication latencies and abundant communication bandwidth that are available on chip. Consequently, manycore architectures deliver significantly more arithmetic bandwidth than multicore processors, and usually achieve superior area and energy efficiency when handling applications with sufficient structure and locality.

- **MPSoC Architectures** : The MultiProcessor System-on-Chip (MPSoC) is a System-on-a-Chip (SoC) which uses multiple processors (see multi-core), combines embedded processors, specialized digital hardware, and often mixed-signal circuits to provide a complete integrated system, usually targeted for embedded applications.

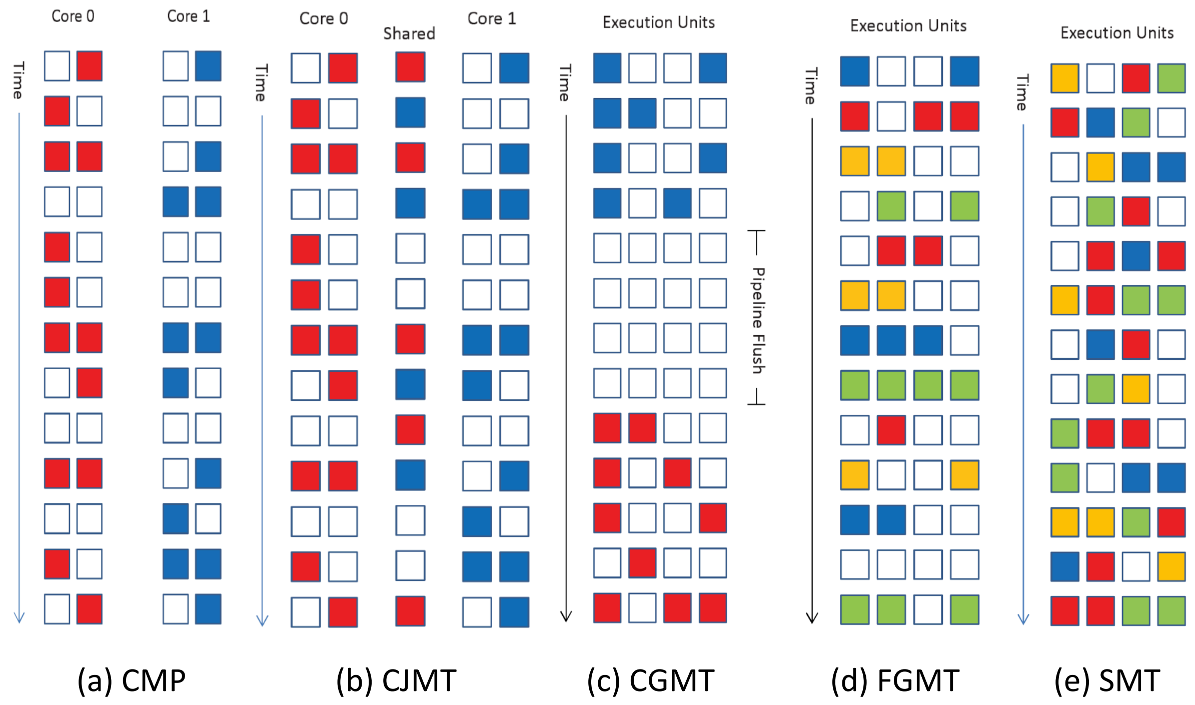


Figure 2.2: Chip architecture and different approaches of multithreading (a) chip multiprocessor (b) conjoined multithreading (c) coarse-grained multi threading (d) fine-grained multithreading (e) symmetric multithreading [249].

It is used by platforms that contain multiple, usually *heterogeneous*, processing elements with specific functionalities reflecting the need of the expected application domain, a memory hierarchy (often using scratchpad RAM and DMA) and I/O components. All these components are linked to each other by an on-chip interconnect. MPSoCs make the most sense in high-volume markets that have strict performance, power, and cost goals. These architectures meet the performance needs of multimedia applications, telecommunication architectures, network security and other application domains while limiting the power consumption through the use of specialized processing elements and architecture. For instance, embedded computer vision is one example of an emerging field that can use essentially unlimited amounts of computational power but must also meet real-time, low-power, and low-cost requirements. Table 2.1 lists some of the examples of contemporary multi/manycore processors specifically targeting embedded system applications.

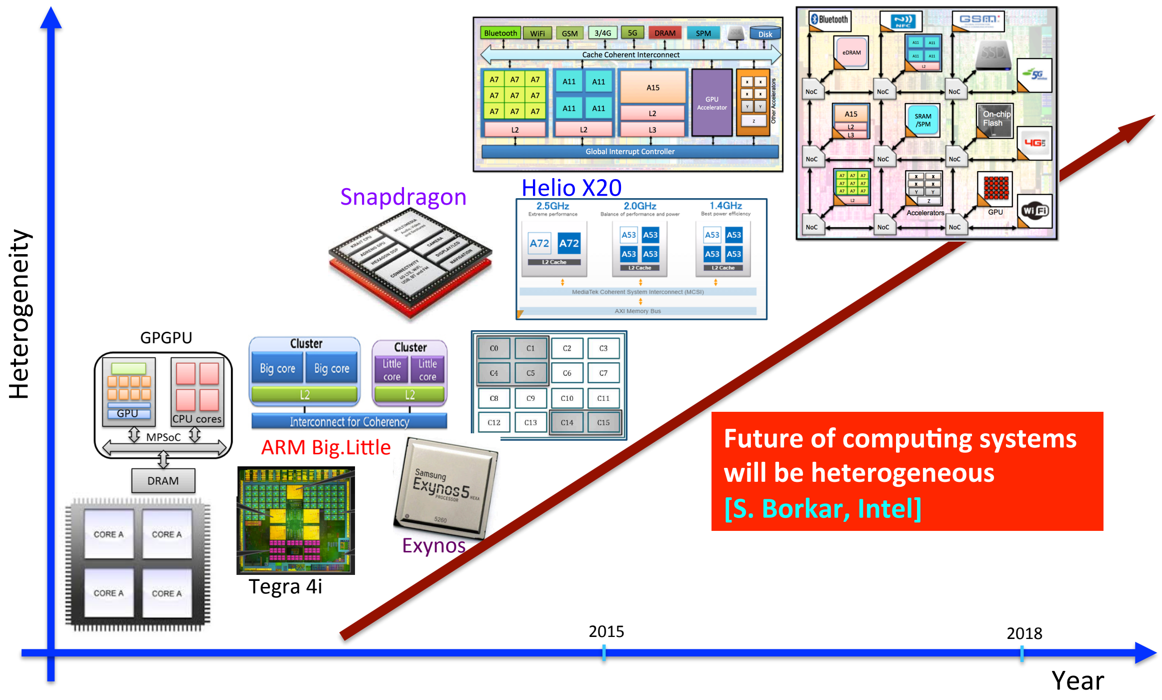


Figure 2.3: Heterogeneity trends in emerging SoCs. S. Borkar of Intel asserts that future of microprocessors will be heterogeneous [53].

2.2 Emerging MPSoC and Embedded System Trends

Integrated circuit technology has reached the nanoscale era, introducing a multitude of challenges stemming from the end of perfect Dennard scaling [107, 280, 111] and worsening process variations [54]. "Dark silicon" will be a defining feature of future SoCs, where only small portions of a chip may be powered on at a time in order to manage power density and heat [107]. Further complicating matters, systems are typically not very energy proportional due to high static power and a dearth of active low-power modes aside from CPU dynamic voltage/frequency scaling (DVFS) [30, 29]. The dark silicon phenomenon and the need for greater energy proportionality and efficiency are major driving forces behind research and development in many-core SoCs and heterogeneous devices, architectures, and systems. In particular, many-core computational platforms already face significant resiliency challenges with errors resulting from manufacturing process variability, exponentially increasing power dissipation and heating, environmental effects (e.g., radiation induced soft errors [35], and aging/wearout [43]). These problems are exacerbated in the nanometer era with exploding core counts and on-chip resources. The combined systemic and random effects in nanoscale technologies results in high variability (and thus higher error rates) manifested at the circuit level all the way to the architecture and system levels [54], requiring new strategies for ensuring application resilience when executing on these computing platforms. Furthermore, emerging embedded computing platforms that deploy complex SoCs will be characterized by the following key features that provide both challenges and opportunities for simultaneously managing system resilience, energy, and adaptivity:

- They will see much larger fault rates. More integration results in larger platforms facing more dominant failure mechanisms (with technology scaling), causing increased fault rates [43]. This is especially true for memories in emerging data-centric platforms [246, 333].
- They will be monitor-rich. To assess the state of health of the system, these computing systems will employ a network of interconnected monitors looking for signatures of faults, wearout and impending failures [112, 194, 215]. These monitors will span circuit, microarchitecture and software layers.
- They will be aggressively heterogeneous in the computing fabric, covering all dimensions: processing (for accelerating application/domain-specific functions) [53], interconnect (to handle scalability and high-throughput) [318], and memory (combining volatile and non-volatile storage, e.g., [95]) as shown in Fig. 2.3.
- They will be memory-heavy, and will deploy heterogeneous memory technologies. Data-centric nature of several emerging applications creates demand for denser memories. Memories are likely to dominate energy as well as reliability concerns [246, 333] for computing systems. Moreover, technology trends such as 3D integration [52] and heterogeneous memory organizations (e.g., combining traditional SRAMs with emerging faster, denser, non-volatile memories) [379] pose new challenges for energy efficiency and resilience.

2.3 Intelligence and Awareness for Smart Embedded System

In this section, we present the notion of smart embedded system with attributes of intelligence and awareness based on the work in [104].

The ubiquitous deployment of computing in virtually every facet of today's society has led to the colloquial usage of terms as such "embedded computing", "cyber-physical systems", and more recently the "Internet-of-Things (IoT)". At the heart of such systems are software/hardware computing platforms that interact with the physical world through sensors, actuators, communication/networking, and decision making engines. These systems range from the tiniest of embedded devices (e.g., small sensor motes [131, 273, 108, 342, 328, 344]) to complex system-of-systems, such as autonomous swarms of robots [295, 296] and complex human-in-the-loop systems (e.g., an Airbus

Table 2.1: Examples of Contemporary Embedded Multi/Manycore Processors.

Processor	Architecture	Cores	Speciality	References
RAW	Tiled multicore	16 simple RISC	◦ Expose low-level resource to software	[351]
		Mesh Network	◦ Distributed register architecture	[352]
			◦ StreamIt Programing Language	[355]
			◦ Exploit task & data-level parallelism	[212]
			◦ Space-time scheduling	
Tilera	Tiled multicore	64-bit	◦ Commercial version of RAW	[38]
		3-way wide	◦ Sub-word SIMD operations	
		VLIW	◦ Software-routed scalar operand networks	[373]
		Multiple Mesh Network	◦ General-purpose dynamically routed networks	
			◦ Fine-grain pipeline parallelism	
			◦ Improve the efficiency	
Berkeley VIRAM	Vector Processor	4 no of 64-bit vector lanes	◦ Uses embedded DRAM technology	[196]
	Processor-in-Memory	Vector length of 32	◦ General-purpose scalar control processor	
			◦ Vector co-processor	
			◦ Issue individual vector instructions in order	
			◦ Operate on sub-word SIMD operands within each lane	
			◦ Efficient with regular data-level parallelism	
			◦ Vector flags, compress, and expand operations	
			◦ Allow data-parallel loops with conditions	
Stanford Imagine	Stream processor	8 data-parallel lanes	◦ Several data-parallel execution units	[183]
		7 function units	◦ Multiple lanes exploit SIMD data-level parallelism	
			◦ Function units within lane exploit ILP	
			◦ Uses a micro-sequencer to sequencing of kernels	
SPI Storm Stream Processors	Stream processor	16 data-parallel lanes	◦ Commercial version of Stanford Imagine	[184]
MIT Scale Processor	Vector-thread processor	4-wide VLIW execution	◦ Supports both vector and multithreaded processing	[198]
		control processor	◦ Efficient regular control-flow and data accesses	[197]
		parallel vector lanes	◦ Common control and memory operations are factored out	
		clustered function unit organization	◦ Vector fetch instruction & more fetch bandwidth	
			◦ Uses control thread & collection of micro-threads	
Illinois Rigel Accelerator	Programmable accelerator architecture	Collection of simple 8 cores	◦ data parallel and task-parallel computation	[178]
		Hierarchical caches (2 level)	◦ single-program multiple-data models of computation	[179]
		Explicit synchronization instructions	◦ distributed collection of global cache banks	
		Software memory coherence	◦ implicit shared memory interprocessor communication	
			◦ support implicit locality management	
			◦ software enforce cache coherence protocol	
			◦ coherence at a limited points reduce com. overhead	

330 [150, 136]). For the purpose of this thesis, we refer broadly to all of them as "embedded systems". A common characteristics across this diverse set of embedded systems is the need to operate correctly in the face of highly dynamic environmental conditions, changing application characteristics, as well as changes in the computing platforms itself (e.g., degradation or failures). Moreover, depending on the embedded system context, their architectures are highly customized to achieve the often conflicting constraints of performance, energy, and cost, reliability, etc. Furthermore, the complexity of the embedded software and hardware can vary over several orders of magnitude, depending upon the application domain, the usage context, and their physical deployment. Consequently, designers of embedded systems aim to increase the level of "smartness" in these systems, to adapt seamlessly to changes, increase the level of autonomous operation, and incorporate learning strategies. Depending on the type of embedded system, these needs typically translate into guarantees for functional and non-functional constraints; adaptation to dynamism, and the ability to tolerate failures.

With the increasing complexity of tasks faced by embedded systems, the designers of software and hardware systems have naturally borrowed concepts from biological systems in an attempt to mimic their ability to be self-aware, evolve, and achieve a high level of resilience in the face of highly dynamic and unpredictable environments. A large body of research in intelligent, autonomous systems, agent-based distributed systems, and advanced control theory have all used variants of the phrase "self-x", where "x" variously refers to capabilities such as awareness, healing, optimization, adaptation, etc. Unfortunately with this alphabet soup of terminology, there is little consensus of what these terms mean in the context of software and hardware systems, and for embedded systems in general. To disambiguate loosely-used terminology in the embedded systems literature, in Section 2.4 we begin by reviewing notions of self-awareness, self-adaptivity and autonomic systems in the large body of the literature in cognitive sciences (Section 2.4), on large software systems (Section 2.4.1), embedded systems (Section 2.4.2), and SoCs (Section 2.4.3). We then present a taxonomy in Section 2.5 to structure the terminology and related work based on the work presented in [104].

In spite of the bewildering diversity of embedded systems in general, at the core of all these embedded systems are integrated circuits made of silicon. As the number and variety of those devices grow exponentially, it becomes increasingly harder to guarantee perfect functionality and performance over the entire life time of these devices. This chapter will therefore focus on smart embedded systems primarily from the perspective of a System-on-Chip (SoC), and the associated challenges for developing software and hardware platforms upon which reliable, autonomous, and smart embedded systems can be built. From an SoC perspective, smart embedded systems (SES) are an emerging area of computing system with unique architectural attributes. SoCs as SES have many similarities with autonomic computing systems [180] but are severely resource and capability constrained. They can be analyzed through the computing-communication-control (C3) centric notions of cyber-physical systems (CPSs) [210] but are limited due to the lack of the explicit notions of the operating systems and compilation principles in C3.

2.4 Self-Aware, Self-Adaptive and Autonomic Computing Systems

Cognitive science has a long and rich history of theories about the mental faculties that link perception to action [362]. Two main categories can be distinguished: *cognitivist* and *emergent systems* approaches. The cognitive paradigm is based on the classic view that cognition is a “kind of computation” that uses symbolic and abstract representations of the real world and that algorithmically calculates actions [279]. In contrast, proponents of the emergent systems paradigm, which includes *connectionist*, *dynamical* and *enacting systems* approaches [362], argue that cognition is an emerging phenomenon in self-organizing, dynamic systems, that interactively identify and use regular patterns in the environment to continuously adapt, react and anticipate [354, 82]. Cognitivist approaches assume the existence of an objective reality which should be abstracted and symbolically represented, while the emergent systems community views the system and its environment as mutually dependent and continuously co-evolving.

One cognitive theory of consciousness, which falls into the emergent systems camp and which is relevant for our topic, is Baars' *Global Workspace Model* (GWM) [23]. Many parallel processes operate unconsciously and concurrently, but only one, or one coalition of processes, obtains access to the global workspace at any time, allowing it to broadcast its message globally and thus to marshal many global resources for its purpose. Hence, the global workspace serves to allocate and synchronize limited resources. Since its formulation, many phenomena predicted by the GWM have been confirmed in experiments [24, 22] making it today the top contender for explaining consciousness.

2.4.1 Awareness in Software Systems

The insight that a sense of awareness can facilitate robust and dependable behavior even under radical environmental changes and drastically diminished capabilities have inspired researchers to study the utility of cognitive features like adaptivity, awareness, or consciousness for robots, large scale software systems, and embedded systems. The benefits are more obvious for some features such as adaptivity but less for awareness or consciousness. Hence, adaptivity, and in particular self-adaptivity has been the focus of much research. In complex software systems, self-adaptivity is expected to help in managing the complexity [298]. Manual troubleshooting, reconfiguration, and maintenance are demanding and error prone. Above a certain complexity of the system, it becomes infeasible. Self-adaptive behavior is triggered either by changes of the system's *self* (internal causes like faults or mode transitions), or by changes of the system's *context* (external events like changes in user request rates or user objectives) [298].

In 1997 a DARPA Broad Agency Announcement offered a definition of self-adaptive software: "Self-adaptive software evaluates its own behavior and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible" [205], which points to several important aspects: (1) The system monitors its own behavior; (2) it knows what behavior is expected; (3) it compares its observed behavior to the expected behavior; (4) performance matters in addition to functionality. About the same time, Oreizy et al.[264] highlighted the importance of the environment: "Self-adaptive software modifies its own behavior in response to changes in its operating environment" [264], which requires that (5) the system monitors its environment, (6) knows what behavior of the environment is expected, and (7) knows its own appropriate behavior for a given environment.

Features (1)-(4) are related to *self-awareness* and (5)-(7) to *context-awareness* which form the bases of all self-modifying capabilities such as self-configuration, self-optimization, or self-adaptivity. In the often used hierarchy of self-* properties they are located in the *primitive level* below the *major level* which is populated by specific self-changing capabilities [298], as illustrated in Fig. 2.4.

IBM's original vision of autonomic computing [180], formulated in the early 2000s, puts its emphasis on the upper levels, implicitly assuming that awareness is a simple capability. In contrast, we argue that achieving awareness is hard, but once achieved,

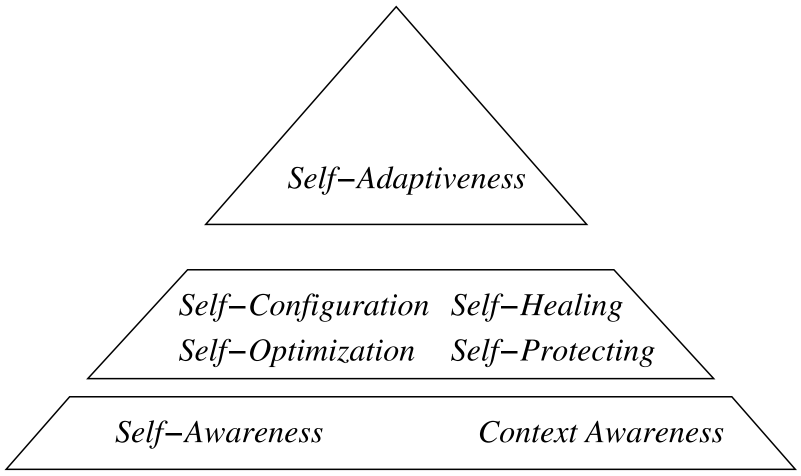


Figure 2.4: Hierarchy of self-* properties [298].

realizing the higher level properties at the major and general levels are difficult but tractable engineering tasks.

In reaction to the DARPA initiative and IBM's vision, research on self-* properties has flourished. Two recent surveys on self-healing by Ghosh et al. [117] and Psaier and Dustdar [278] discuss the approaches taken for detecting and reacting to faulty states of a system. Self-healing is rooted in work on fault tolerant and self-stabilizing [97] systems but emphasizes continuous availability and focuses on the recovery process. Both surveys agree that a kind of self-awareness is critical, but often view it narrowly as a mechanism to detect faults, which then triggers recovery procedures. Hence, the system perceives itself to be in one of two states: healthy or not healthy. Our understanding is broader and implies a richer perception of a system's own well being and performance allowing for a nuanced assessment as to which degree expectations and goals are met including a track record and a sense of historical performance.

Partially overlapping are efforts to design self-adaptive systems as elaborated from a variety of aspects in a book edited by Cheng et al. [77]. A self-adaptive system is more general than a self-healing system as it also adapts gracefully to changing environmental conditions. Again, publications on self-adaptivity view self-awareness rather narrowly as a means to detect unusual states and focus mainly on the reaction to such observations. However, it has been noticed that a more comprehensive approach to self-awareness aspects would be both desirable and challenging. For instance, Cheng et al. [77] note that knowledge of expectations by the environment, for which Finkelstein has coined the term *requirements reflection*, would be useful and conclude that "Future work is needed to develop technologies to provide such infrastructure support" [77].

In control applications models of the self have reached significant sophistication. Kaindl et al. proposed an explicit, symbolic representation of self for the purpose to monitor and self-configure the system based on changing needs and requirements [170].

An emotion based approach to assess the inner state and the wellness of a system is described by Sánchez-Escribadno and Sanz [300]. They use a prioritization mechanism to compare and relate the importance of otherwise independent states or events and call it "emotion". Sanz et al. have gone furthest by incorporating an explicit self-model in the control system, which elegantly is based on the model of the system used during the design process [303], resembling requirements reflection mentioned above. This establishes a secondary control loop in which the primary control algorithm can be adapted.

A. Morin [242] has formulated nine neurocognitive models of self-awareness distinguishing *unconsciousness*, *consciousness of external stimuli and events*, *self-awareness of public* and *private self-aspects*, and *meta self-awareness*. Based on Morin's classification Lewis et al. [217] consider categories of self-awareness with respect to their relevance to computing systems. They offer a working definition distinguishing between information that a system has about its own state (*private self-awareness*) and knowledge about how it is perceived by its environment (*public self-awareness*). Also, organization of self-aware systems in groups of peers leading to group-awareness is considered. The categorization outlined in Section 2.5 [161], is consistent and to a large degree aligned with Lewis et al.'s [217] definition, but our concepts are more detailed and formulated with the objective to engineer self-aware systems under tight resource constraints.

Chen et al. have proposed a pattern based approach to the design of self-aware systems [73]. Based on Lewis' classification they formulate seven patterns for specific functions relevant to self-awareness: *Basic information sharing, coordinated information sharing, temporal knowledge sharing, temporal knowledge awareness, goal sharing, temporal goal awareness*, and *meta-self-awareness*. Architectural patterns and a methodology for designing self-aware and self-expressive systems are formulated and applied to case studies with cloud computing and a smart camera networks applications.

2.4.2 Awareness in Embedded and Cyber-Physical Systems

In *embedded* and *cyber-physical systems* (CPS) the main problem is not so much the size induced complexity of an individual system but rather the tight resource constraints, the large number of those systems and their interaction, and the unpredictable environmental conditions of their deployment. Analysts expect 26 billion devices connected to the Internet of Things by 2020 (www.gartner.com/newsroom/id/2636073). Manual maintenance, diagnostics, and repair of most of these devices is soon impossible. Thus, there is a growing need that CPSs have a better understanding of their own state, their behavior, their performance, and the surrounding conditions. We call this "better understanding" *awareness*, which improves the behavior of systems, making them more robust while reducing processing, communication and energy requirements. A variety of bio-inspired approaches have been proposed for the operation, modeling, design, optimization, and verification of embedded systems and SoCs, as a recent collection illustrates [85]. For instance, Zakaria et al. [385] describe techniques to handle uncertainties because of faults due to process variation and limited yield, to manage power consumption and synchronization between different clock domains in SoCs. Evolvable hardware is a hardware that can change its architecture and behavior dynamically and autonomously [383, 134]. The hardware design is encoded in some kind of “chromosome” and evolutionary techniques such as genetic algorithms are deployed to modify this chromosome and thus the hardware as a reaction to a changing environment or faulty components. Since FPGA is a perfect medium for the implementation of evolvable hardware, the research field has flourished since the advent of FPGAs in the 1990s and is continuously exploiting FPGA features as they emerge [64].

However, designing and implementing self-awareness in an ad-hoc manner for every new system is not feasible. Introducing awareness as a separate concept in the CPS infrastructure promises to simplify development and operation of such systems. As CPS are typically Systems of Systems (SoS), the awareness must be solved comprehensively, ensuring that the understanding of the situation is coherent and consistent across the systems of systems (SoS).

Bakhouya and coworkers draw more explicit parallels to natural phenomena such as the immune system, cell organization and ant colonies [25, 26]. They correctly put emphasis on positive and negative feedback loops that are pervasive in natural systems and a key in the design of adaptive behavior in smart embedded systems.

Awareness is not necessarily confined to individual components; it may as well as emerge in cooperating systems of systems (SoS). Preden and coworkers have studied distributed surveillance systems and assign particular importance to the role of attention and context aware processing and sensing [243, 276, 274, 275]. They argue that these properties facilitate efficient operation of distributed sensing systems. Based on Endley’s *Situation awareness* [106], Preden et al. have developed the concept of *situation parameters* [274]. A situation is defined by the values and interpretation of a set of situation parameters, which are monitored or computed independently and represents a property of the situation of interest. The information for generating situation awareness is collected and processed independently of the application functionality and can be considered as part of the CPS platform.

Witnessing the high interest in this topic are surveys on related and relevant topics such as on-chip self-monitoring [194], bio-inspired hardware design [85], and situation identification techniques [384].

2.4.3 Awareness in Systems-on-Chip

There is a large body of literature on SoCs developed for embedded and cyber-physical systems that exhibit self-awareness characteristics at various levels. We have listed an incomplete set of examples focusing on reliability and power management in Tables 2.2 and 2.3, respectively. A number of German national projects have focused on

Table 2.2: Smart Dynamic Reliability/Resilience Management.

References	Adaptation Type		HMP	Sensing and Monitoring					Decision Making Layer					Cross-Layer Actuation Level				
	Simple*	**Self-Aware†**	**Support**	**Ckt**	**HW**	**NW**	**OS**	**App**	**Ckt**	**HW**	**NW**	**OS**	**App**	**Ckt**	**HW**	**NW**	**OS**	**App**
Shapiro et al. 2004 [321]	✓	Self-heal	No	✓	✓	–	✓	–	–	–	–	✓	?	–	–	–	✓	?
Sylvester et al. 2006[347]	✓	Self-heal	No	✓	✓	–	–	–	✓	✓	–	?	–	✓	✓	–	✓	–
Karl et al. 2006[172]	–	–	No	✓	✓	–	–	–	✓	✓	–	–	–	✓	✓	–	–	–
Austin et al. 2086[21]	✓	–	No	✓	✓	–	–	–	✓	✓	–	–	–	✓	✓	–	–	–
Sun et al. 2009[345]	✓	–	No	✓	?	–	✓	–	✓	?	–	✓	–	✓	?	–	✓	–
Das et al. 2009[91]	✓	–	No	✓	✓	–	–	–	✓	✓	–	–	–	✓	✓	–	–	–
Reddi et al. 2009 [287]	✓	–	No	✓	✓	–	–	–	✓	✓	–	?	–	✓	✓	–	?	–
Reddi et al. 2010[289]	✓	–	No	✓	✓	–	–	–	✓	✓	–	–	–	✓	✓	–	–	–
Leem et al. 2010[214]	✓	–	No	✓	✓	–	?	✓	✓	✓	–	?	✓	✓	✓	–	?	✓
Reddi et al. 2012[290]	✓	–	No	✓	✓	–	–	–	✓	✓	–	–	–	✓	✓	–	–	–
Kleeberger et al. 2013[192]	✓	–	No	✓	✓	–	–	–	✓	✓	–	–	–	✓	✓	–	–	–
Mercati et al. 2013[234]	✓	–	No	✓	✓	–	✓	–	✓	✓	–	✓	–	✓	✓	–	✓	–
Li et al. 2013[221]	✓	–	No	✓	✓	–	–	✓	✓	✓	–	–	✓	✓	✓	–	–	✓
Rehman et al. 2014[291]	✓	–	No	✓	✓	–	?	✓	✓	✓	–	?	✓	✓	✓	–	?	✓
Mercati et al. 2014[235]	✓	–	No	✓	✓	–	✓	–	✓	✓	–	✓	–	✓	✓	–	✓	–
Mercati et al. 2014[236]	✓	–	No	✓	✓	–	✓	–	✓	✓	–	✓	–	✓	✓	–	✓	–

* **Implicit Model** † **Explicit Model,** ? **Not explicitly discussed; Ckt = Circuit, HW = Hardware, NW = Network, OS = Operating System, App = Application**

computing systems that incorporate self-X properties, including the Organic Computing project [265], the InvasIC project [132], and the SPP1500 project on dependable embedded systems [133]. There is also a wealth of research on power management, thermal management [57, 87, 105, 306] and more recently, on an integration of both objectives [39, 238, 193]. The trend towards more elaborate management of aspects that are considered critical is apparent in research but also in industry, and we expect growing sophistication in the handling of individual concerns such as power consumption, over-heating, reliability, performance, etc., and a widening of scope to the simultaneous management of multiple, critical issues.

Most interesting are those that maintain a more sophisticated, internal model about the system's state, work that often draws on control theory. For instance, Wang et al. [369] propose a control algorithm based on an online model estimator to control accuracy and system stability. In a similar vein, Shafique et al. [320] use *implicit models* to predict key features such as required resources for an approaching time interval. The models in these, and many other, examples are implicit and serve a narrow purpose. History based prediction is a good example and commonly used. Based on a record on past resource usage of an application, the resource requirements for a future time interval are estimated. The past resource usage, perhaps only a single number, is considered a narrow model that represents a property of interest. Since almost all the approaches in Tables 2.2 and 2.3 focus on single issues with relatively simple objectives, they maintain narrow, implicit models of the systems themselves. The broader the objectives become and the more aspects that are integrated in the decision process, the richer the internal models grow. The power, thermal and reliability models used in

Table 2.3: Smart Dynamic Power Management.

References	Adaptation Type		HMP	Cross-Layer Sensing and Monitoring					Decision Making Layer					Cross-Layer Actuation Level				
	Simple*	**Self-Aware**†	Support	**Ckt**	**HW**	**NW**	**OS**	**App**	**Ckt**	**HW**	**NW**	**OS**	**App**	**Ckt**	**HW**	**NW**	**OS**	**App**
Kumar et al. 2003[202]	✓	–	No	–	✓	–	✓	–	–	–	–	✓	–	–	–	–	✓	–
Wu et al. 2004[376]	✓	–	No	–	✓	–	–	–	–	–	–	✓	–	–	–	–	✓	–
Wu et al. 2004[377]	✓	–	No	–	✓	–	–	✓	–	–	–	✓	✓	–	–	–	✓	✓
Isci et al. 2006[156]	✓	–	No	✓	✓	–	–	–	–	✓	–	✓	–	–	✓	–	✓	–
Nathuji et al. 2007[247]	✓	–	No	–	✓	–	✓	✓	–	✓	✓	✓	–	–	✓	✓	✓	–
Curtis et al. 2007[88]	✓	–	No	✓	✓	–	✓	–	–	✓	–	✓	–	–	✓	–	✓	–
Verma et al. 2008[361]	✓	–	No	–	✓	–	✓	–	–	✓	–	✓	–	–	✓	–	✓	–
Sridharan et al. 2008[338]	✓	–	No	✓	✓	–	–	–	✓	✓	–	✓	–	✓	✓	–	✓	–
Rangan et al. 2009[281]	✓	–	No	–	✓	–	✓	–	–	✓	–	✓	–	–	✓	–	✓	–
Wang et al. 2009[369]	✓	✓	No	–	✓	–	✓	–	–	✓	–	✓	–	–	✓	–	✓	–
Bartolini et al. 2010[33]	✓	–	No	✓	✓	–	?	–	✓	✓	–	–	–	✓	✓	–	–	–
Hoffmann et al 2011[142]	✓	✓	No	–	✓	–	✓	✓	–	–	–	✓	✓	–	–	–	✓	✓
Bartolini et al. 2011[32]	✓	✓	No	–	✓	✓	–	–	–	✓	✓	–	–	–	✓	✓	–	–
Rotem et al. 2011[294]	✓	–	No	✓	✓	–	–	–	✓	✓	–	✓	–	✓	✓	–	✓	–
Sun et al. 2013[346]	✓	–	No	✓	✓	–	–	–	✓	✓	–	✓	–	✓	✓	–	✓	–
Shafique et al. 2013[320]	✓	–	No	✓	✓	–	–	–	–	✓	–	✓	–	–	✓	–	✓	–
Shafique et al. 2013[319]	✓	–	No	✓	✓	✓	–	–	–	✓	✓	–	–	–	✓	✓	–	–

* **Implicit Model** †**Explicit Model,** ? **Not explicitly discussed; Ckt = Circuit, HW = Hardware, NW = Network, OS = Operating System, App = Application**

the virtual platform described by Bartolini et al. [33] are more detailed and elaborate, even though the models integrated in the final device, as part of the on-line feedback based control algorithm, are simplified and optimized. Often design time information is not sufficient or accurate enough due to unforeseen influences or aging effects. To counter such limitations on-line self-calibration and learning techniques are employed to improve the models used in the control algorithms [32]. Such needs require more detailed and *explicit models* to represent more of the system's features, thus gradually increasing their sophistication.

Starting from the other end, more systematic approaches towards self-awareness have been taken by the HAMSoC and SEEC projects.

HamSoC [123, 124, 122] is an SoC platform with a hierarchical agent structure as illustrated in Fig. 2.5. The cell agents are hosted by individual processing cores in a multi-core SoC. Clusters are formed along sub-system boundaries and the platform agent is responsible for the entire chip. While the agents at the platform layer and below are application independent, the application agent is customized towards the application needs. The agents perform a set of activities including *Communicate*, *Configure*, *Inquire*, *Order*, *Report*, and *Inform* [124] with the objective to monitor the system's state and performance, communicate with other agents across the hierarchy, and reconfigure the system to adapt it to a changed situation. The agents and their actions are defined in a generic and abstract way to form a framework suitable for a variety of applications and implementations. As an application case a power management system for an NoC based multicore SoC has been implemented and evaluated[160, 122]. The cell and

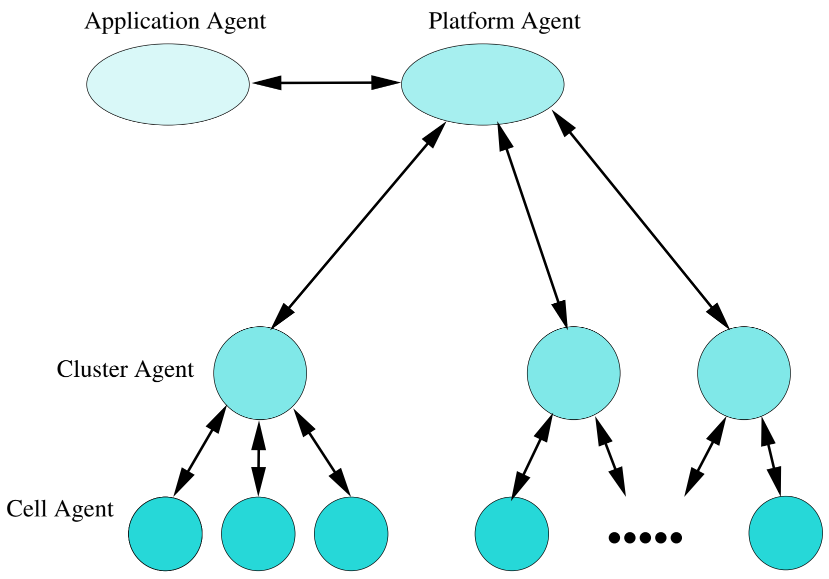


Figure 2.5: Agent hierarchy in the HAMSoC system [124].

cluster agents are realized in hardware while the agents at the platform and application layers are software programs. In the case study performance and power attributes are monitored and controlled but the framework is fairly general and would allow for monitoring any interesting property while the decision process could be assigned to the appropriate agent at the cluster, platform or application layer.

SEEC [140] is a general framework for self-aware computing using an observe-decide-act paradigm. As illustrated in Fig. 2.6, the system cyclically monitors key features, applies a control and decision algorithm, and deploys appropriate actions to adapt to changes in the environment and its own state. It is based on the heartbeats API library [139], which defines a cyclic event called a heartbeat. Through API functions the application can register rate and latency performance goals in terms of the heartbeat period. Hence, the heartbeats API is a standardized mean to monitor the performance of an application. The application itself, or a separate agent, can then adapt and optimize the system's behavior, for instance, by allocating and scheduling resources appropriately. The approach has been further developed and evaluated in several applications for performance optimization [140], power management [142, 138], and managing of multiple objectives [137]. Also, the concept of *knobs* have been introduced [142] to expose steering facilities such as processor speed or power modes. As a conceptual framework it allows to adopt different decision making strategies and algorithms, which has been explored and studied extensively [301, 230].

As we observe the extensive work published in this area, we note that all approach the domain of self-awareness from different directions and angles. The "single-issue" approaches, as listed only incompletely in Tables 2.2 and 2.3, introduce specific and

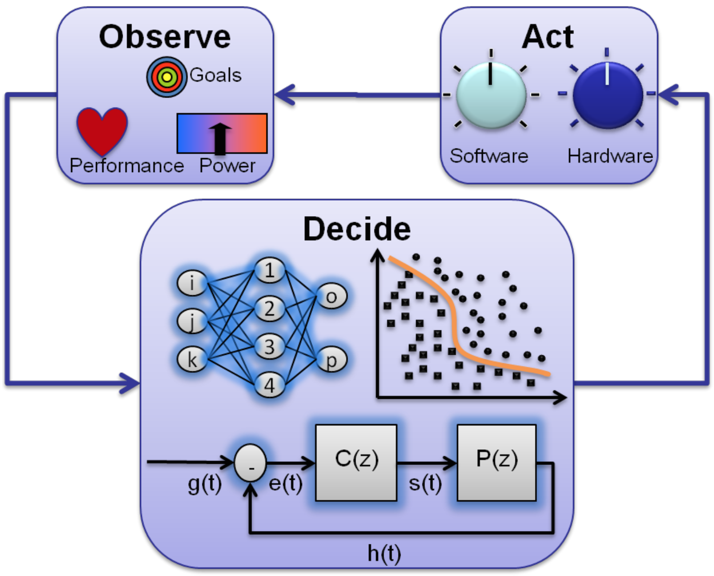


Figure 2.6: The SEEC activity cycle [140].

concrete aspects of self-monitoring and self-adaptation to solve a particular, well but narrowly defined problem. While the proposed techniques lead to effective solutions in the given scope, they do not easily generalize to a situation where a range of objectives have to be met simultaneously under a set of constraining conditions. In particular, various kinds of uncertainties and incomplete information constitute additional complications.

To overcome these limitations, a few general frameworks have been developed such as HAMSoC and SEEC. Both propose a basic concept (hierarchical agent network in the case of HAMSoC and a observe-decide-act operation cycle with the heartbeat paradigm in the case of SEEC) and apply it to ever broader application scenarios while further developing and refining the frameworks. Note that these endeavors are complimentary in terms of the insight they generate and the techniques they describe, as well as study.

2.5 Properties and Levels of Awareness

In contrast, the classification we describe in the following starts from the other end and lists the attributes that we expect to see in a self-aware system.

The various concepts reviewed are certainly multi-dimensional and have too many facets and aspects to easily press them into a simple scheme of classification. But if we concentrate for a while on awareness and self-awareness, which are the basis for many higher level cognitive abilities, we can identify different features, which, although not arranged linearly, constitute a well structured space that will allow us to better assess specific realizations in smart embedded systems. The framework of awareness that we use [161] requires several properties before we can call the system aware of something or self-aware [161]:

- **Abstraction** of the primary input data into a semantic domain which is meaningful for the system at hand.
- **Disambiguation** of the possible interpretations to always settle on exactly one interpretation of the reality at any given time. When new data becomes available, the interpretation may change, but at any given time there is only one interpretation used by the system.
- **Semantic Interpretation** is the result of abstraction and disambiguation and it represents a relevant property of the system or its environment.
- **Desirability Scale** provides a uniform goodness-scale for the assessment of all observed properties.
- **Semantic Attribution** maps properties into the desirability scale suggesting how good or bad an observation is for the system.
- **History of a Property:** Awareness of a property implies awareness of its change over time. This history may be more or less detailed and may slowly fade as time passes, but it certainly is required to allow for assessment of properties, the environment, and the system itself in a historical context.
- **Goals** provide the context in which interpretation and semantic attribution is meaningful.
- **The Purpose** of a smart embedded systems is to achieve all its goals.
- **Expectation on Environment:** The system expects a specific environment, which is a precondition to realize if the environment is profoundly changing. The system's goals are often dependent on the environment.
- **Expectation on Subject:** Similarly, the system's own state and condition are continuously assessed to detect deviations, degradation, excellent performance and malfunctions.
- **Inspection Engine:** Continuously monitoring and assessing the situation requires a specific machinery that integrates all observations into a single, consistent world.

To realize all these properties in a smart embedded system is rather ambitious and not always necessary. Depending on which of these properties are present and to what degree we can group systems in five levels [161]:

- **Level 1, Adaptive**: A classic PID controller adapts to changes in the environment by following reference values. Such a system does some abstraction and has some expectations but in rather limited ways.
- **Level 2, Property Aware**: The system derives a *semantic interpretation* and *attribution* of monitored data. The system has *expectations* regarding the monitored property. It also has *goals* and the attribution is done with respect to these goals. The more properties it follows in this way, the larger the share of the environment becomes that it is aware of. If the system monitors its own properties, we call it *self-aware*.
- **Level 3, History Aware**: If in addition to properly interpreting and classifying properties, the system maintains a *history* of observations, the environmental changes over time are monitored and assessed. Moreover, a *history self-aware* system can monitor and assess its own performance and relate it to expectations and goals.
- **Level 4, Predictive**: The inspection mechanism, that allows the system to observe and assess the environment and itself, can be used to study future scenarios as support for decision making. A system with the capability to simulate if-then-else scenarios is called predictive.
- **Level 5, Group Aware**: In addition to the self and the environment, the system recognizes a *peer group* with shared goals and/or similarity in behavior.

Like many other classifications, the details and boundaries can be debated, but the above classification provides a simple framework to categorize the work done in this field. A classification similar in ambition and scope has been proposed by Lewis and coworkers [216, 109]. Inspired by Neisser's work in psychology [248] they distinguish between the five awareness levels *stimulus-aware*, *interaction-aware*, *time-aware*, *goal-aware* and *meta-self-aware*. There is no simple mapping to the categorization above, but all concepts found in one can also be identified in the other, although with different emphasis. The property awareness in [161] is similar to Lewis et al.'s [216] stimulus-awareness but making the abstraction mechanism explicit. Similarly, level 1 of an adaptive system in [161] is related to interaction-awareness in [216] but with the focus on the adaptivity while keeping the concept of interaction rather implicit. The history awareness in [161] and Lewis et al.'s time-awareness resemble each other quite well. Lewis et al.'s [216] goal-awareness and meta-self-awareness separate aspects of inspection and reasoning about goals and the self, which are combined in the predictive level. The group awareness (i.e., level 5 in the above classification) is not covered in Lewis et al.'s [216] set of levels but they consider it as a distinct aspect under the label of *collectives and emergent self-aware systems* [216].

This direct comparison of these two schemes in [161] and [216] of categorization highlights that there is no single scheme yet that structures the relevant concepts in an obviously more natural way than others. Depending on preference, emphasis and objectives one may choose and adapt a proper categorization. However, it is also reassuring since different schemes tend to cover the same ground and thus, important aspects have most likely not been overlooked.

Reviewing the state of the art with the above five level scheme in mind, we observe that most embedded and cyber-physical systems proposed do some abstraction and interpretation of individual properties such as power consumption, performance, and occurrence of specific faults. Regarding these properties there are also, mostly implicitly, defined goals and expectations. With the exception of the work done by Preden and Helander [277] and the heartbeat framework [139], history records of properties are not kept or used in any systematic way. The systems described by Sanz et al.[303] and, to a more limited extent, by Kaindl et al. [170] use fairly sophisticated inspection engines and modeling capabilities. The work by Sánchez-Escribado and Sanz [300] is an interesting attempt towards, a unified desirability scale and a semantic attribution. A kind of desirability scale is found whenever several objectives are targeted simultaneously [137, 368, 347]. However, it appears implicitly as part of an objective function, which itself is often not maintained explicitly, meaning it cannot be generalized. We expect from a self-aware system that perhaps tens of partially independent observed properties are easily related to each other with respect to an equally large set of desired goals. Similarly, purpose and goals are either implicitly hidden in some decision algorithm or hard-coded at design time, or both. In order to meet a larger set of more or less independent goals, which are also likely to change over time, a more systematic and explicit representation of objectives has to be developed. An interesting step is found in the heartbeats framework [139] which allows applications to register performance goals, that are then monitored by the framework. It would be interesting to explore if this approach can be generalized to functional objectives and made sufficiently flexible to allow dynamic formulation of new goals. Along the same lines we note, that expectations on the environment are handled similarly ad-hoc and implicit, if at all.

The work by Preden et al. [275, 276, 243] can be considered the most advanced attempt towards *group awareness,* although it is still limited in this respect. The recent strong attention on fog computing [164, 314, 143, 50] may speedily advance this field and contribute to an understanding of group awareness of cooperating systems, how it emerges and what it is good for.

In summary, apart from classic control systems which populate Level 1, the overwhelming majority of the work discussed here appear in level 2, though they realize property awareness to grossly different degrees. There are some isolated attempts to address core features of levels 3, 4 and 5. None of the levels, 2 and higher, are covered satisfactorily, hence, significant research challenges remain.

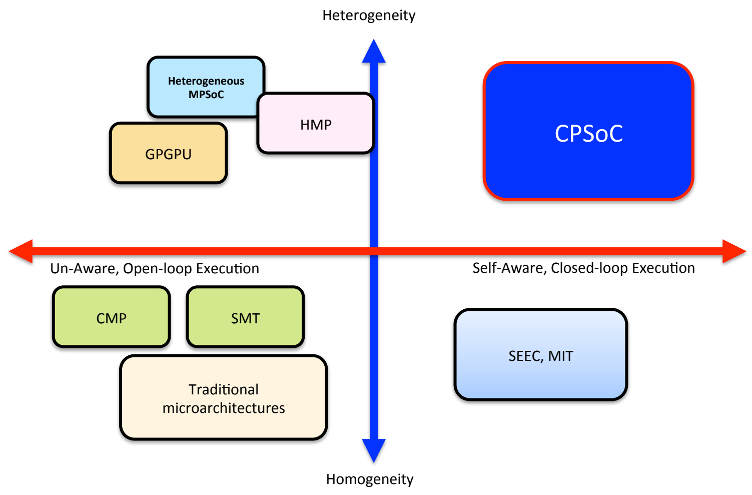

Heterogeneity
Heterogeneous MPSoC
HMP
GPGPU
CPSoC
Un-Aware, Open-loop Execution
Self-Aware, Closed-loop Execution
CMP
SMT
SEEC, MIT
Traditional microarchitectures
Homogeneity

Figure 2.7: Classification of architectures.

2.6 Self-Aware CPSoC

MPSoCs are becoming large and complex [51] following Moore's Law and are adopting diverse types of cores and on-chip networks for scalability. Traditionally, multicore processors were designed for single metric of design merit such as performance or energy; today we need to optimize both energy and performance, manage locality, parallelism, as well as resiliency. In addition to multiple goals, these MPSoCs, which are networked distributed computing systems, not only required to perform computation efficiently, but also have to provide some form of guarantees and assurances to achieve demanding non-functional requirements such as power efficiency, thermal stability, and resiliency which are physical in nature. Programming these systems with multiple conflicting goals has become hard because we are faced with a multidimensional optimization problem. As a result, our fundamental computing model must change to address this multidimensional optimization. Providing formal guarantees with efficient use of the on-chip resources motivated use of control theory [140], making these MPSoCs a cyber-physical system (CPS) by definition. Use of control theory in these cyber-physical systems-on-chip (CPSoCs), fundamentally changes the execution model of these systems. Unlike traditional MPSoCs and other microarchitecture discussed in Section 2.1, CPSoC exhibits heterogenous multi/manycore architecture with a closed-loop self-aware execution model as depicted in Fig. 2.7. CPSoCs are uniquely distinguished by several architectural characteristics as illustrated in Table 2.4 such as self-aware execution model, heterogenous manycore processor (HMP) architecture, distributed on-chip sensor-networks (sNoC), distributed actuation network-on-chip (xNoC) in addition to communication and coherency NoCs (cNoCs). We present the details of these architectural features that supports self-awareness in the subsequent subsections.

Table 2.4: Architectural Features of Open Source Processors.

Processor	ISA	Multicore/ Manycore	HMP	Self-Aware†	sNoC, xNoC	cNoC	OS	FPU	MMU	HW Multithreaded	Performance
pAVR [260]	8b AVR	No	×	×	×	×	×	×	×	×	Low
openMSP430 [154, 258]	16bMSP430	No	×	×	×	×	×	×	×	×	Low
CPU86 [146]	16b x86	No	×	×	×	×	✓	×	×	×	Low
Zet[7]	16b x86	No	×	×	×	×	✓	×	×	×	Low
LatticeMico32 [317]	32b LatticeMico32	No	×	×	×	×	✓	×	×	×	Low
ZPU [8]	32b MIPS	No	×	×	×	×	✓	×	×	×	Low
SecretBlaze [31]	32b MicroBlaze	No	×	×	×	×	×	×	×	×	Low
AltOr32 [258]	32b ORBIS	No	×	×	×	×	✓	×	×	×	Low
aeMB [375, 250, 323]	32b MicroBlaze	No	×	×	×	×	✓	×	×	✓	Medium
Amber [259, 302]	32b ARM v2a	No	×	×	×	×	✓	×	×	×	Medium
OpenRISC [349, 261]	32b/64b ORBIS	No	×	×	×	×	✓	✓	✓	×	Medium
MIPS32 r1 [6]	32b MIPS32 r1	No	×	×	×	×	✓	×	×	✓	Medium
Simply RISC S1 [292]	64b SPARC V9	No	×	×	×	×	✓	✓	✓	×	High
LEON 3 [10, 9]	32b SPARC V8	SMP/AMP	×	×	×	×	✓	✓,($)	✓	×	Medium
OpenScale [61]	32b MicroBlaze	Manycore	×	×	×	✓	✓	×	×	×	Medium
XUM [5, 232]	32b MIPS32 r2	Manycore	×	×	×	✓	✓	×	×	✓	High
BERI [4, 372]	64b MIPS/CHERI	Multicore	×	×	×	×	✓	✓	✓	✓,(BERI2)	High
OpenSPARC T1/T2 [262, 263]	64b SPARC V9	Multicore	×	×	×	×	✓	✓	✓	✓	High
RISC-V Rocket [213, 358]	64b scalar RISC-V	Manycore	×	×	×	✓	✓	✓	✓	×	High
RISC-V Boom [66, 357]	64b scalar RISC-V	Manycore	×	×	×	✓	✓	✓	✓	×	High
OpenPiton [28]	64b SPARC V9	Manycore	×	×	×	✓	✓	✓	✓	✓	High
CPSoC	32b/64b SPARC	Manycore	✓	✓	✓	✓	✓	✓	✓	✓	High

† **Self-Aware Execution Model, sNoC = sensor Network-On-Chip; xNoC = Actuator Network-On-Chip,**

2.6.1 On-chip Sensing and Monitoring for Awareness

Diverse real-time monitoring functions assist towards self-awareness objective through the collection of important system metrics, such as throughput of processing elements, communication latency, or resource utilization for each application. The online evaluation of these metrics can result in localized or global decisions that attempt to improve aspects of system behavior, system performance, quality-of-service, power and thermal effects under nominal conditions. Observation and collection of the essential pieces of information plays a central role in creating and using several on-chip mechanisms. Designers commonly place custom, application specific observation probes at various locations in a platform, to dynamically capture critical events, to calculate statistics, or to achieve error-resilient processing and communication. This results in significant number of sensors and monitors. For example, IBM Power 7 [215] processor has over 40 distributed thermal sensors, processor core and memory activity counters, off-chip current and voltage sensors. These sensors include physical sensors such as delay sensors [67, 100, 206], voltage and power sensors [168, 289, 55], temperature sensors [69, 79, 251, 270], and reliability sensors [12, 76, 222, 215]. Sensors can also be implemented at higher level of system stack such as architecture performance counters [337] and NoC traffic/congestion monitors [80]. Most existing sensors are designed solely to monitor specific phenomena. On the other hand, CPSoC expands the role of native sensors by multi-purposing sensors and reducing sensing overheads through extensive use of sensor fusion rather than the development of new sensors. Various types of on-chip sensors used in CPSoC are used to develop predictive behavior models that are essential for awareness of the system architecture and execution model.

2.6.2 Cross-Layer Architecture Models for Self-Aware Adaptation

Emerging MPSoC faces formidable challenges, especially with the slow-down of traditional integrated circuit scaling has prompted comprehensive research on faster, more reliable and energy-efficient switches. However, better switches alone will not deliver the necessary leaps in performance. Instead of focusing solely on improving transistors or memory cells, an integrated approach that shift from transistor-scaling-driven performance improvements to a new post-scaling whole-stack co-design would be the key to improved efficiency and promises of significant benefits. An end-to-end approach is essential, as substantial improvements are generally rare, and cannot be achieved with uncoordinated improvements in architectures, transistors, or memory cells alone but through symbiotic relations among components to enhance key performance metrics. Over-optimizing a single component without considering the full-system may not improve the system margin, as components from other layers may be the bottleneck. Improving each component/layer of the system stack where each component/layer shows comparable improvement can result in synergies; when exploited through careful co-design and co-optimization of software and hardware can enable higher efficiency and performance. For example, faster memory accesses cut core idle times, reducing energy consumption and overall execution time [89]. When combined with increased memory bandwidth and improve memory locality, enabling many concurrent memory accesses reducing memory access contention significantly, performance speedup and energy efficiency improves appreciably [37]. On the other hand, comprehensive improvement in the layers of the stack through (scale-out and scale-up) parallelism and near-threshold voltage scaling with tolerance of variability across the layer can provide necessary leaps in performance and energy efficiency [103]. There is need for integrated approach spanning emerging logic devices and memories, computer architecture, cooling solutions, and synergistic runtime software.

In order to advance whole-stack co-design and optimization at a reasonable development costs and time, cross-layer awareness both at design time as well as run-time is key to enabling this objective. Awareness is critical because an un-aware computing system can never respond to users and system goals by being oblivious to system state, behavior, and environment. For instance, because of the lack of accurate awareness of the design margins, pessimistic assumptions incurring significant overheads are made across the layer during hardware design time. With the availability of certain system/architecture-level awareness information, overhead and margin can be reduced by relaxing some of these assumptions. Additionally, system design margin may be dominated by the existence of certain pathological scenarios. These pathological scenarios may be caused by behaviors from different layers of system abstraction. Cross-layer awareness and approach is required as awareness of a single-layer may be inadequate to avoid these scenarios during design optimization approach. Furthermore, some hardware-related phenomena are difficult to monitor by hardware monitors alone. For example, timing and soft error rate are extremely difficult to monitor due to the rare occurrence nature and unique ambient dependence. Cross-layer awareness approaches can use system and cross-layer information for better monitoring and awareness to investigate and address several limitations of current SoC design.

A fundamental step in the awareness of the whole-stack of the system is cross-layer monitoring and cross-layer modeling in a principled way. Cross-layer analytical models predict chip key metrics by capturing key technology trends, design constraints, workload characteristics and microarchitectural variables into a mathematical framework. Tractable, accurate and insightful analytical models will become crucial in evaluating many-core chip designs of the future. Principled approaches to tractable, accurate and insightful analytical cross-layer modeling are crucial in order to understand, evaluate, and optimize the whole-stack key system design metrics of many-core chip designs of the future.

In order to build models online, continuous monitoring of the system to construct a training dataset of performance statistics is required. Correlation of the dataset by process, process type, and process group, collection of traces and logs of process activity, and deriving predictive model to capture the performance (and other design metrics such as power/ energy) for the observed patterns are commonly performed. In situ simulation of new algorithms using run-time measurements and prediction model are used in adapting the system according to system or user goals.

2.7 Summary

Self-awareness has a long history in biology, psychology, medicine, and more recently in engineering and computing, where self-aware features are used to enable adaptivity to improve a system's functional value, performance and robustness. With complex many-core Systems-on-Chip (SoCs) facing the conflicting requirements of performance, resiliency, energy, heat, cost, security, etc. – in the face of highly dynamic operational behaviors coupled with process, environment, and workload variabilities – there is an emerging need for self-awareness in these complex SoCs. These concerns highlight the need for smart SoC paradigm and architectures that dynamically balance multiple objectives across multiple levels of the design abstraction stack, manage their limited resources and are always keenly aware of their own accomplishments and shortcomings. These abilities and attributes distinguish them from traditional embedded MPSoC design and motivate the need for a new design paradigm. In this chapter, we presented the relevant background, prior work, and trends in emerging SoC that falls into several categories. We described the classification of contemporary microarchitecture and their differences with smart SoC architecture, self-aware, self-adaptive, autonomic systems. We then introduced the notion of self-aware execution model for heterogeneous manycore processors as new class of architecture and discuss the CPSoC paradigm as an example of such architecture with on-chip sensing for self-awareness, awareness modeling, and self-aware architectures. The CPSoC approach described resembles and is inspired by the emergent systems paradigm in that it provides for simple faculties of monitoring, actuating and control in a bottom-up manner to allow larger systems to evolve more potent capabilities as individual and as groups of collaborating systems. However, it hopes to short-cut long learning cycles by equipping systems with key capabilities without waiting for their emergence through evolution.

Chapter 3

CPSoC: Concept and Architecture

3.1 Introduction

MPSoCs are becoming large and complex [51] following Moore's Law and are adopting diverse types of cores and on-chip networks for scalability. Traditionally, multicore processor were designed for a single design metric (design figure of merit) such as performance or energy; today we need to optimize both energy and performance, manage locality, parallelism, as well as resiliency. In addition to these multiple goals, MPSoCs, which are networked distributed computing systems, not only are required to perform computation efficiently, but also have to provide some form of guarantees and assurances to achieve demanding non-functional requirements such as power/energy efficiency, thermal stability, reliability and resiliency. Programming and managing these systems with multiple conflicting goals has become hard as we are faced with a multidimensional optimization problem that requires improved awareness and runtime decision making capability. As a result, our fundamental computing model must change to address this multidimensional optimization and adaptation problem.

In this chapter, we present the rationale and concept of a self-aware adaptive SoC paradigm called the Cyber-Physical-System-on-Chip (CPSoC). We present the architectural features and attributes that distinguish CPSoCs from traditional MPSoCs. We describe the models of the CPSoC architecture that can be implemented using existing open framework, tools and methods. This chapter provides overview of the different components of the CPSoC paradigm and an outline connecting the constituting parts with details in subsequent chapters.

3.2 Rationale and Concept of CPSoC

Engineering variability challenges and the struggle to control variations in MPSoCs with exploding number of computing cores [51] calls for a rethinking of the computational devices and their architectural design paradigm. With an increasing tilt towards adaptive SoC design and influence of physical parameter dynamics, MPSoCs bear several similarities with cyber-physical systems (CPSs) that are real-time controlled and coordinated systems relying on computational infrastructure. For example, both these types of systems have complex functional specifications, demanding non-functional specifications, multi-modal behaviors, as well as use of networks in their design. Even though the operational life-time and timeline of designs are very different, design of both these systems involves a control/computing co-design of closed-loop plant with computer systems in the loop. The characteristics of the plant specially vary, the former being computation and the latter being a physical system, with disproportionate scale (e.g., one distributed in the nano-scale and the other geographically distributed).

In Fig. 3.1 the concept of a cyber-physical system and the models used in different components of the CPS representation are illustrated where the tight integration of computation and networking with physical processes are shown with an example of a spacecraft avionics. These system-of-systems are extensively monitored, coordinated, controlled in real-time. When the same parallel is drawn with that of emerging heterogeneous MPSoC, the plant represents the physical dynamics of the MPSoC (such as power, thermal, reliability etc.) controlled by computational process and on-chip networks (as illustrated in Fig. 3.2(a)) and are called Cyber-Physical-System-on-Chip (CPSoC) by analogy. The adaptation and control mechanisms adopted by these SoCs may either be incorporated in the same chip, or using a additional external service chip. However, as large computational resources are available in emerging SoCs, the adaptation mechanisms are expected to be present in the same chip. In order to capture the physical behavior of the whole chip, the MPSoC plant encompasses both the computational process (i.e., the cores running software) and network behavior as illustrated in Fig. 3.2(d)-(e). This integrated behavior model of MPSoC physical characteristics (such as thermal- power dynamic behaviors, the responsiveness, reliability etc.) greatly benefits whole-stack adaptation and awareness of the chip. The use of adaptive control theory helps to achieve self-aware adaptation in CPSoCs.

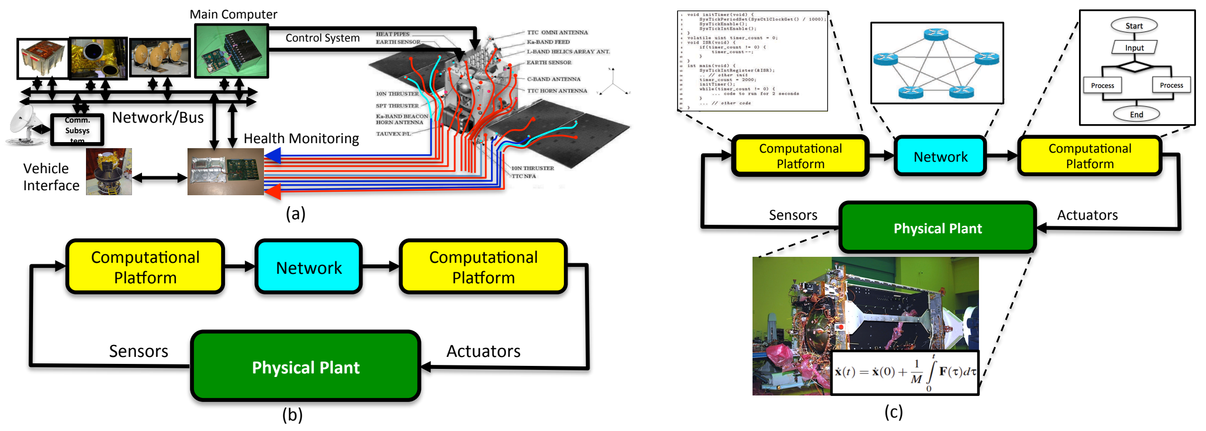


Figure 3.1: Concept and representation of Cyber-Physical Systems (CPSs) (a) a spacecraft example (b) block diagram representation of CPS (c) models used in CPS representation [211] .

Use of control theory in these CPSoCs, fundamentally changes the execution model of these systems. Unlike traditional MPSoCs and other microarchitectures discussed in Section 2.1, CPSoCs are uniquely distinguished by several architectural characteristics such as self-aware execution model, heterogeneous manycore processor (HMP) architecture, distributed on-chip sensor-networks (sNoC), distributed actuation network-on-chip (xNoC) in addition to communication and coherency NoCs (cNoCs). In this chapter, we present the concept of CPSoC and its architectural features that support self-awareness.

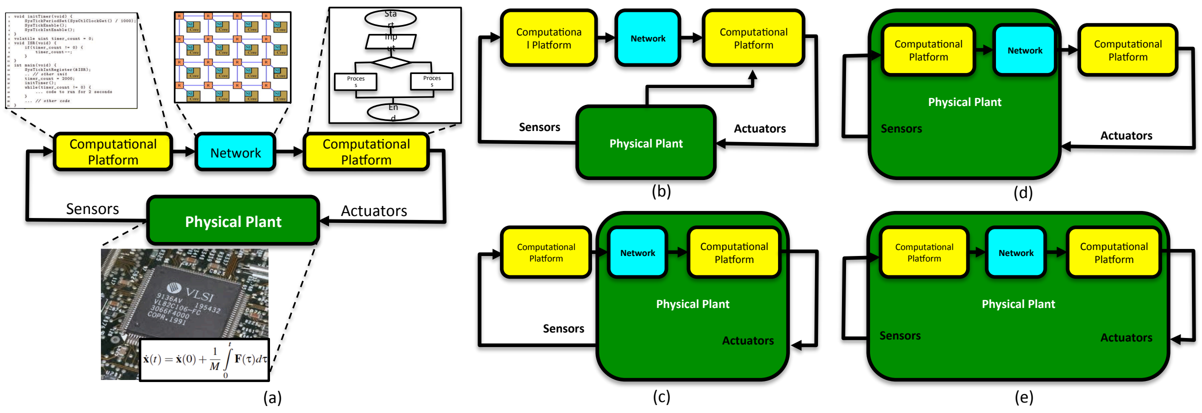


Figure 3.2: From CPS to CPSoC: concept and model representation.

3.3 CPSoC Paradigm

CPSoC [312, 313] is a sensor-actuator-rich self-aware computing-communication-control (C3) centric paradigm with an adaptive middleware (a flexible hardware-software stack and interface between the application and OS layer) to control the manifestations of computations (e.g., aging, overheating, parameter variability etc.) on the physical characteristics of the chip itself and the outside interacting environment. Inspired by the adaptive and learning abilities of autonomous computing [180] and C3 paradigm of CPSs [210], CPSoC provides a computing framework that assures the dependability of the cyber/information processing (i.e., the cyber aspects such as integrity, correctness, accuracy, timing, reliability and security) while simultaneously addressing the physical manifestations (in performance, power, thermal, aging, wear-out, material degradation, reliability, and dependability) of the information processing on the underlying computing platform. Note unlike the reference architecture proposed by Lewis et al. [216], CPSoC aims to coalesce these two traditionally disjoint aspects/abstractions of the cyber/information world and the underlying physical computing worlds into a unified abstraction of computing by using cross-layer virtual/physical sensing and actuation to enable a C3 centric self-aware computing platform.

The CPSoC architecture consists of a combination of sensor-actuator-rich computation platform supported by adaptive NoCs (cNoC – communication NoC; and sNoC – sensor NoC), introspective sentient units (ISU), and an adaptive middleware to manage and control both the cyber/information and physical environment and characteristics of the chip [312, 313]. The CPSoC architecture is broadly divided into several layers of abstraction, for example, application, operating system, network and bus communication, hardware, and the circuit / device layers. CPSoC inherits most features of MPSoC in addition to on-chip sensing and actuation to enable the Observe-Decide-Act (ODA)

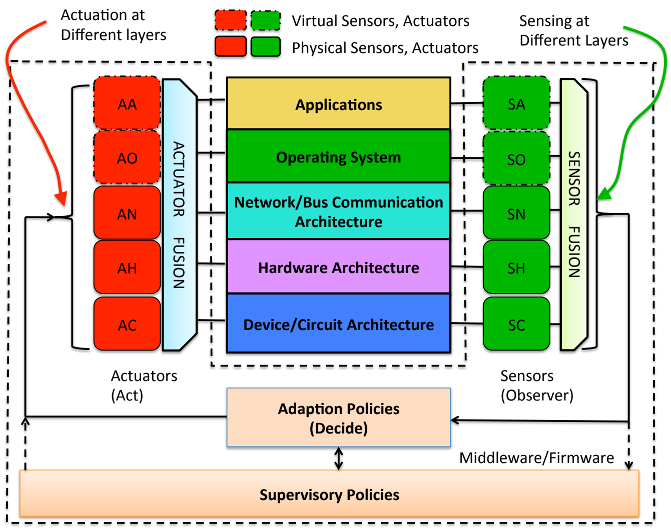

Actuation at Different layers
Virtual Sensors, Actuators
Physical Sensors, Actuators
Sensing at Different Layers
AA
AO
AN
AH
AC
ACTUATOR FUSION
Applications
Operating System
Network/Bus Communication Architecture
Hardware Architecture
Device/Circuit Architecture
SA
SO
SN
SH
SC
SENSOR FUSION
Actuators (Act)
Sensors (Observer)
Adaption Policies (Decide)
Middleware/Firmware
Supervisory Policies

Figure 3.3: Cross-layer virtual sensing and actuation at different layers of CPSoC [309, 313].

paradigm. Unlike traditional MPSoC, each layer of the CPSoC can be made self-aware and adaptive, by a combination of software and physical sensors and actuators as shown in Fig. 3.3. These layer specific feedback loops are integrated into a flexible stack which can be implemented either as firmware or middleware as shown by the dotted line in Fig. 3.3.

CPSoC distinctly differs from a traditional MPSoC in several ways. Traditional MPSoC paradigms lack the ability to sense the system states and behaviors across layers of the system stack due to lack of architectural support; they are incapable of exploiting and exposing process and workload variations due to lack of suitable abstractions at multiple layers. Furthermore, they sacrifice usable performance and energy opportunities by adopting worst case design (guard-bands), and lack support for multi-level actuation mechanisms and adaptations to aggressively meet competing and conflicting demands. Moreover, traditional MPSoCs lack self-learning mechanisms that can anticipate failures and predict vulnerabilities. CPSoC overcomes these limitations as detailed below.

3.4 Features and Attributes of CPSoC

The CPSoC framework supports four key ideas: 1) physical and virtual sensing and actuation 2) simple and self-aware adaptations 3) cross-layer interactions and interventions 4) predictive modeling and learning. We briefly describe these below.

3.4.1 Cross-Layer Virtual and Physical Sensing and Actuation

CPSoCs are sensor-actuator-rich MPSoCs that include several on-chip physical sensors (e.g., aging, oxide breakdown, leakage, reliability, temperature, performance counters, as well as voltage, current, and power sensors [312, 313]) on the lower three layers as shown by the on-chip-sensing-and-actuation block (OCSN) in Fig. 3.7. On the other hand, **virtual sensing** is a physical-sensor-less sensing of immeasurable parameters using computation [311]. It can be viewed as a software sensor that provides indirect measurement of abstract conditions, contexts, inferences or estimates by processing (e.g., combining, aggregating, or predicting) sensed data from either a set of homogeneous or heterogeneous sensors. It is also a computational technique that enhances and/or adds sensing capability, introduces sensing options, increases sensitivity, enables efficient sensor resource uses, and overcomes physical placement and cost restrictions. When combined with different kinds of sensors, virtual sensing enables consensus to resolve faults and errors while providing a testbed for on-chip sensor fusion [341]. Table 3.1 shows examples of physical and virtual sensors and actuators across the system stack that are key to achieving system goals and quality-of-service (QoS). QoS is used as a general quality measure in the sense of user perceived performance, or degree of satisfaction to the user. Application-level QoS can be ensured by either resource-reservations at all underlying levels or by adaptation of application to cope with changing resource availability.

Similarly, we define virtual actuations [312, 313] (e.g., application approximation, algorithmic choice, checkpointing, duty cycling) as software/hardware interventions that can predictively influence system design objectives such as performance, power, energy efficiency, and reliability. Virtual actuation can be combined with physical actuation mechanisms commonly adopted in modern chips (e.g., DVFS and adaptive body biasing (ABB)) to control the chip's performance, power, and parametric variations.

3.4.2 Simple and Self-Aware Adaptations

Self-awareness is used to describe the ability of the CPSoC to observe its own internal states and behaviors such that it is capable of making judicious decisions that optimize performance and other quality-of-service (QoS) metrics [180]. Self-aware systems are capable of adapting their behavior and resources to automatically find the best way to accomplish a given goal despite changing environmental conditions and demands. A self-aware system must be able to monitor its behavior to update one or more of its components (hardware architecture, operating system and running applications), to achieve its goals.

Two key attributes of the self-aware CPSoC are adaptation of each layer and multiple cooperative ODA (Observe-Decide-Act) loops. As an example, the unification of a

Table 3.1: Virtual/Physical Sensing and Actuations Across Layers.

Layers	Virtual/Physical Sensors	Virtual/Physical Actuators
Application	Workload, Power, Energy and Execution Time, Phases	Loop Perforation, Approximation, Algorithmic Choice, Transformations
Operating System	System Utilization and Peripheral States, TLP	Task Allocation, Partitioning, Scheduling Migration, Duty Cycling
Network/ Bus Communication	Bandwidth, Packet/Flit Status and Channel Status, Congestion	Adaptive Routing, Dynamic BW Allocation and Ch. no and Direction Control
Hardware Architecture	Cache Misses, Miss Rate, Access Rate, IPC, Throughput, ILP, MLP	Issue-Width Sizing, Reconfiguration Resource Provisioning, Static/Dynamic Redundancy
Circuit/Device	Circuit Delay, Aging, Leakage Temperature, Oxide Breakdown	DVFS, ABB, Voltage Frequency Island Clock Gating, Power Gating

heterogeneous computing platform (with combined scheduling and DVFS as actuation mechanism) along with application/algorithmic level approximations as an additional knob, offers an extra dimension of controlled solutions in comparison to the traditional MPSoC architecture. These cooperative and hierarchical control loops –e.g., the combination of traditional control loop (dotted lower box in Fig. 3.5) together with virtual sensing enabled optimized loop (upper loop in Fig. 3.5) – effectively translate user goals or QoS into one or more multiple design objectives [312, 313].

3.4.3 Cross-layer Interactions and Interventions

Two key attributes of the self-aware CPSoC– unlike autonomous computing [180], and invasive computing [353] –are adaptation of the layers and multiple cooperative ODA loops. As an example, the unification of an adaptive computing platform (with combined DVFS, ABB, and other actuation means) along with a bandwidth adaptive NoC offers a completely different approach (extra dimensions of control) and solutions in comparison to traditional MPSoC architecture. These cooperative and hierarchical control loops – e.g., the combination of traditional control loop (dotted lower box in Fig. 3.5) together with virtual sensing enabled optimized loop (upper loop in Fig. 3.5) – effectively translate application goals or QoS into one or more multiple design objectives.

On-chip self-awareness with cross-layer virtual and physical sensing and actuation is a key enabling technology for efficient use of heterogeneous architectures, and applica-

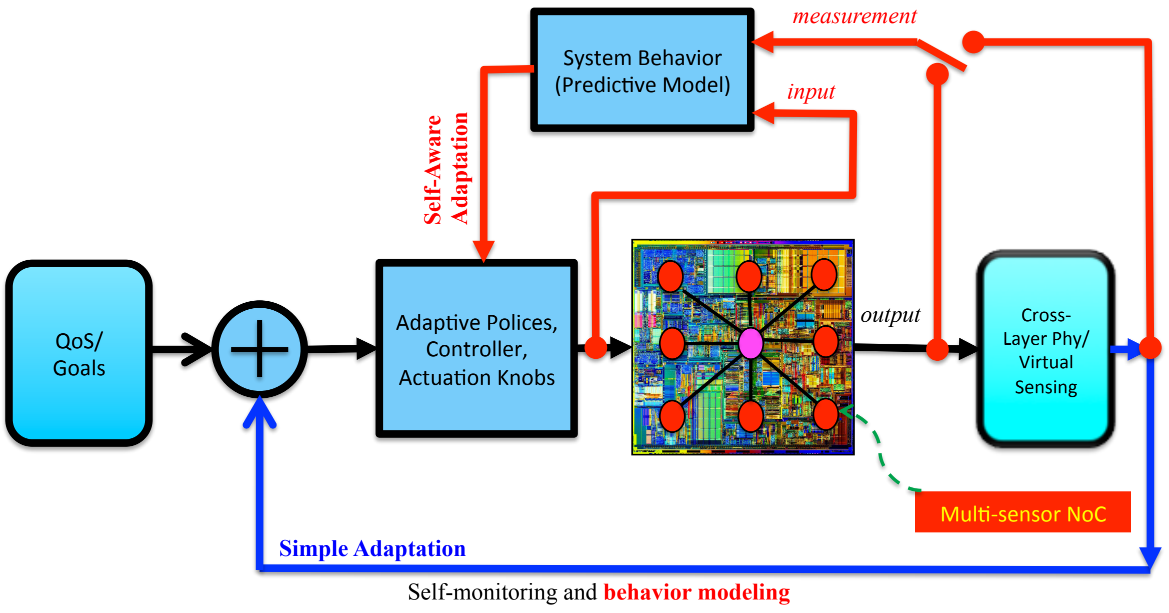
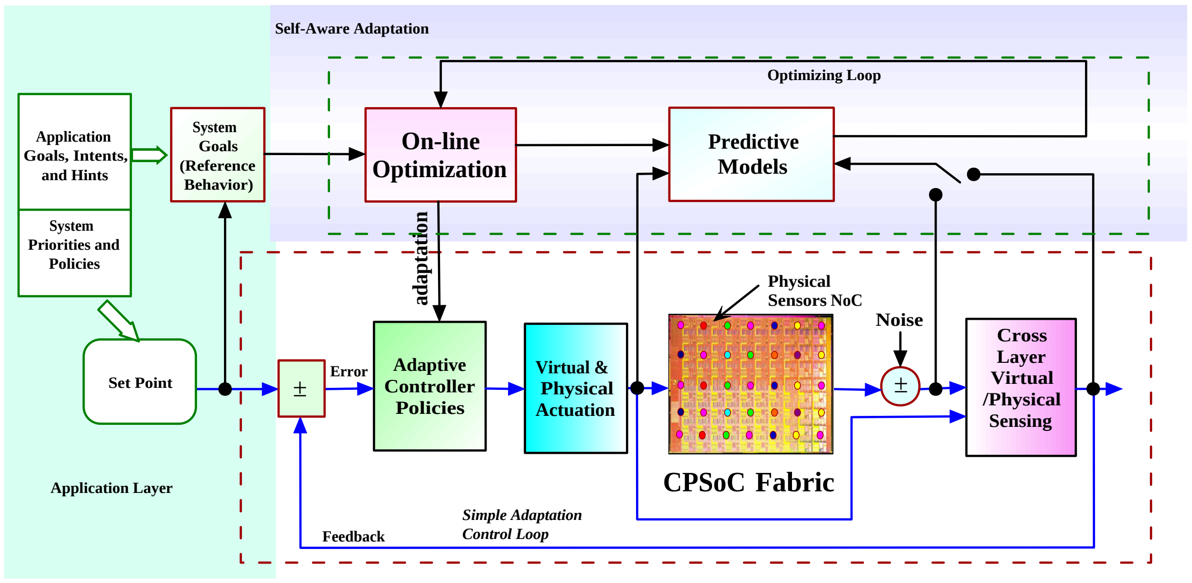

measurement
System Behavior (Predictive Model)
input
Self-Aware Adaptation
QoS/ Goals
Adaptive Polices, Controller, Actuation Knobs
output
Cross-Layer Phy/ Virtual Sensing
Multi-sensor NoC
Simple Adaptation
Self-monitoring and behavior modeling

Figure 3.4: High-level abstraction of CPSoC self-awareness.

tions with guaranteed run-time system goals and QoS (performance, reliability, power, thermal behavior) in a highly dynamic environment. The thesis demonstrates the use of multi- and cross-layer interactions and interventions for managing multiple design constraints (e.g., power, performance, thermal, resilience, aging), as well as in different design contexts (e.g., mobile platforms, data-intensive applications, long-mission applications, etc.) [306, 313, 244].

3.4.4 Predictive Models and On-line Learning

Predictive modeling and on-line learning abilities of the system behavior as well as internal and external (environmental) states provide self-modeling abilities in the CPSoC paradigm. The system behavior and states can be built using on-line or off-line linear or non-linear models in time or frequency domains [226]. We specifically use statistical and machine learning approaches [130, 110] such that the model accuracy can be traded-off for model computational complexity. We use regression based linear predictors and nonlinear predictors to build models of the system performance, power and energy consumption using the cross-layer events, hardware counters, and on-chip sensor data. In addition, use of coupling parameters (a metric that quantifies the interactions between layers) helps to develop application and cross-layer interaction models for nominal and abnormal operations. We use the predictive and learning abilities of CPSoC to improve autonomy in managing the system resources and assisting proactive resource utilization in the run-time system [312, 313].



Self-Aware Adaptation
Optimizing Loop
Application Goals, Intents, and Hints
System Priorities and Policies
System Goals (Reference Behavior)
On-line Optimization
Predictive Models
adaptation
Physical Sensors NoC
Noise
Set Point
±
Error
Adaptive Controller Policies
Virtual & Physical Actuation
±
Cross Layer Virtual /Physical Sensing
CPSoC Fabric
Application Layer
Simple Adaptation Control Loop
Feedback

Figure 3.5: Adaptation using predictive control model and policies in CPSoC [309, 313].

3.5 CPSoC Organization

The CPSoC organization consists of a combination of sensor-actuator-rich computing tiled clusters supported by different types of NoCs (cNoC, sNoC), introspective sentient units (ISU), and an adaptive reflective middleware to manage and control both the cyber/information and physical environment and characteristics of the chip. The CPSoC architecture is broadly divided into several layers of abstraction, for example, application, operating system, network and bus communication, hardware, and the circuit / device layers. The CPSoC computation fabric is embedded with on-chip sensing and actuations (OCSA) for homogeneous and strongly heterogeneous architecture as shown in Fig. 3.8. In this distributed sensing and actuation approach, each component and core can make the decision to manage the fabric. An exemplar architecture with such sensing and actuation mechanism is shown in Fig. 3.9. Note that as the sensing and actuation mechanisms in such a configuration need to be piggy backed on the traditional best effort core-to-core communication NoC, additional design considerations (e.g., real-time processing of sensors and actuators) need to be considered. On the other hand, such concern can be handed to a dedicated custom network which can be specialized to meet these requirements efficiently. One such example of a specialized sensing network to efficiently sense several distributed sensor types is shown in Fig. 3.11. The information generated by these sensor types needs to be aggregated and processed. To achieve that, CPSoC introduces a specialized unit called ISU to collect, monitor, and process the sensing data and make meaning about the system's present states and context as well as future states. Each ISU is a processor based system that is interfaced to the sNoC

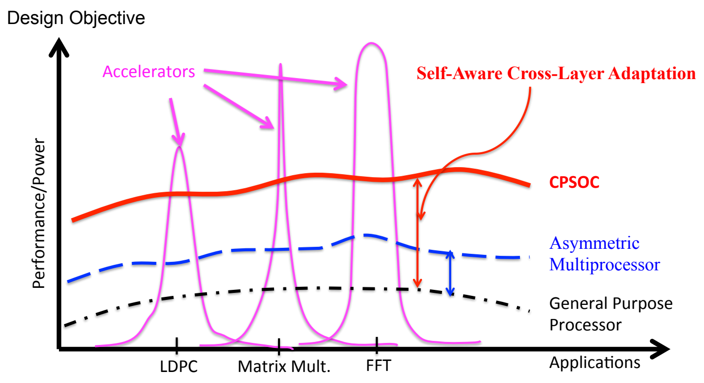


Figure 3.6: Opportunity with Self-Aware Adaptation.

where the virtual sensing approach is performed. The information from the ISUs are distributed to the computational cores. The placement and configuration of the ISUs can be determined based on the overheads and communication bottlenecks. A small core (e.g., Cortex M3) may be sufficient for low processing requirements. On the other hand, when aggressive processing is required to model and predict various phenomena in real-time, larger cores (e.g., A9) can be used.

3.6 CPSoC Architectural Components

Since the CPSoC platform introduces several new hardware and architectural modules for on-chip sensing, analysis, and actuation, in this section, we analyze the overheads incurred by these architectural assists.

3.6.1 Heterogeneous Tiled Cluster Architecture

CPSoC fabric is characterized by heterogeneity at multiple levels of abstraction and granularity. Fig. 3.12 depicts the various architectural configurations due to heterogeneity of the cluster and the tiles. In Fig. 3.12 (a), all the clusters and tiles are uniform

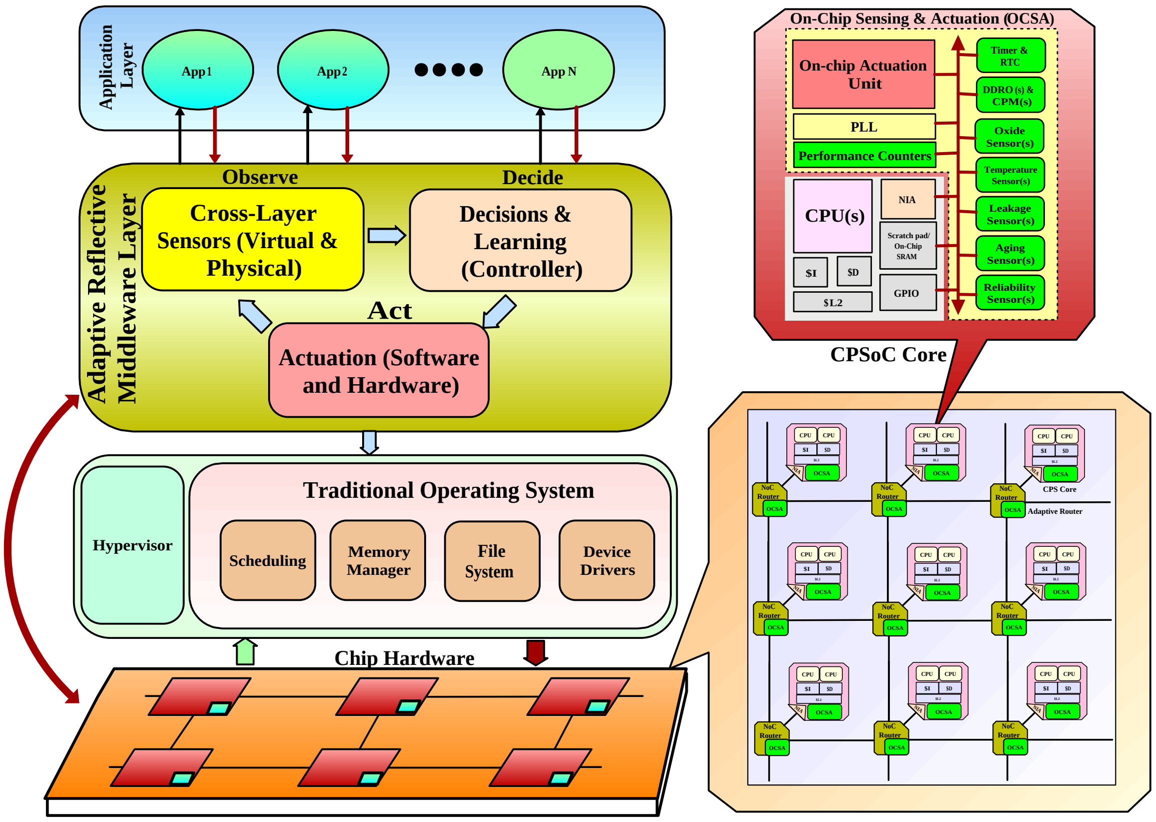


Figure 3.7: CPSoC architecture with adaptive Core, NoC, and the Observe-Decide-Act Loop as Adaptive and Reflexive Middleware[309, 313].

resulting in a perfect homogeneous tiled cluster, while Fig. 3.12 (b) depicts the architectural configuration with heterogeneous tiles and cluster. Similar to the previous two configurations, we can also have heterogeneous architectural configurations where heterogeneity is introduced in the cluster size as well as jointly in cluster size and tiles respectively.

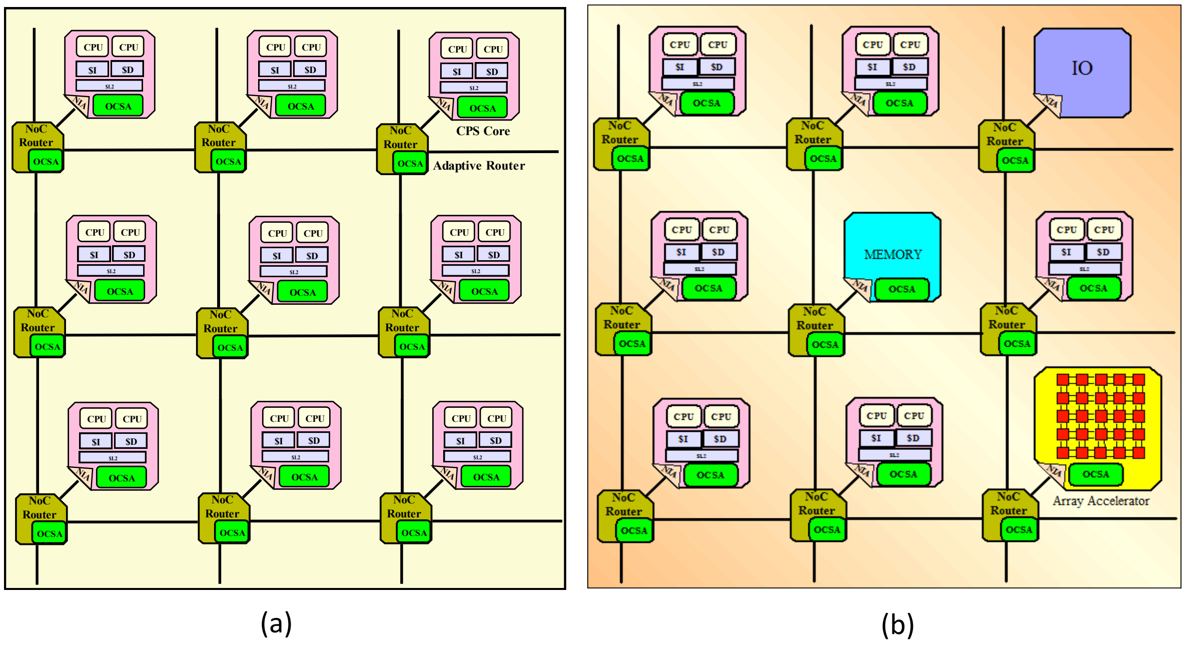


Figure 3.8: CPSoC Computational Fabric with on-chip sensing and actuations (OCSA) for (a) homogeneous titled architecture (b) strongly heterogeneous architecture.

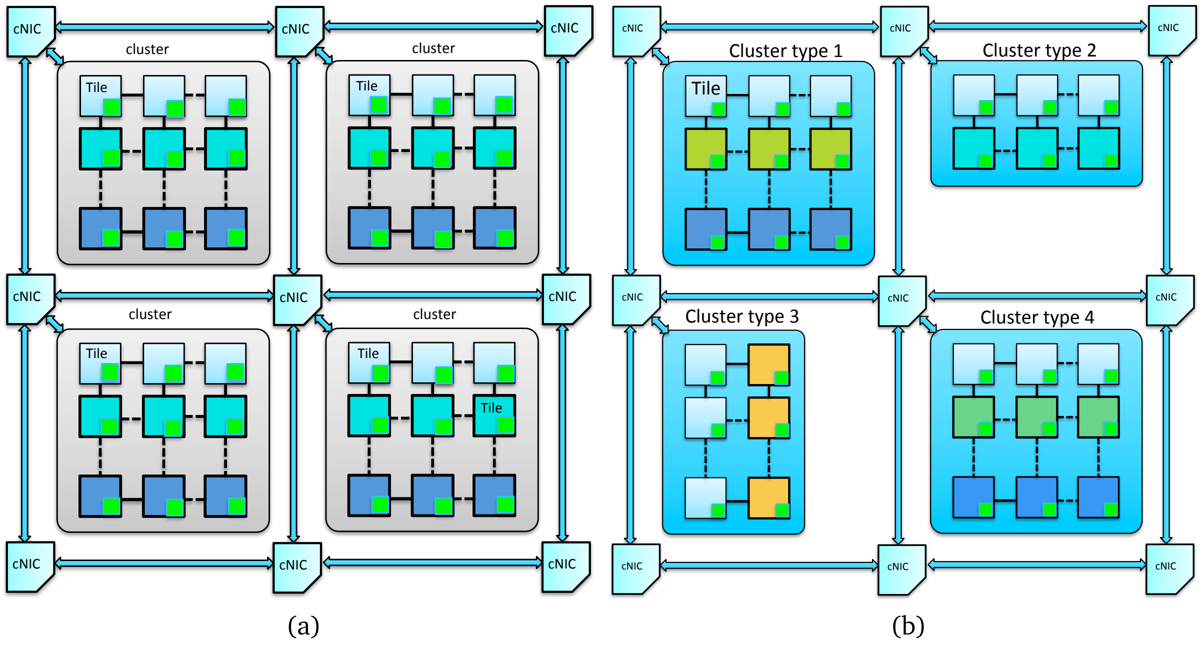


Figure 3.12: Homogeneous and Heterogeneous Tiled Architecture of CPSoC fabric (a) homogeneous cluster and tile (b) heterogeneous cluster and tile.

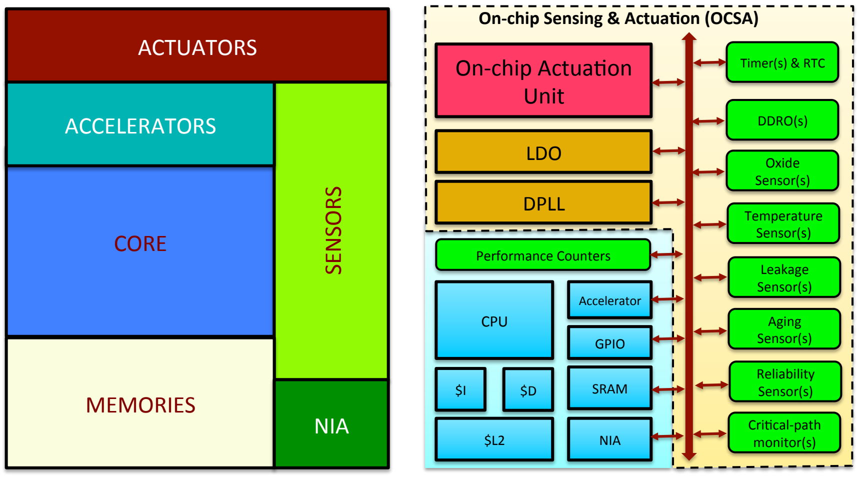


Figure 3.9: CPSoC core with on-chip sensors and actuators connected to the local bus. The cores have independent L1 cache and may share L2 and L3 cache with other cores.

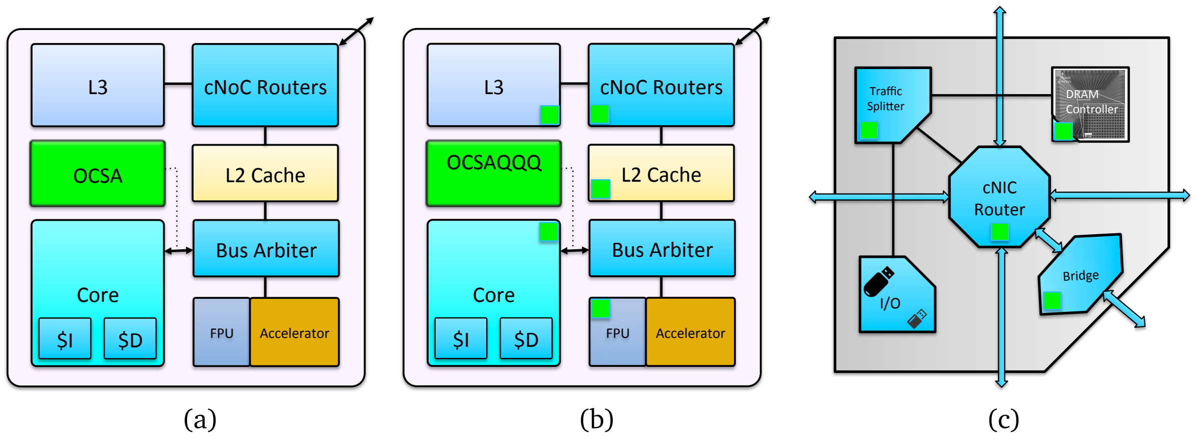


Figure 3.13: Architecture of the tile and its different configurations (a) tile with on-chip sensing and actuation (OCSA) and accelerators (c) tile with distributed OCSA (d) Cluster network interface chipset (cNIC).

3.6.2 Tiles and cNIC

The architecture and configurations of the tile are shown in Fig. 3.13. The base line tile architecture consists of a SPARC cores [115, 151] with L1 cache, private L2 cache, on-chip cNoC routers, and distributed L3 cache, along with optional floating point unit (FPU) and accelerators. In order to support self-awareness and actuation mechanisms, the core is modified to include the on-chip sensing and actuation (OCSA) unit in the

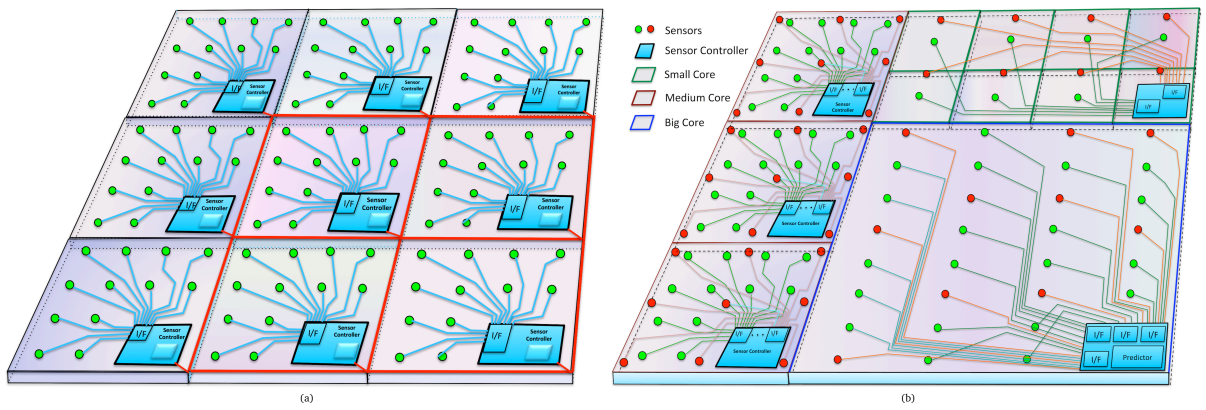


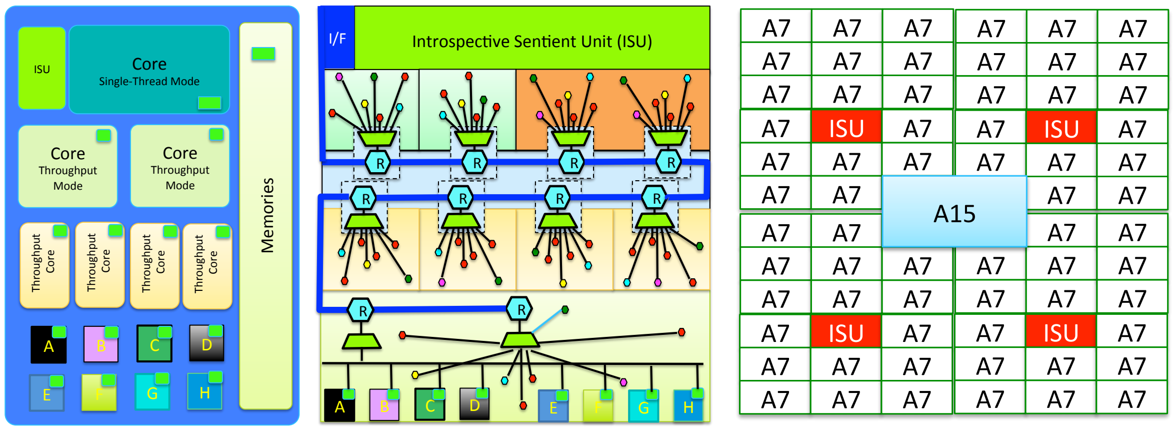
Figure 3.10: Distributed sensor networks-on-chip for homogeneous and heterogeneous architectures.

tile. The on-chip L2 and L3 caches connect directly with the cNoCs and use the bus arbiter interface (e.g., CPU-cache-crossbar (CCX) of OpenSPARC) to connect the cores, the caches, FPU, I/O and other components. The crossbar bus arbiter demultiplexes memory and floating-point requests from the core to the L2 and FPU and arbitrates responses back to the core.

3.6.3 Processor Cores

CPSoC platform can be built using processors with different instruction set architectures (ISA) that provide the ability to reconstitute or compose them (with varied component specifications) during the design time along with the development, programming, and configuration tool chains. Open source processor cores (see Table 8.1) that provide the necessary tool chains are ideal candidates and we consider the following cores for the CPSoC FPGA prototyping and emulation. In this work, we use the SPARC cores (OpenSPARC T1 and Leon cores) because of the industry-hardened design, multi-threaded capability, simplicity, and modest silicon area requirements. In addition, these open frameworks have a stable code base, a matured ISA with compiler, OS, and validation test suite.

In order to build a heterogeneous CPSoC architecture and reduced programming complexity, cores of the same ISA are modified with different resources to get heterogeneous cores. Heterogeneous architectures with different ISA can also be built, but the added complexity and tool compatibility are limited in our present development. For a selected processor type and ISA, the IP cores and peripherals have to follow the bus standard of the processor. However, in order to facilitate the library components' use with different bus architectures, we develop bus converters and bridges such that components can be interfaced to a processor with different bus specifications. Our design of the different components of the library are AMBA, AXI compatible and we provide



(a) (b)

Figure 3.11: CPSoC with the distributed sensor network (sNoC) and the introspective sentient unit (ISU). (a) A ring network with a time division multiplexed (TDM) router connects all the distributed sensors of different types in cores, memories and accelerators. (b) placement of multiple ISU for distributed sensing.

bus converters (e.g., CCX to AXI, AMBA to Wishbone, AXI to APB and vice versa) to interface with other processor bus specifications.

3.6.4 Memory Hierarchy

The L3 cache is distributed write-back cache shared by all the tiles in the cluster. The default cache configuration is 64KB per tile and 4-way set associativity with 64 bytes per cache line and 64 bit per entry but both the size and associativity are configurable.

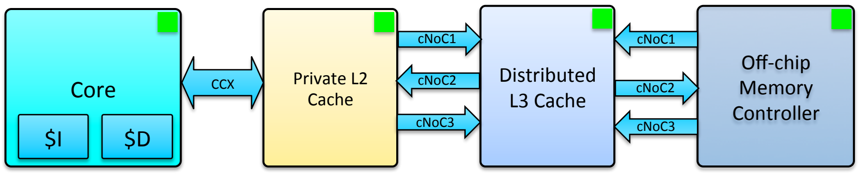


Figure 3.14: Memory hierarchy showing the data path and interfaces.

3.6.5 Interconnects

CPSoC deploys different types of networks and interconnects in order to observe and act in a distributed way. The following subsections briefly discuss each of these networks.

3.6.5.1 Network-on-Chip (NoC)

The core-to-core NoC (cNoC) is an integral part of the tiled distributed architectures of CPSoC. There are three cNoCs similar to the scheme in [28] that connect the tiles in a 2D mesh topology with purpose of providing communication between the tiles for cache coherence, I/O and memory traffic, inter-core interrupts. In addition, to routing traffic destination for other cluster through the cNIC chipset bridge, the cNoCs maintain point-to-point ordering to maintain consistency. The cNoCs implementation uses physical networks, credit base flow control, and wormhole routing to ensure a deadlock-free operation.

3.6.5.2 cNIC Cluster Bridge

The cluster bridge connects the tile array to the cluster network interface chipset (cNIC) as shown in Fig. 8.3. All memory and I/O requests are directed through the bridge interface to be served by the cNIC. The bridge transparently multiplexes the cNoCs over a single link and provides a link between I/O and core clock domains. The bridge is connected to a cNIC router that implements virtual channels over a off-chip physical channel providing arbitration logic. In addition to traditional credit based flow control in the cNIC router, we introduce a software-defined configurable router with bandwidth, channel direction, and routing policy control to achieve improved adaptivity and control of the inter-cluster networks as shown in Fig. 3.15.

3.6.5.3 cNIC Chipset

The cluster network interface (cNIC) chipset connects multiple clusters in the CPSoC prototype and it consists of the DRAM controller, the cluster bridge, the intra-chip network router (cNIC router) and the I/O interfaces.

3.6.6 Actuation Networks-on-Chip (xNoC)

The CPSoC platform can support many actuation mechanisms across several layers; here we consider emulation and prototyping of these mechanisms in the FPGA architecture. The actuation mechanisms in the software layers can be supported with modifications to application, programming models, and the runtime system. However, the mechanisms in the networking, and architecture layers require hardware design modifications. We illustrate the support of some of these mechanisms in the FPGA platform. We consider the dynamic frequency scaling (DFS) per core using a digital phased locked loop (DPLL) implementation and providing the DPLL control directly to the runtime system and OS as shown in Fig. 3.16. Another actuation mechanism that can easily be implemented is clock gating which is an extension or special case of DFS. We introduce

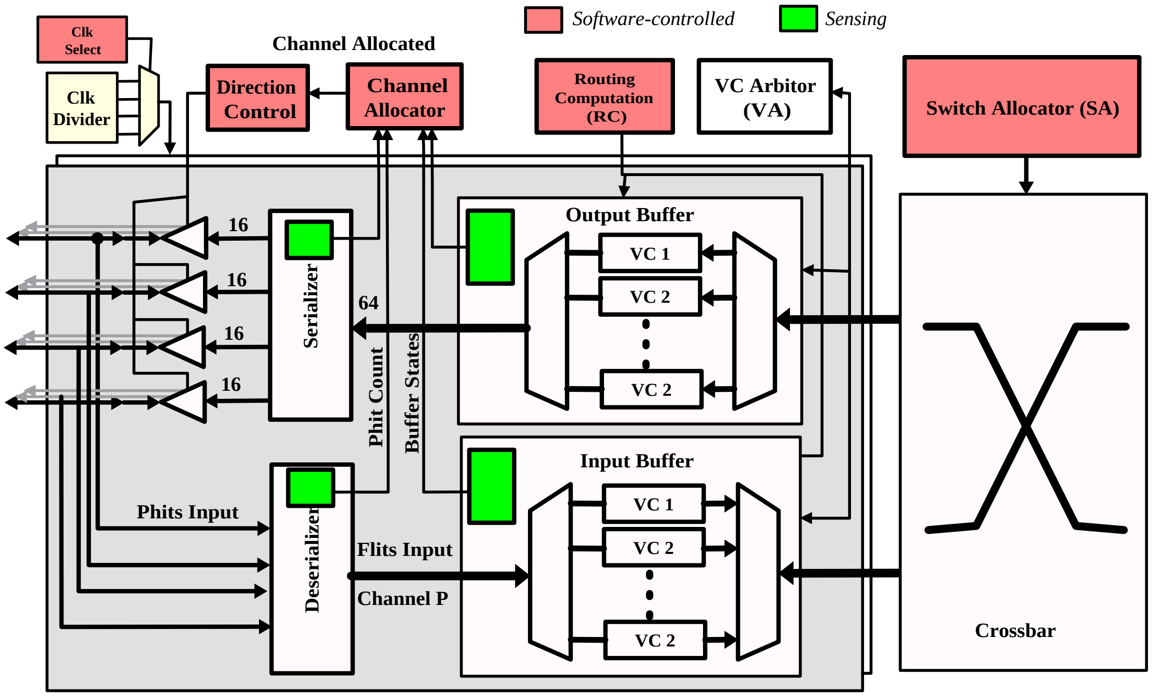


Figure 3.15: A software-defined adaptive router for cNIC with bandwidth, channel direction, and routing policy control.

the heterogeneity of cores by different cache sizes and number of functional components (e.g., integer unit, FPU, timers etc.) that provides performance-power trade-offs for the runtime system to exploit.

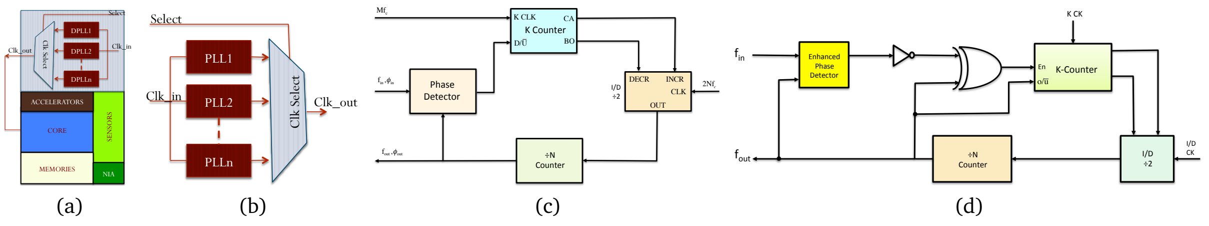


Figure 3.16: Clock frequency adaptation and control in CPSoC platform (a) clock scheme (b) clock selection (c) DPLL (d) DPLL with jitter reduction.

3.6.6.1 Fusion of Actuation Mechanism

Actuator fusion combines different types of actuators across different layers of the system stack into virtual actuators. By fusing actuation mechanisms from the highest levels (e.g., application and software) together with lower layers (OS, hardware, and circuit layers), we can exploit wider opportunities for controlling system QoS properties (e.g., performance, power, thermal, and reliability guarantees). Actuator fusion affords several advantages: graceful degradation of system performance in the face of actuation mechanism failures; faster response and more dynamic range of actuation; use of simpler (low-overhead) actuation mechanisms for virtual actuation; and repurposing of different actuation mechanisms to overcome specific constraints and malfunctions.

Actuator fusion can be used to guarantee runtime system QoS (performance, reliability, power, thermal behavior) even in a highly dynamic environment. As a specific example, consider a manycore architectural stack executing heterogeneous applications/workloads (for example, probabilistic/RMS and safety-critical workloads). Also consider three actuation mechanisms that are traditionally applied independently to improve performance within the power consumption goals: loop perforations in the application layer, task allocation in OS layer, and DVFS in the hardware layer. The fusion of these three actuation mechanisms (loop perforation, task allocation, and DVFS) enable aggressive power savings that would not be possible beyond a critical voltage and frequency with simple DVFS. Once this critical limit is reached, any further reduction in voltage/ frequency results in exponential degradation in system reliability and error characteristics. Loop perforation in the actuator fusion framework enables additional power saving without jeopardizing the system reliability and dependability. This additional dimension of control provides added flexibility in improving the range and speed of qualities of service and experience of computing platforms.

3.6.7 Middleware and OS Support for Adaptation

The word “adaptation” has several meanings in the systems community. However, the most accepted notion of adaptation is defined as the ability to adjust and improve system response using feedback, the ability to respond to change along with the capability to alter itself dynamically to achieve goals. From a software engineering perspective, a system is adaptive when it allows for modifying its structure or behavior at runtime; i.e., without interrupting its service by coupling the system with an adaptation manager. On the other hand, from a control engineering viewpoint, adaptation or adaptive control adds a degree of flexibility to the control mechanism, where the controller may change its own control policies, structure, or control parameters dynamically. We reconcile both these viewpoints by defining a generalized closed-loop structure where the system monitors and detects the changes, analyzes their impact, and, if needed, plans and executes actions in response to the changes. In addition, knowledge about the system that is captured by suitable models updated at runtime is used to support adaptation. We specifically focus on the operating system support needed for adaptation of emerging heterogeneous MPSoCs for achieving system goals (e.g., energy efficiency) dynamically.

Middleware is a computer software that provides services to software applications beyond those available from the operating system. Adaptive middleware is a software whose functional behavior can be modified dynamically to optimize for a change in environmental conditions or requirements. The requirements of runtime adaptive system include measurement, sensing, control, feedback and stability. Adaptations can be triggered by changes in the environment, change in the configuration or policy, instructions from another program, and user requests. Reflection is a technique that enables adaptation. Reflection is the integral ability of a program to observe or change its own code as well as all aspects of its programming language - even at runtime [336]. Structural reflection provides the ability to alter the statically fixed internal functional structures. Structural reflection changes the internal makeup of a program. On the other hand, behavioral reflection enables the ability to intercept an operation such as a method invocation and alter the behavior of that operation. Behavioral reflection alters the actions of a program. Reflective middleware moves reflection to the middleware level.

CPSoC presents a general computational model for enabling applications and system to cooperate in a self-aware manner. In the CPSoC paradigm, application and system goals are considered jointly, system runtime decides possible actions and takes the actions to achieve and meet those goals. The self-awareness computing framework is implemented using the adaptive middleware that implements the *observation*, *decision*, and *action* phases of an ODA loop. The ODA loop is the characteristic of a control system; during the observation phase the system collects information, which is fed to the decision phase. During the decision phase the system determines whether recent observations require a change in behavior and, if needed, what form this change should be performed. If adaptation is desired, the action phase implements the adaptation dictated by the decision process. The CPSoC paradigm, as shown in Fig. 3.7, supports this form of closed-loop execution by generalizing the observation and decision phases, providing standard techniques that work with a broad range of actuation mechanisms (actions) that is available in adaptive middleware layer. The adaptive middleware exhibits some of the properties of reflection, specifically behavioral reflection, to alter the behavior of the system and application in order to adapt to changes in workload and environmental condition as discussed in Chapters 7 and Chapter 6.

3.6.8 Self-Awareness Properties and Levels in CPSoC

In Section 2.5, we described a necessary set of properties for a CPSoC to become self-aware based on the classification levels. For a CPSoC to be self-aware in the ideal sense, it should exhibit all of these properties at the highest level of fidelity and with maximal range of attributes. However, depending on the context of the engineered system and the use case, it may not need all of these attributes, nor does it need to have maximal coverage in terms of the fidelity and range of attributes. Indeed, in the CPSoC context, we are typically resource constrained, with limited area, power, and thermal budgets. Thus we need to consider carefully how to incorporate different self-awareness attributes, and at what levels they can be designed within the overall envelope of all the SoC design constraints. Towards that end, we describe below how the CPSoC exemplar has incorporated the self-aware properties described earlier. Recall that CPSoC supports four main concepts: (1) physical and virtual sensing and actuation, (2) self-monitoring and adaptation, (3) cross-layer interactions, (4) predictive modeling and learning.

Physical and virtual sensing collects the primary data but also does a great deal of abstraction with specific goals in mind. Disambiguation is performed partially and implicitly but not in a formal manner. Hence, CPSoC offers implicit **semantic interpretation**.

A **desirability scale** is built into the system for each sensor or their combination in the virtual sensing approach. All the functions – from the collection of sensory data to the control algorithms and the actuation mechanisms – use this desirability scale for accurate sense-making and deriving insights (e.g., how the system should perform and what constitutes a malfunction). CPSoC may appear to use an implicit desirability scale but in fact uses an explicit one through mapping functions and calibration tables as discussed in the implementation of a temperature sensor in Chapter 4. An explicit mapping function (based on look-up tables (LUTs)) in a ring oscillator based thermal sensor transforms the ring oscillator frequency to that of actual temperature reading on the die (measured in $^{\circ}C$) is used as the desirability scale for thermal awareness. Moreover, as the mapping functions and LUTs are virtualized in software as discussed in Chapter 4, the provision for re-purposing these sensors, for example, a ring oscillator based leakage power or aging sensor can be realized. An explicit desirability scale has the advantage of increased flexibility by decoupling observations from decisions. When new types of sensory data and new observations have to be interpreted, they are explicitly mapped onto a desirability scale allowing the control and decision algorithms to remain unchanged while still taking the new information into account. Hence, we consider that CPSoC has an explicit desirability scale while performing **semantic attribution** implicitly.

CPSoC keeps track of the **historic evolution of properties** by using explicit notions of epoch and sampling time [313, 310]; hence it embodies this dimension of awareness as illustrated in Chapter 7. As an example, the state space dynamic models in [309, 310] uses explicit notions of states in the previous epoch to predict the states value in the current epoch. Additionally, sensor data across multiple epochs are stored as history in order to make an assessment of the average behavior of certain states of interest.

CPSoC has clear **goals** such as the maximization of energy efficiency and the ability to detect and tolerate faults and failing components. Again, these goals are configurable and can be specified, selected, or changed in the control algorithms, which make them efficient and effective and provide flexibility to adapt to new goals during the system's lifetime. As an example, the online Simulated Annealing based optimization scheme in Chapter 7 is made configurable to accept or change the objective function at run-time by using a Linux system call for performance maximization while achieving energy efficiency.

The **purpose** is only defined partially, because CPSoC is a *platform* that can be used in a range of applications. By definition, a platform will always only define some of the system's goals, such as detecting and tolerating faults, but will leave the definition of other goals to the application as discussed in Chapter 5.

CPSoC being a platform with system level goals (e.g., thermal efficiency) is aware of some aspects of the environment (e.g., the ambient temperature) and has consequently **expectations on the environment** in addition to the **expectations on itself** as encoded in the sensing and control algorithms. However, these expectations are implicit and reactions to environmental changes are limited to specific cases corresponding to the system-level goals.

The **inspection engine** is based on prediction models (or their variants) as discussed in Chapter 4, 5 and 6, thus providing the capability to inspect state variables in time and space.

In summary, the design of the CPSoC framework has carefully considered how to balance the needs of self-awareness in an SoC context, and has realized different self-aware attributes with the dual goals of maximizing self-awareness, while minimizing overheads and simultaneously meeting the complex, interdependent set of constraints faced by the system. Hence, it can be considered as a self-aware system as defined in Section 2.5 and in our earlier work [161, 310, 104], to the extent reasonable in an SoC context. Indeed, it should be noted that more awareness is not necessarily better. There is a trade-off between the cost of awareness, the efficiency of implementation, and the flexibility and generality that would come with higher degrees of awareness. This is particularly important in the specific context of self-aware SoCs that must meet a multitude of cross-purpose constraints.

3.7 Summary

Emerging SoCs arguably face an even more complex set of conflicting constraints, in the face of highly dynamic workloads, as well as process and environmental variability. Furthermore, with increasing complexity of, and heterogeneity in the SoC platform architecture, there is a critical need for these SoCs to be self-aware, and perform in an adaptive manner. In this chapter, we presented the rationale and concept of self-aware adaptive SoC paradigm called Cyber-Physical-System-on-Chip (CPSoC). We presented the architectural features and attributes that distinguish CPSoCs from traditional MPSoCs. We described the models of the CPSoC architecture that can be implemented using existing open framework, tools and methods. The chapter introduced the overview of the different components of the CPSoC paradigm and an outline connecting the constituting parts with details in subsequent chapters. We presented the CPSoC platform as an exemplar self-aware heterogeneous manycore SoC platform that achieves self-aware adaptation through a principled orchestration of ubiquitous (virtual) sensing and actuation, coupled with health-monitoring and predictive model building. We then briefly described how each of the self-awareness properties is manifested in the CPSoC platform. Since these facilities must be tightly woven into the SoC's hardware and software fabric, CPSoC's self-awareness properties have been engineered carefully to prevent the excessive overheads of intrusive sensing/actuation.

Chapter 4

Multi-Sensor NoC (sNoC) for Self-Awareness in CPSoCs

4.1 Introduction

Increasing integration densities results in increasing number of processing elements, storage components, and growing pool of interfaced intellectual property (IP) in a single multicore SoC. This trend gives rise to increasing heterogeneous architecture, massive complexity, and wide dynamic behaviors in multicore SoCs. The design intent and the expected quality-of-service while executing demanding applications on such multicore SoC require matching different system services and platform resources. Run-time adaptive mechanisms requiring online monitoring and sensing are increasingly utilized to address the challenging application need and achieve key design objectives as well as functional requirements. Observation, aggregation, reconstruction, fusion, and processing of key pieces of information play a central role in creating different monitoring and awareness mechanisms, architecture, and implementation methods.

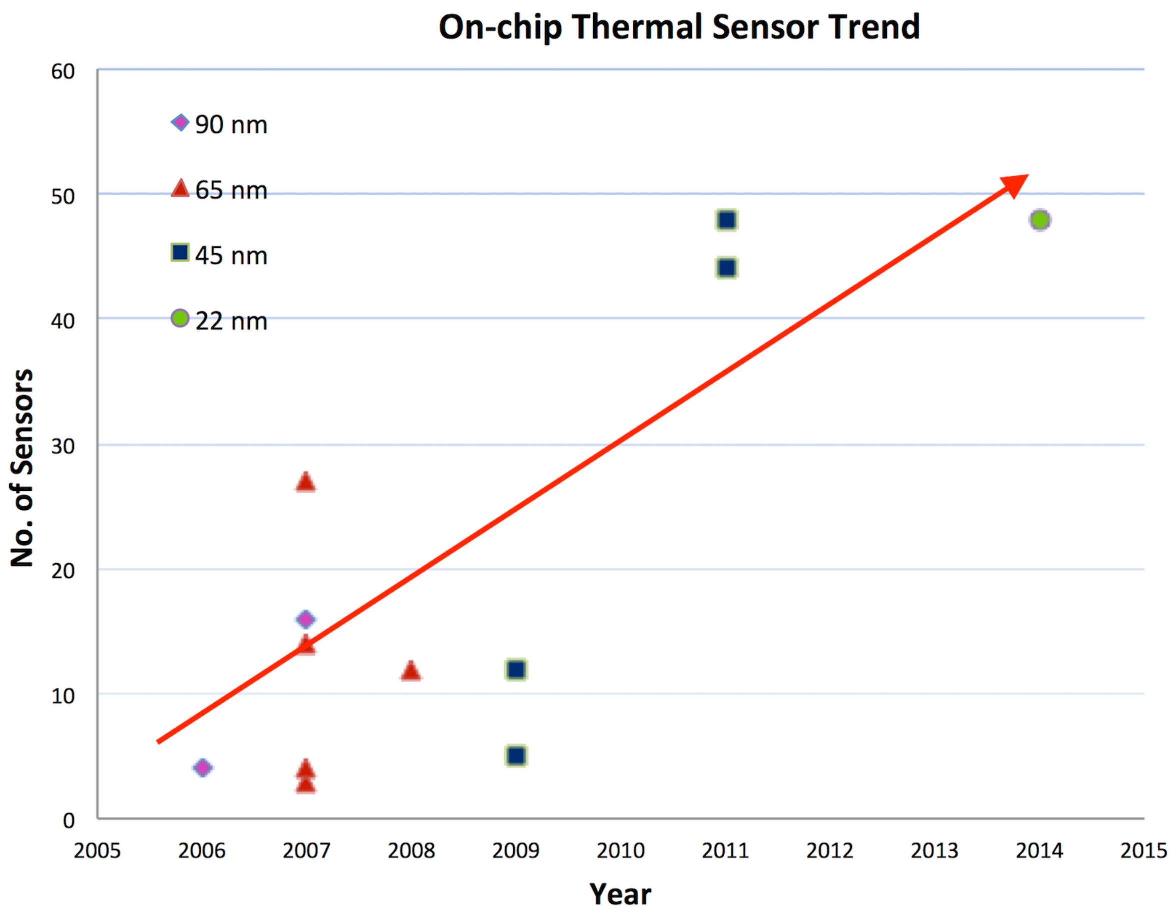


Figure 4.1: Recent trends in numbers of on-chip temperature sensors in microprocessors and SoC [381].

Infusing self-awareness in the CPSoC paradigm requires on-chip sensors and monitors that are carefully designed, interfaced, and placed at suitable locations in the die. The number of sensors required to monitor a given awareness dimension, for instance, thermal behavior of the emerging SoC, is going to increase, as shown in Fig. 4.1, in order to provide accurate monitoring and good fidelity of the observed on-chip phenomena. Similarly, as the need for improved awareness increases for better characterization of multiple phenomena such as aging and NBTI, oxide breakdown, electromigration, timing error violations as well as leakage power distributions, sets of multiple sensors of different types along with their sensor specific interfaces need to be included in the design.

CPSoCs are uniquely distinguished by the inclusion of such diverse sets of sensors and their associated interfaces in the design and architecture. The inclusion of multiple sensors and their interfaces are non trivial and require careful consideration and design planning in order to reduce the design complexity, verification effort, as well as design overhead. As a solution, the CPSoC paradigm proposes a multi-sensor networks-on-chip (sNoC): customized for different sensor types, suitably placed considering several design and placement constraints, as well as a scalable sensing and monitoring architecture to support thousands of such sensors for multi and manycore architecture of the future. In this chapter, we discuss the design approach of such multi-sensor networks and formulate an efficient sensor placement and fusion problem and their optimizing solutions to implement and improve several self-monitoring and awareness properties in CPSoCs. We present a novel design time and runtime approach to sensor placement and sensor fusion using on-chip sensors networks, redefine the sensor allocation and placement problem using a combination of heterogeneous sensors, as well as present an algorithm to find the best sensor combination and their location without either surrendering the accuracy of thermal monitoring or exceeding the sensor area power budget. We show that significant improvements in sensing accuracy and reconstruction error (around 10-100$\times$ compared to the state-of-the-art) is possible with three different sensor types for the same overheads as well as reduction in algorithm execution time by over $20\times$ in comparison to the state-of-the-art technique.

4.1.1 Sensor Network-on-Chip (sNoC)

CPSoCs are distinguished by their ability to be aware of different self-aware properties across the stack as discussed in Chapter 2. CPSoC deploys specialized on-chip sensor networks for self-monitoring of on-chip parametric variations and dynamic behaviors in order to deal with the multitude of challenges stemming from worsening process and other sources of variability. These on-chip sensor networks in emerging multicore architectures are becoming critical to meet the varying application requirement at runtime and wide spectrum of qualitative requirements in terms of optimum performance, predictability, fault diagnosis, recovery, as well as power efficiency. Unlike the traditional NoCs, these on-chip networks are distinguished by their unique properties and features as discussed below.

4.1.1.1 Properties and Features of Multi-sensor NoC (sNoC)

Sensor-networks-on-chip (sNoC) deployed by emerging SoCs are characterized by the runtime adaptation requirements and the quality and level of awareness these networks are required to provide. Although these networks require specific features based on each instance, we highlight some general properties and features below.

- **Temporal Timescale of Monitoring:** Effective monitoring is needed to effectively handle slow as well as fast changing effects. In order to select a suitable sNoC architecture, Fig. 4.2 categorizes the discussed variations according to their time constants and sampling requirements. As the dynamics of variation effects depend on various circumstances, the shown classification is rather qualitative. While process variations occur during fabrication and lead to fixed changes in device parameters, device characteristics can, however, still be affected by aging induced wear-out during operation. Significant degradation of CMOS logic delays mainly develops in the long run. The temperature, also influencing circuit speed, changes rather slowly, whereas power noise has normally very short time constants. The shorter the time constant of a variation effect, the more challenging becomes the monitoring and adaptation mechanism.

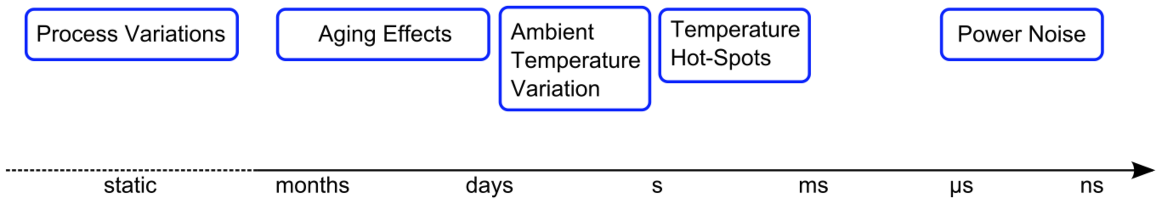


Figure 4.2: Temporal classification of variations.

- **Sampling Method:** In order to infer the temporal characteristics of a dynamic phenomenon, sensors readings are performed according to sampling policies, for example, uniform or probabilistic sampling (random, stratified, cluster).

- **Accuracy:** Accuracy of the independent sensor impacts the overall accuracy of the system monitoring and the spatial and temporal reconstruction of a phenomenon. Not only the number of sensors, but their accuracy as well as area-power overhead are design parameter as the number of sensors embedded increases rapidly in emerging architectures (see Fig. 4.1).

- **Reliability:** In order to have a reliable reading of the sensors, several sensors can be tallied using majority voting schemes of nearby sensors. Redundancy of the sensors reading could be used to improve reliability of the sensor reading in the sNoC.

- **Calibration:** Sensor calibration is performed using single point or two point calibration methods. Calibration is performed by designating a reference sensor in the sensor network or a more accurate sensor in the sNoC.

- **Scalability**: The trend towards massively parallel (100+) cores and hardware accelerators integrated on 3-D stacked dies calls for scalable control algorithms running in a few microseconds. Distributed control algorithms are needed that leverage the spatial localization of heat exchange and which can exploit parallel hardware.

- **Spatial Distribution:** The distribution of the sensors in the die to capture the underlying spatial phenomena is performed at the system level or core level. For example, if a multicore SoC consists of several cores, the whole system should be considered in deciding the sensor allocation. In a tile based architecture, the sensors are distributed in a tile and the design is replicated across the whole chip.

- **Topology:** The topology of the sNoC can be either distributed or centralized. Measurement from the sensors are collected at a collection node through the interconnection bus or signal between the sensor interface and the collection point.

- **Data Packet Optimization:** Sensor communication is optimized through packing multiple sensors into single read operations. With the exponential growth of cores, the number of on-chip sensors of different types would also see exponential growth.

- **Robustness:** Sensor readings are affected by significant output noise. System monitoring and awareness approaches are needed that are robust to measurement and process noise. Accurate on-chip temperature sensors have high area cost and are affected by significant systematic and random noise. In addition, to keep post-manufacturing testing costs low, not all the sensors are accurately calibrated. Hence, manycore thermal management is a large-scale, hybrid, nonlinear multivariable control problem, affected by significant sensor, actuator, and process noise.

- **Adaptability**: Fluctuations of process variations and ambient conditions (temperature, heat sink occlusion, etc.) may change the thermal behavior over the lifetime of a component. Model recalibration strategies and online system identification algorithms are required.

- **Inspection Engine:** The monitoring and sensing infrastructure could be used for rapid first silicon inspection, debug and variation analysis, IC characterization, timing margining, production test, IR-drop and temperature analysis, wearout analysis, and data collection.

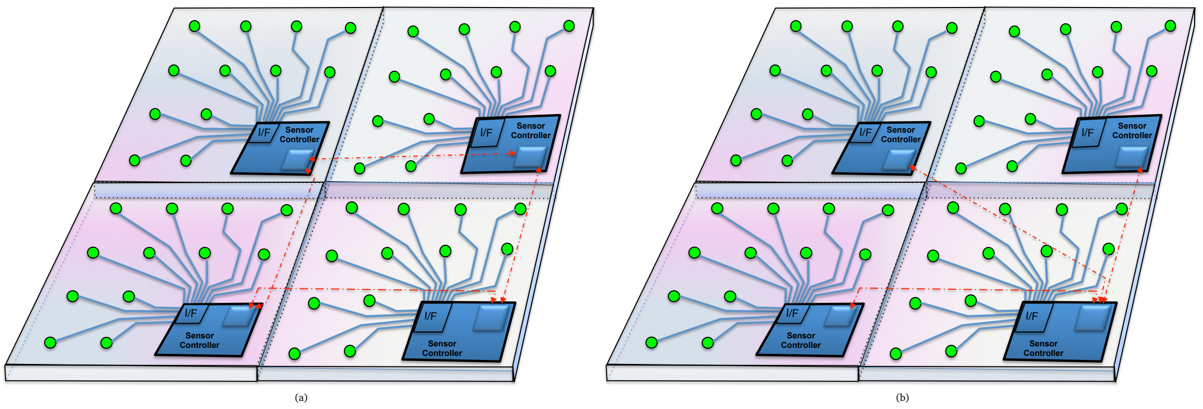


Figure 4.3: Distributed Sensors-Network-on-Chip (sNoC) for on-chip self-awareness using (a) mesh topology (b) aggregation tree topology.

4.1.2 Types of sNoC in CPSoC

An on-chip monitoring system is composed of a set of monitors that provide an output dependent on the corresponding physical magnitude; an interconnection network that delivers the data from the monitors to a controller; and a control system that can be either centralized or distributed. In this work, we focus on the interconnection network with the objective to provide a network-on-chip architecture with a high degree of simplicity, minimum impact in area and power, as well as scalability needs of emerging SoCs. Based on the review of several monitoring types and characteristics summarized in Table 4.1, we observe that the sampling frequency, quantization, and density needs vary greatly depending on the type of monitor as shown in Table 4.1 and Fig. 4.2. Some monitors, (such as voltage droop monitor) require fast response and are directly connected to a distributed control system that takes immediate actions, and thus does not use the NoC. They fall in the category of specialized monitoring infrastructure that falls out of the scope for this work.

4.1.2.1 Thermal Sensor-NoC

As integrated circuit technology continues to scale to the nanoscale era, power and thermal issues become increasingly important and major concern for processor

Table 4.1: Summary of on-chip Sensors and Monitors.

Sensing	Cause	Adverse effect	Period	Density	Quantization	Data Volume
Aging	Prolonged usage	Reduction Speed /Malfunction	$100ms$	hundreds/chip	~8 bits	691.2 MB /day
Temperature	Localized power consumption	Loss of Performance/Reliability	$100ms$	hundreds/chip	~8 bits	864 MB/day
Power	Switching and Leakage	Hot-spots and thermal runaway	μs	few-tens/chip	~10 bits	8.64 TB/day
Critical path	Timing uncertainties	Loss Speed	μs	hundreds/chip	~12 bits	103.68 TB/day
Supply voltage	Supply network impedances	Voltage droop/Malfunction	ns	few-tens/chip	~6 bits	5.1840e+03 TB/day

design [54, 334, 56]. Local on-die thermal transients have time constants of microseconds, whereas at the package and board level we see complex, nonlinear dynamics unfolding in seconds to minutes. A single chip can have hundreds of thermal domains that vary greatly in workload and intrinsic power density. Power consumption and heat generation are thus spatially and temporally heterogeneous, with nonlinear temperature dependency caused by leakage. The on-chip thermal sensor network plays a crucial role in characterizing these thermal behaviors as well as accurately identifying hotspots in the multicore system.

Earlier works [382, 269, 207, 363] have shown that elevated temperatures directly impact all key circuit metrics including: lifetime and reliability, speed, power, and costs. Thermal hotspots reduce the mean time to failure (MTTF) as most failure mechanisms (e.g., electromigration, time dependent dielectric breakdown, and negative bias temperature instability) have strong temperature dependencies [269]. With more than 50% of all integrated circuit failures being accounted to thermal issues [269], a mere 10^oC – 15^oC rise in the operating temperature could halve the life span of the chip [363]. Not only does the fault rate double for every 10^oC increase in temperature [207], but different thermal expansion coefficients of chip materials also cause mechanical stresses that can eventually crack the chip/package interface [56], resulting in increased packaging cost. Additionally, accuracy of thermal measurements directly affects the efficiency of thermal management as well as the performance of the CPU [293, 388]. Inaccuracies in thermal tracking decreases the processor's performance and wastes power. In particular, it was shown that a 1 ^oC accuracy translates to 2W power savings while 1.5^{o}C accuracy in temperature measurement is equivalent to 1W of CPU power in mobile computers [293]. Due to localized heating, the temperature variation within a single chip can reach up to 10s of degrees. For instance, within-die temperature variation of up to 50 ^oC was reported in [54]. Due to lack of proximity, sensor measurements and hotspot temperatures could differ by up to 10 ^0C [293]. Such inaccuracies in thermal estimates can either trigger early or late activation of dynamic thermal management (DTM) resulting in unwanted performance loss [388] or severe reliability degradation [269].

As the effectiveness and efficiency of dynamic thermal management approaches heavily rely on the accuracy of on-chip temperature measurement, several types of thermal sensors varying in accuracy, resolution, area, and power consumption have been proposed. The specifications and relative area and power characteristics of some recent smart sensors are listed in Table 4.2 and Fig. (4.4) respectively. For an area limited

Table 4.2: Comparison of Recent Smart Temperature Sensors.

Sensor	Resolution(^0C)	Accuracy(^0C)	Range(^0C)	Power	Area(mm^2)	Technology	Reference
S1	0.139	-5.1∼+3.4	0∼60	150μW@1.0V	0.01	65nm	Chung et al.[79]
S2	0.16	-0.7∼+0.9	0∼100	0.49mW@3.3V	0.175	0.35μm	Chen et al. [72]
S3	0.0918	-0.25∼+0.35	0∼90	36.7μW@3.3V	0.6	0.35μm	Chen el al. [71]
S4	0.01	± 0.1	-55∼125	247μW@3.3V	4.5	0.7μm	Pertijs et al. [270]
S5	0.03	± 0.2	-70∼125	10μW@1.2V	0.1	65 nm	Sebastiano et al. [316]

design, sensor S1 would be preferred. On the other hand, for a power limited design, sensor S3 would be preferred over S1 because of its lowest power characteristics. For design spaces between the two extremes of area or power limited design, neither S1 nor S3 but a combination of sensors (e.g., S1, S2, S3) would potentially provide a better solution by leveraging the diversity of each sensor type. Additionally, in order to accurately capture the wide within-die temperature variations, a large number of them must be deployed throughout the chip to collect thermal data in real time. For example, Intel's Dunnington Xeon processor with 6 cores contains 12 thermal sensors [204], AMD's quad-core opteron deploys 38 sensors [99], and IBM's POWER7 microprocessors includes 40 temperature sensors [371] at different locations. This trend is depicted in Fig. 4.1.

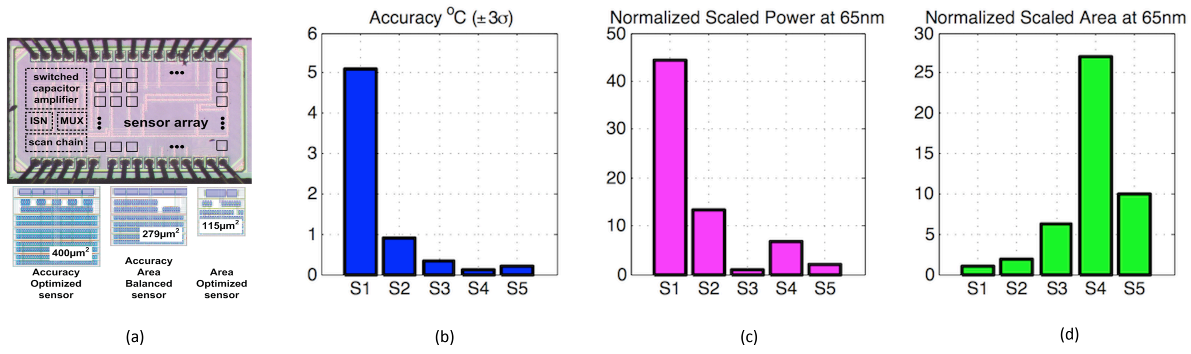


Figure 4.4: (a) Heterogeneous on-chip thermal sensors at 65nm [381] (b) sensor accuracy, (c) normalized power and (c) area of the digital temperature sensors of Table (4.2) scaled to 1.0V, 65 nm technology.

Ring oscillator (RO) based on-chip thermal sensors are most common as they are fully compatible with digital CMOS technologies while consuming small area and power. Fig. 4.5 (a) shows the basic structure of an RO-based on-chip thermal sensor that has an odd number of inverters. The transition time for each inverter to switch levels (H-L or L-H) is determined by several factors [388]. The high-to-low time t_{PHL} for an NMOS and low-to-high transition time t_{PLH} for a PMOS are given respectively by [388]:

$$\begin{array}{c} t_{PHL} = \frac{C}{\mu_n (W/L)_n (V_{DD}-V_t)} \left[\frac{2V_t}{V_{DD}-V_t} + ln \left(\frac{3V_{DD}-4V_t}{V_{DD}} \right) \right] \\ t_{PLH} = \frac{C}{\mu_p (W/L)_p (V_{DD}-V_t)} \left[\frac{2V_t}{V_{DD}-V_t} + ln \left(\frac{3V_{DD}-4V_t}{V_{DD}} \right) \right] \end{array} \tag{4.1}$$

where μ_n, μ_p are the electron mobility in silicon for the NMOS and PMOS respectively, C_{ox} is the capacitance per unit gate area, $(W/L)_n, (W/L)_p$ are the ratio between width and length in NMOS and PMOS respectively, V_{DD} is the supply voltage, V_t is the threshold voltage, and C is the load capacitance that the inverter drives. The frequency output of the RO is given by

$$f = \frac{1}{N\left(t_{PHL} + t_{PLH}\right)} \tag{4.2}$$

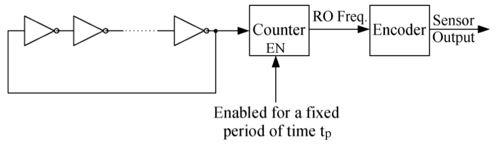
where N is an odd number of inverters in the RO chain. Due to process variation and environmental uncertainties, most of the parameters in Eq. 4.1 will be random variables, and it is reasonable to assume that they follow a Gaussian distribution [387]. In addition, μ_n,μ_p and V_t are temperature-sensitive parameters, and we assume they follow these empirical equations, respectively [3]:

$$\begin{array}{c} \mu_n = \mu_{no}(T/T_o)^{-1.5} \\ \mu_p = \mu_{po}(T/T_o)^{-1.5} \\ V_t = V_{t0} - 0.002(T - T_o) \end{array} \tag{4.3}$$

where μ_{n0}, μ_{p0} are the nominal values of mobility for NMOS and PMOS respectively, T_0 is the room temperature. Under the ideal circumstances where all sensor parameters are nominal and known, the relationship between sensor observation and its local temperature is deterministic and can be approximated by the following linear relationship [388]:

$$T = a_0 f + b_0. \tag{4.4}$$

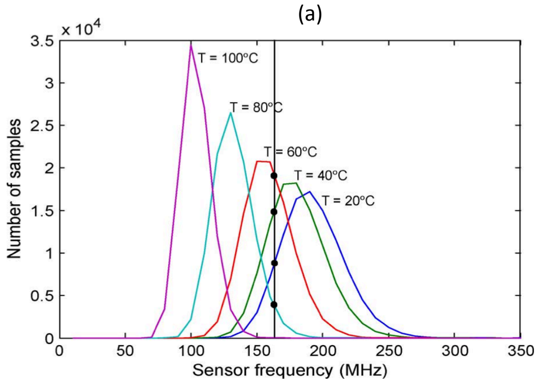
This one-to-one correspondence between temperature and sensor frequency f can be used to accurately calculate when a certain T is observed. Traditionally, single-point and two-point calibration methods are used, while for higher precision, a piece-wise linear calibration method or a nonlinear least square regression method based on multiple calibration points might be considered. In case of a single-point calibration, either the gain a_0 or the offset b_o is adjusted to generate correct temperature readings [348]. In other words, the frequency output at the single calibration point (temperature) is used as a reference point in the conversion from the frequency outputs of a sensor to the temperature readings. A two-point calibration method usually gives better results than a single-point method, because both the gain a_0 and the offset b_o are adjusted together based on the frequency outputs of a sensor at two calibration points. However, as most of the sensor parameters are random due to process variation, the coefficients a_0 and b_0 become random as well, making the deterministic approach inappropriate [388].



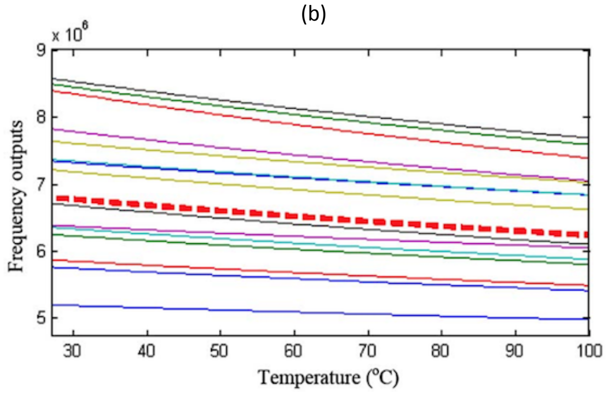
(a)

Parameters	W_n/W_p (nm)	L_n/L_p (nm)	T_{ox} (nm)	V_{DD} (V)	V_t (V)	μ_n/μ_p $(m^2/V{\cdot}s)$
Mean	270	180	4.1	3	0.45	0.034

(b)



(c)



(d)

Figure 4.5: Impact of process variation on thermal sensor characteristics (a) Ring oscillator (RO) based thermal sensor (b) nominal process parameters for the sensor (c) probability distribution of the sensor frequency output for the same temperature [388](d) variations in frequency outputs of sensors due to process variation and environmental uncertainties [327].

4.1.2.2 Aging Sensor-NoC

On-chip transistor and interconnect aging has become a primary limiting factor in the lifetime of emerging multicore SoCs. As chips age, key circuit and device parameters change, and these changes worsen with technology scaling. For example, BTI and HCI cause shifts in threshold voltage V_{th}, electromigration leads to increased interconnect resistance, and time-dependent dielectric breakdown (TDDB) can cause catastrophic transistor gate breakdown. NBTI alone can degrade circuit speed by upwards of 20% over a ten year period [366]. As ICs are often designed for 11 year life-spans [339], large design margins are required to ensure reliable operation over the circuit lifetime. While individual sensor design has been presented in several earlier work (refer Table 4.3), the use of these sensors in the system level has been lacking. In this work, the focus is at using different aging sensors and implement these sensors at the system level using a aging sensor NoC.

The aging sensor NoC consists of a sensor specific interface controller (similar to thermal sensors) that handles the control signals for the different modes of operations, and counts the number of event when polling, aggregates multiple sensor reading in a data frame, and communicates the sensed data to the collection point. A finite state machine could handle the required control signals. Aging being a slow phenomenon, speed of monitoring is not a primary concern. The aging sensor data can be acquired using a slow serial scan chain or a parallel bus interface to the collection point in the sNoC.

4.1.2.3 Power Sensor-NoC

The on-chip power sensor network is becoming extremely crucial in emerging multicore SoC, as the focus of multicore SoC design are shifting from traditional performance centric design to energy/power efficient design. Power constrained environments have completely changed the approach to SoC design. Since the highest performance implementations dissipate too much power, performance is becoming a secondary concern for most designers.

On-chip power sensing at different levels of granularity are going to be crucial for power and energy efficiency characterization as well as improving multicore SoC design through optimization. Traditionally, power sensors are present at the chip level where the capability to monitor the full system power is provided. However, as the significance of power awareness is increasing in the SoC design and runtime management of resources, more and more power sensors and monitors are directly included in the SoC at finer level of granularity such as monitoring at the cluster or core levels. As an example, the Samsung Exynos octa processor [266] implementing ARM's big.Little architecture [16, 163] only provides power monitoring for the Big or the small cluster, but not the individual cores. Measurement of the power of individual cores, major functional units such as on-chip cache memories, accelerators, and NoCs would be required for fine grained monitoring and eventual control using different power modes and control mechanism.

As emerging SoCs increasingly operate in different voltage and frequency domains and are moving towards the near threshold operation with large variation in power dissipation characteristics, power monitoring sensors and the network should take into account voltage scalability domains.

4.1.2.4 Critical-Path Delay Monitoring

The critical path monitor is a high-bandwidth sensor that provides timing margin accurate to within two inverter (FO2) delays for use in system level adaptation (for example by the voltage/frequency control loop) [102]. Critical path monitors provide a path-delay measurement architecture capable of providing real-time timing margin information. These monitoring structures do not ideally use a network-on-chip architecture, as the sampling requirement of these monitors are very high. As the time to take any mitigating action based on these monitors information is extremely small, they are directly connected locally to the control structure to avoid any latency incurred due to on-chip network transmission [102, 101]. Therefore, the design and implementation of these monitoring infrastructures is not discussed here and is out of scope for this work.

4.2 Heterogeneous Sensor Placement and Fusion

As discussed previously, emerging SoCs need to deploy self-monitoring of on-chip parametric variations and dynamic behaviors in order to deal with the multitude of challenges stemming from worsening process variations, increasing complexity, and highly dynamic on-chip environment. The efficiency and effectiveness of self-monitoring of several dynamic behaviors such as on-chip distribution of thermal, power, and reliable operation of emerging SoCs for full-chip characterization and subsequently for run-time behavior prediction are directly impacted by on-chip sensor characteristics and their placement. As the efficiency of the monitoring not only depends on number of sensors but also on the type of sensors distinguished by the sensor characteristics such as sensor accuracy, area, and power along with their careful placement, we present an efficient approach to effectively fuse the information from diverse type of sensors specifically with different accuracy, area, and power characteristics jointly with their placement.

To the best of our knowledge state-of-the-art approaches only considered homogeneous sensors of one type without leveraging the variability induced heterogeneities or design diversities among different sensor types. In this work, we exploit the flexibility and trade-off in accuracy and area/power characteristics of varied sensors types to perform a heterogeneous sensor placement and fusion (HSPF) to achieve efficient self-monitoring for full-chip characterization of dynamic behavior such as thermal profile characterization. Unlike traditional sensor allocation and placement techniques that consider only one sensor type, HSPF finds a combination of the heterogeneous sensors along with their placement for a given budget of sensor area and power such that the fusion of the information from the sensor combination produces minimal full-chip characterization error.

Experimental results with multicore SoC using Alpha processor show significant improvements compared to the state-of-the-art in terms of reconstruction accuracy for the same sensor area and power budget. In particular, our HSPF approach achieves superior accuracy (around 10-100$\times$ error reduction with three types of sensors in comparison to a single type without any additional overhead) and execution speedup of over 20$\times$ for full-chip monitoring over the state-of-the-art techniques. This work illustrates the HSPF method using thermal sensors of different types and is applicable for other types of sensors such as aging and power.

4.2.1 Motivation and Approach

Accurate on-chip temperature sensors have high area cost and are affected by significant systematic and random noise. Due to process variation and environmental

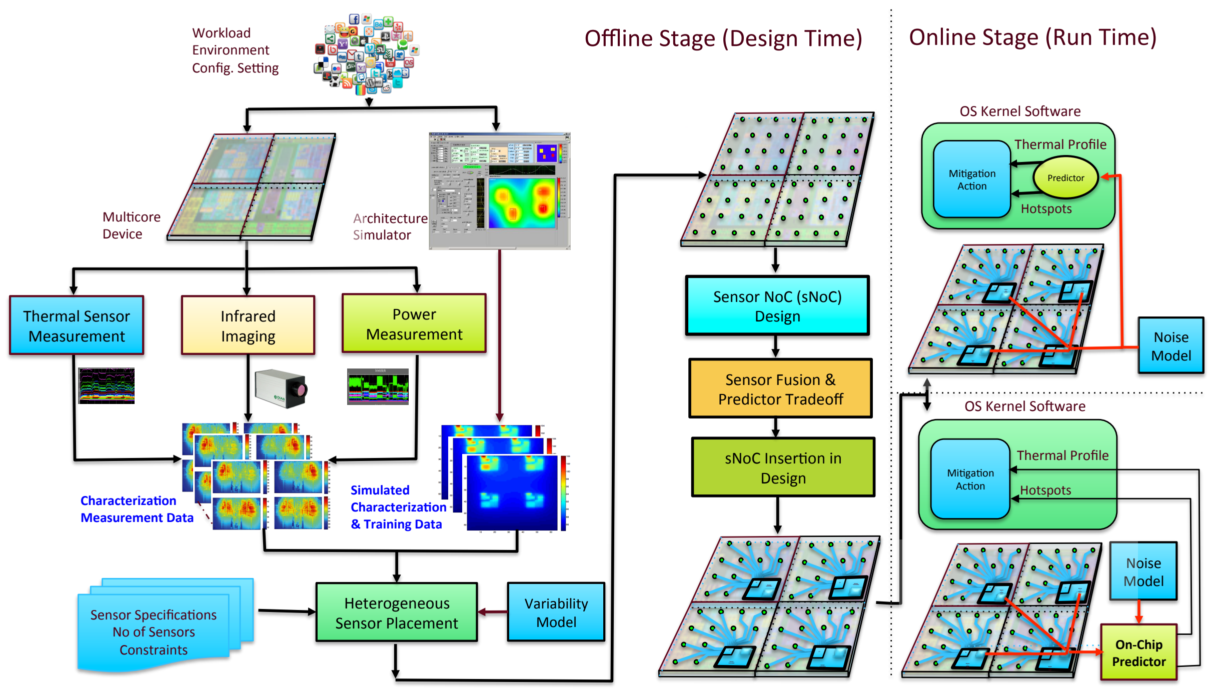


Figure 4.6: Heterogeneous Sensor Placement and Sensor NoC (sNoC) Design Approach.

uncertainties, the frequency outputs of ring oscillator (RO)-based thermal sensors will be different from sensor to sensor. Fig. 4.5(d) shows variations in the frequency outputs of a number of sensors with exactly identical design parameters. A reference frequency output, which is calculated from nominal parameter values, is also shown as a thick dashed line in the same figure for comparison. We can observe the frequency output of each sensor has a different gain (or slope) and a different offset. Therefore, each sensor behaves as a different sensor type even when they have same design parameters and the impact of the variations on the sensor accuracy can be as much as 71.5% in the worst case for blindly trusting a sensor to be noise-free [388].

In addition, to keep post-manufacturing testing costs low, not all the sensors are accurately calibrated. In other words, the frequency outputs of each sensor should be interpreted properly or considered with additional error in their reading. Specifically, the conversion from the frequency outputs of a sensor to temperature readings should be performed with great care in order to minimize possible temperature reading errors [348] as the actual frequency outputs of sensors at deeper technology nodes are going to be lot noisier [388]. Consequently, the effects of calibration error and noise have to be considered in the full chip monitoring.

As temperature sensors along with their peripheral circuits introduce non-negligible overhead in silicon area and power consumption, it is extremely important to minimize the design overhead associated with temperature sensors and their placement

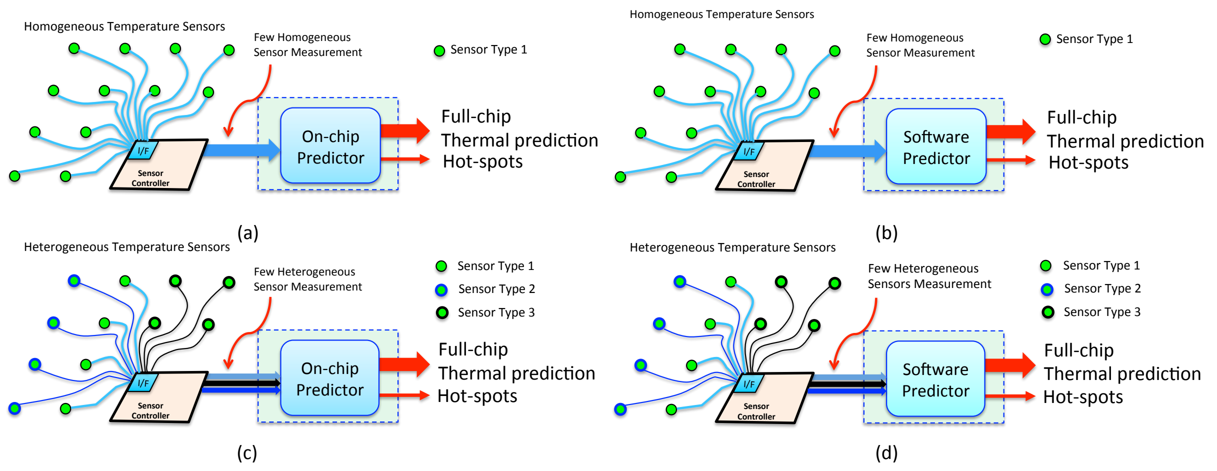

Homogeneous Temperature Sensors
Few Homogeneous Sensor Measurement
Sensor Type 1
On-chip Predictor
Full-chip Thermal prediction
Hot-spots
I/F
Sensor Controller
(a)
Software Predictor
(b)
Heterogeneous Temperature Sensors
Few Heterogeneous Sensor Measurement
Sensor Type 2
Sensor Type 3
(c)
Few Heterogeneous Sensors Measurement
(d)

Figure 4.7: Run-time configuration of thermal sensing networks for emerging SoC (a) homogeneous sensors with hardware predictor (b) homogeneous sensors with software predictor (c) heterogeneous sensors with hardware predictor (d) heterogeneous sensors with software predictor.

without surrendering the accuracy of thermal monitoring. On one hand, large number (few hundreds) of temperature sensors are needed for accurate thermal monitoring as core counts scales, on the other hand, they incur substantial die real-estate and power while facing large process variations. To effectively address this inherent trade-off in thermal monitoring, we propose heterogeneous sensor placement and fusion (HSPF) using heterogeneous mix of sensors while considering the inherent sensor variabilities. In this generalized and integrated approach, we exploit the flexibility and trade-off in area, power, and accuracy characteristics of varied thermal sensors to perform a improved sensor allocation and placement while precisely recovering the full-chip thermal profile of emerging multicore architectures. Unlike state-of-the-art sensor allocation and placement techniques that use a single type of homogeneous sensor, our proposed HSPF approach as depicted in Fig. 4.6 leverages the variability induced heterogeneities or design induced diversities among different sensor types such that full-chip thermal monitoring accuracy is improved. HSPF finds a combination or the mix of different sensors for given sensor area and power budget along with their placement such that the full-chip thermal characterization error is minimized. Contrary to the existing techniques that minimizes the number of sensors to reduce the area and power overhead, the proposed approach performs a joint design space exploration of sensor and on-chip sensor network while maximizing the number of meaningful samples used in the reconstruction without increasing the sensing overhead. HSPF differs from traditional approach by having a design-time offline stage as well as an online run-time stage that fuses the runtime measurement to be used by a configurable predictor (on-chip hardware or software) to provide improved fidelity and full-chip awareness of the thermal characteristics while drastically reducing the sensing and reconstruction error.

4.3 Full-chip Thermal Reconstruction: Problem Formulation

Let $t[i,j]$ be the two-dimensional thermal map where i and j represent the coordinates of the locations within the two-dimensional floor plan. The two-dimensional thermal map $t[i,j]$ can be represented using a one-dimensional vector $x[k]$ where $1 \leq k \leq N$ by stacking the columns such that $x[k] = t\left[k \, mod \, H, floor\left[\frac{k}{W}\right]\right]$. Thus, $\mathbf{x} = \{x_1, x_2, .., x_N\}$ represents the full temperature field at each grid locations $N = WH$ of the die. Let $\hat{\mathbf{x}}$ be the reconstructed signal of $\mathbf{x}$ in presence of noise and truncation errors and $\mathbf{x_S}$ be the measurement with total sensors $M \ll N$ at locations $\mathbf{L} = \{i_1, i_2, ..., i_M\}$ such that $\mathbf{x_S}$ =$\mathbf{x}(\mathbf{L})$. In presence of sensor noise and errors $\boldsymbol{\epsilon}$, the sensor measurement is:

$$\mathbf{x_S} = \mathbf{x}(\mathbf{L})+\boldsymbol{\epsilon} \tag{4.5}$$

where the error $\boldsymbol{\epsilon}$ is suitably represented by a probability distribution such as multivariate Gaussian distribution ($\boldsymbol{\epsilon} \sim \mathcal{N}_M(\boldsymbol{\mu},\boldsymbol{\Sigma})$) [388] with mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\Sigma}$. Let $\widehat{\mathbf{x}}_\mathbf{S}$ be an estimate of $\mathbf{x_S}$ which is the measurement from R heterogeneous sensor types with accuracy σ_r, power p_r, area a_r for sensor type r. Let $\mathbf{m} = \{m_1, m_2, ..., m_R\}$ represents combination of the heterogeneous sensors, i.e., m_r is the number of sensors of type r that is used in the measurement at locations $\mathbf{L_r}$ (representing the location of the sensors of type r) and $\mathbf{x_{S_r}}$ represents the measurement from sensor of type r located at $\mathbf{L_r}$ such that :

$$\mathbf{x_{S_r}} = \mathbf{x}(\mathbf{L_r}) + \boldsymbol{\epsilon_r} \tag{4.6}$$

where $\boldsymbol{\epsilon_r} \sim \mathcal{N}_{\mathbf{m_r}}(\boldsymbol{\mu_r}, \boldsymbol{\Sigma}_r)$ is Gaussian noise of dimension m_r, mean $\boldsymbol{\mu_r}$, and variance $\boldsymbol{\Sigma_r}$. If we represent the covariance of the sensor error $\boldsymbol{\epsilon_s}$ as $\boldsymbol{\Sigma} = \sigma^2\mathbf{V}$, where $\mathbf{V}$ is a matrix and σ is a positive bounding constant corresponding to maximum value in $\boldsymbol{\Sigma}$, there are two special cases (a) $\mathbf{V} = \mathbf{I}$ (b) $\mathbf{V}$ =$diag(V_1, V_2, ..., V_R)$. In the first case, where the variance is the same for each component (sensor), is called **homoscedasticity** and in the second where variance of each component is **not** the same is called **heteroscedasticity.** As we have different types of sensors with varied error characteristics, the sensor error $\boldsymbol{\epsilon_s}$ has heteroscedastic distribution. Thus, combined sensor measurements are heteroscedastic and is given by:

$$\mathbf{x_S} = \mathbf{x_{S_1}} \bigcup \mathbf{x_{S_2}} ... \bigcup \mathbf{x_{S_R}} = \left\{ \bigcup_{r=1}^{R} \mathbf{x_{S_r}} \right\} \tag{4.7}$$

in presence of the sensor noise $\boldsymbol{\epsilon_s} = \bigcup\limits_{\mathbf{r=1}}^{\mathbf{R}} \boldsymbol{\epsilon_r}$ at the locations :

$$\mathbf{L} = \mathbf{L_1} \bigcup \mathbf{L_2} .... \bigcup \mathbf{L_R} = \left\{ \bigcup_{\mathbf{r=1}}^{\mathbf{R}} \mathbf{L_r} \right\} \mid \mathbf{L_i} \bigcap \mathbf{L_j} = \emptyset, \forall \mathbf{i} \neq \mathbf{j}. \tag{4.8}$$

As the number of sensors of a particular type is equal to the size of the location vectors, i.e., $m_r = |\mathbf{L_r}|$, the total number of sensors is $M = \sum\limits_{r=1}^{R} m_r = |\mathbf{L}|$. Similarly, the total sensor area and power consumed by the sensors is given by:

$$\begin{array}{c} \sum_{r=1}^{R} a_r * m_r \leq AB \\ \sum_{r=1}^{R} p_r * m_r \leq PB \end{array} \tag{4.9}$$

where, AB and PB are the sensor area and power budget respectively.

4.3.1 Signal Estimation and Recovery

The sensor measurement signal in Eq. (4.5) can be represented using a generalized linear regression model:

$$\mathbf{x_S} = \boldsymbol{\Phi_s \alpha_s} + \boldsymbol{\epsilon_s} \tag{4.10}$$

where $\boldsymbol{\Phi_s}$ is the basis or kernel matrix, $\boldsymbol{\alpha_S}$ are the coefficients of the expansion over the basis $\boldsymbol{\Phi_s}$. With the assumption of zero mean and deterministic basis, following theorems are well known results in linear regression analysis and estimation theory [283].

Theorem 1. *Let* $\mathbf{x} = \boldsymbol{\Phi\alpha} + \boldsymbol{\epsilon}$ *be the linear regression model with zero mean homoscedastic noise distribution* ($E(\boldsymbol{\epsilon}) = \mathbf{0}$, $Cov(\boldsymbol{\epsilon}) = \boldsymbol{\sigma^2 I}$) *and deterministic basis* $\boldsymbol{\Phi}$ *such that* $rank(\boldsymbol{\Phi}^{n\times p}) = p$ *then* $rank(\boldsymbol{\Phi}'\boldsymbol{\Phi}) = p$ *and* $\left(\boldsymbol{\Phi}'\boldsymbol{\Phi}\right)^{-1}$ *exists. In that case, the ordinary least square (OLS) solution is unique and is given by:*

$$\widehat{\boldsymbol{\alpha}}_{OLS} = \boldsymbol{\Phi}^{\dagger}\mathbf{x} = (\boldsymbol{\Phi}'\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}'\mathbf{x} \tag{4.11}$$

with the orthogonal projection as $\widehat{\mathbf{x}} = \boldsymbol{\Phi}\widehat{\boldsymbol{\alpha}}_{OLS} = \boldsymbol{\Phi}(\boldsymbol{\Phi}'\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}'\mathbf{x} = \mathbf{P}\mathbf{x}$ *where* $\mathbf{P} = \boldsymbol{\Phi}(\boldsymbol{\Phi}'\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}'$ *is the projection matrix and* $\mathbf{P}\boldsymbol{\Phi} = \boldsymbol{\Phi}$. *The OLS estimate has the following properties:*

- $E(\widehat{\boldsymbol{\alpha}}_{OLS}) = \boldsymbol{\alpha}$,
- $Cov(\widehat{\boldsymbol{\alpha}}_{OLS}) = \sigma^2(\boldsymbol{\Phi}'\boldsymbol{\Phi})^{-1}$,
- $\widehat{\boldsymbol{\alpha}}_{OLS}$ is the best linear unbiased estimator (BLUE), i.e., the minimum variance unbiased estimator.

Theorem 2. *Let* $\mathbf{x} = \mathbf{\Phi}\boldsymbol{\alpha}+\boldsymbol{\epsilon}$ *be the generalized linear regression model where* $rank(\mathbf{\Phi}^{n\times p}) = p$*, with zero mean heteroscedastic noise distribution such that* $E(\boldsymbol{\epsilon})$=0,$Cov(\boldsymbol{\epsilon})$ =$\mathbf{\Sigma} = \sigma^2\mathbf{V}$*, with known* $\mathbf{V}$, *then there exists a transformation of* $\mathbf{x}$ *to a new response vector which has a covariance matrix* $\sigma^2\mathbf{I}$ *. Ordinarily least square applied to the transformed* $\mathbf{x}$ *yields a solution:*

$$\widehat{\boldsymbol{\alpha}}_{GLS} = (\mathbf{\Phi}'\mathbf{V}^{-1}\mathbf{\Phi})^{-1}\mathbf{\Phi}'\mathbf{V}^{-1}\mathbf{x} \tag{4.12}$$

with the orthogonal projection as

$$\widehat{\mathbf{x}} = \mathbf{\Phi}\widehat{\boldsymbol{\alpha}}_{GLS} = (\mathbf{\Phi}'\mathbf{V}^{-1}\mathbf{\Phi})^{-1}\mathbf{\Phi}'\mathbf{V}^{-1}\mathbf{x} = \widetilde{\mathbf{P}}\mathbf{x} \tag{4.13}$$

where $\widetilde{\mathbf{P}} = (\mathbf{\Phi}'\mathbf{V}^{-1}\mathbf{\Phi})^{-1}\mathbf{\Phi}'\mathbf{V}^{-1}$ *is the generalized projection matrix and* $\widetilde{\mathbf{P}}\mathbf{\Phi} = \mathbf{\Phi}$*. The generalized least square (GLS) estimate is the best linear unbiased estimator (BLUE) with the following properties:*

- $E(\widehat{\boldsymbol{\alpha}}_{GLS}) = \boldsymbol{\alpha}$,
- $E(\widehat{\boldsymbol{\alpha}}_{GLS}) = \boldsymbol{E}(\widehat{\boldsymbol{\alpha}}_{OLS})$,
- $Cov(\widehat{\boldsymbol{\alpha}}_{GLS}) = \sigma^2(\mathbf{\Phi}'\mathbf{V}^{-1}\mathbf{\Phi})^{-1}$,
- $Cov(\widehat{\boldsymbol{\alpha}}_{OLS}) = \sigma^2(\mathbf{\Phi}'\mathbf{\Phi})(\mathbf{\Phi}'\mathbf{V}\mathbf{\Phi})(\mathbf{\Phi}'\mathbf{\Phi})^{-1}$.

From Theorems (1) and (2) we can infer that OLS is the best linear unbiased estimator (BLUE) in presence of homoscedasticity and GLS is the BLUE in presence of heteroscedesticity. Under heteroscedesticity, OLS is still an unbiased linear estimator, but not the best estimator (i.e., not efficient). In other words, OLS is not the minimum variance estimate in the presence of heteroscedesticity. The usual variance of the OLS estimator is biased and thus inefficient. Consequently, we use the GLS estimate in Eq. (4.12) to find the coefficients from the sensor measurement as

$$\widehat{\boldsymbol{\alpha}}_s = (\mathbf{\Phi}_s'\mathbf{V}^{-1}\mathbf{\Phi}_s)^{-1}\mathbf{\Phi}_s'\mathbf{V}^{-1}\mathbf{x}_\mathbf{S} \tag{4.14}$$

where we call $\mathbf{\Phi}_s$ as the sensing matrix.

4.3.2 Accounting for Process Variability and Noise in Full Signal Prediction

As the number of sensors are very few compared to the number of grids / monitoring points in the die ($M \ll N$), the spatial thermal profile has to be recovered from the few measurements $\mathbf{x}_\mathbf{S}$. If we apply the generalized linear regression model for the full thermal map, we have $\mathbf{x} = \mathbf{\Phi}\boldsymbol{\alpha} + \boldsymbol{\varepsilon}_p$, where $\mathbf{\Phi}$ is a deterministic basis of size $N \times N$, and the coefficients $\boldsymbol{\alpha}$ are estimated as $\widehat{\boldsymbol{\alpha}}$ by GLS to get $\widehat{\mathbf{x}}$, an estimate of the thermal map due to the true process noise $\boldsymbol{\varepsilon}_p$. As thermal maps are often sparse, we can approximate the thermal map with a linear combination of K columns of $\mathbf{\Phi}$ and K elements of $\boldsymbol{\alpha}$ out of N such that $\widetilde{\mathbf{x}} = \mathbf{\Phi}_{\boldsymbol{K}}\boldsymbol{\alpha}_{\boldsymbol{K}}$. Let $\mathbf{K} = \{j_1, j_2, ..., j_K\}$ be the vector of the locations of the coefficients in $\boldsymbol{\alpha}$ such that $\boldsymbol{\alpha}_{\boldsymbol{K}} = \boldsymbol{\alpha}(\mathbf{K})$. Note that this approximation makes $\mathbf{\Phi}_{\boldsymbol{K}}$ of dimension $N \times K$ and $\boldsymbol{\alpha}_{\boldsymbol{K}}$ of dimensions $K \times 1$. This approximation or truncation introduces an additional error or noise term $\boldsymbol{\varepsilon_t}$ in the regression models such that $\mathbf{x} = \mathbf{\Phi}_{\boldsymbol{K}}\boldsymbol{\alpha}_{\boldsymbol{K}} + \boldsymbol{\varepsilon_p} + \boldsymbol{\varepsilon_t}$. When the number of sensors M is equal to (or greater than) the number of basis vectors or columns K , $\boldsymbol{\alpha}_{\boldsymbol{K}}$ can be represented by $\boldsymbol{\alpha_s}$ in addition to a sensor noise term $\boldsymbol{\varepsilon_s}$ in the thermal map such that $\mathbf{x} = \mathbf{\Phi}_{\boldsymbol{K}}\boldsymbol{\alpha_s} + \boldsymbol{\varepsilon_p} + \boldsymbol{\varepsilon_t} + \boldsymbol{\varepsilon_s}$. Representing the total noise as sum of all the noise components as $\boldsymbol{\varepsilon} = \boldsymbol{\varepsilon_p} + \boldsymbol{\varepsilon_t} + \boldsymbol{\varepsilon_s}$, we can make an estimate of the thermal map as:

$$\widehat{\mathbf{x}} = \mathbf{\Phi}_{\boldsymbol{K}}\widehat{\boldsymbol{\alpha}}_{\boldsymbol{s}} = \mathbf{\Phi}_{\boldsymbol{K}}\left[(\mathbf{\Phi}_{\boldsymbol{s}}'\mathbf{V}^{-1}\mathbf{\Phi}_{\boldsymbol{s}})^{-1}\mathbf{\Phi}_{\boldsymbol{s}}'\mathbf{V}^{-1}\right]\mathbf{x_S}. \tag{4.15}$$

Note that the sensing matrix $\mathbf{\Phi}_{\boldsymbol{s}}$ is formed from the basis matrix $\mathbf{\Phi}$ corresponding to the sensor locations $\mathbf{L}$ and coefficient locations $\mathbf{K}$ such that $\mathbf{\Phi}_{\boldsymbol{s}} = \mathbf{\Phi}(\boldsymbol{L}, \boldsymbol{K})$. The reconstruction matrix $\mathbf{\Phi}_{\boldsymbol{K}}$ is formed from K columns corresponding to the coefficient locations such that $\mathbf{\Phi}_{\boldsymbol{K}} = \mathbf{\Phi}(:, \boldsymbol{K})$ where the operator ':' represents all rows of $\mathbf{\Phi}$. For valid reconstruction in Eq. (4.15) , $\mathbf{V}^{-1}$ and $(\mathbf{\Phi}_{\boldsymbol{s}}'\mathbf{V}^{-1}\mathbf{\Phi}_{\boldsymbol{s}})^{-1}$ must exist and results in $M \geq K$ as a requirement.

To evaluate the reconstruction / recovery accuracy over a set of T thermal traces such that $\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, .., \mathbf{x_T}\}$ of size $T \times N$ with each row indicating a trace $\mathbf{x}$, we define the residual vector $\widehat{\boldsymbol{\varepsilon}}_{\boldsymbol{i}} = \mathbf{x_i} - \widehat{\mathbf{x}}_{\mathbf{i}}$ for the thermal trace $\mathbf{x_i}$ and the residual sum of square $RSS_i = \widehat{\boldsymbol{\varepsilon}}_{\boldsymbol{i}}'\widehat{\boldsymbol{\varepsilon}}_{\boldsymbol{i}} = \|\mathbf{x_i} - \widehat{\mathbf{x}}_{\mathbf{i}}\|^2$ such that the mean square error of the trace $\mathbf{x_i}$ is $MSE_i = E(\widehat{\boldsymbol{\varepsilon}}_{\boldsymbol{i}}'\widehat{\boldsymbol{\varepsilon}}_{\boldsymbol{i}}) = \frac{1}{N}\sum_{j=1}^{N}(x_i[j] - \widehat{x}_i[j])^2$. The total average mean square error over all the traces is:

$$MSE = \frac{1}{TN}\sum_{i=1}^{T}\sum_{j=1}^{N}(x_i[j] - \widehat{x}_i[j])^2. \tag{4.16}$$

4.3.3 HSPF Problem Statement

The HSPF problem is defined as an optimization problem which selects the best combination of heterogeneous sensors, their locations along with the basis vector combinations (basis selection) such that the full chip thermal reconstruction error is minimum subject to the constraints discussed in Section (4.3). The formal statement of the problem is as follows:

Given R heterogeneous sensor types with each sensor type r having power consumption p_r, and area a_r, accuracy σ_r, the problem is to select the combination of sensors $\mathbf{m} = \{m_1, m_2, ..., m_R\}$ (where m_r is the number of sensors of type r with the total number of sensor $M = \sum_{r=1}^{R} m_r$), their placement $\mathbf{L} = \{i_1, i_2, ..., i_M\}$, and the coefficient locations $\mathbf{K} = \{j_1, j_2, ..., j_K\}$ to select the basis vectors such that the full-chip reconstructed residual error is minimized subject to the constraints as in Eq. (4.17).

$$\begin{array}{cc} \underset{\mathbf{L}, \mathbf{m}, \mathbf{K}, \widehat{\boldsymbol{\alpha}}_{\boldsymbol{s}}}{Minimize} & \| \mathbf{x} - \boldsymbol{\Phi}_{\boldsymbol{K}} \widehat{\boldsymbol{\alpha}}_{\boldsymbol{s}} \|_2^2 \\ Subject\ to \quad : & \begin{array}{c} \| \widehat{\boldsymbol{\alpha}}_{\boldsymbol{s}} \|_0 \leq K \leq M = \sum_{r=1}^{R} m_r \\ \sum_{r=1}^{R} a_r * m_r \leq AB \\ \sum_{r=1}^{R} p_r * m_r \leq PB \\ \mathbf{L} = \bigcup_{r=1}^{R} L_r ; \mathbf{L_i} \bigcap \mathbf{L_j} = \emptyset, \forall i \neq j \end{array} \end{array} \tag{4.17}$$

4.4 Methodology and Solution

The HSPF problem is a generalization of the sparse regression problem [356] with additional constraints and variables. The sparse regression problem [356] is NP-hard and consequently the HSPF problem is NP-hard. With NP-hardness established, an optimal polynomial time algorithm is unreachable. Consequently, we propose heuristics and simplification strategies in developing a greedy solution for the HSPF problem. Specifically, we investigate a greedy approach that consists of three stages with the aim of decoupling the problem into subproblems for easy solution. We discuss the subproblems and some of the implementation issues in the following sections.

4.4.1 Stage-wise Greedy Solution (gHSPF)

The stage-wise greedy solution approach is motivated by the work by Ranieri et al. [282] and Reda el al. [285]. gHSPF attempts to decouple the HSPF into heterogeneous sensor selection as the first stage, the basis vector and coefficient location selection as second, and the sensor placement and reconstruction as the third stage. By using the observation in Section 4.3.1, we decouple the HSPF into two optimization problems such that solution to both directly improves the reconstruction accuracy. By doing so, we can easily deploy effective existing solutions at much reduced complexity. We discuss all the stages and the optimization algorithms in the following subsections.

4.4.2 ILP based Heterogeneous Sensor Selection

It has been shown in [356] that the probability of reconstructing a signal exponentially increases with increasing number of samples. If the number of samples is chosen as $M_S \geq cKln\left(\frac{N}{K}\right)$, it is possible to reconstruct every K sparse coefficients with a probability exceeding $1 - e^{-NM_S}$. As the number of samples M_S exponentially impacts the reconstruction accuracy, it is logical to maximize them while choosing them from good sensors. However, as the number of samples M_S is directly related to the number of sensors M as well as their area, power, and accuracy trade-offs, we define an optimization to maximize the number of samples as a weighted sum of the samples obtained from a sensor of particular type within the area and power budget. The weights w_r can be chosen based on accuracy to give preference to or penalize one type over the other in selecting the samples. We cast the problem as an integer linear program (ILP) by defining a vector $\mathbf{m} = [m_1, m_2, ..., m_R]$ such that $M_S = \sum_{r=1}^{R} w_r m_r = \mathbf{w}'\mathbf{m}$. The ILP thus is stated as follows:

$$\begin{aligned} \underset{\mathbf{m}}{Maximize} \quad & M_S = \textstyle\sum_{r=1}^{R} w_r m_r = \mathbf{w}'\mathbf{m} \\ subject\ to & \textstyle\sum_{r=1}^{R} p_r * m_r \leq PB \\ & \textstyle\sum_{r=1}^{R} a_r * m_r \leq AB \end{aligned} \tag{4.18}$$

The ILP in Eq. (4.18) can easily be solved using any standard solver. The solution of the ILP in Eq. (4.18) determines the sensor combinations and total sensor used in the sensor placement and reconstruction.

4.4.2.1 Selection of Weights in Sensor Selection ILP

The first stage of the HSPF performs the sensor selection based on the area, power and sensor accuracy. An optimization problem to maximize the number of samples used in the reconstruction is formulated as an integer linear programming (ILP). The intuition is based on the results derived in compressive sensing [98] and dimensionally reduction techniques, which states that the probability of reconstruction can be exponentially increased with increasing samples for a sparse signal. We establish a relationship between total samples used in the reconstruction with that of the sensor accuracy, area, and power overhead so as to maximize number of samples. The number of samples M_S is defined as a weighted sum of the samples collected from a particular type of sensor. The weights will decide which sensor type will have maximum contributions to the reconstruction samples. Thus by suitably selecting the weights we can either penalize a particular type of sensor or give preference to another. Selecting the weights can determine the number of samples collected by a particular sensor type for given area-power budget of the sensors. In selecting the sensors, one approach would be to give equal preference to each sensor type. In such a scenario, irrespective of the accuracy of the sensor, the number of samples are predominantly determined by the area and power budget. If the design is area constraint, i.e., the area budget AB is small, the sensors with the least area overhead would be selected. On the other hand, if the design is power constraint such that the sensor power budget PB is small, the solution to the ILP would be guided toward the sensors with least power budgets. As both the constraints are simultaneously achieved, the solution meets design corners in between the two extremes.

Furthermore, as the overall reconstruction accuracy is affected by the sensor accuracy effects, higher the accuracy of the sensors, better would be the overall reconstruction. To consider the sensor accuracy in the selection of the sensors, we make the weights proportional to the accuracy (or inversely proportional to the sensor noise variance). This way, we can account for the sensor accuracy in the sensor selection while fine tuning the results to a better solution.

4.4.3 Basis Vector and Coefficient Selection

The purpose of this step is to find the K best basis vectors in $\mathbf{\Phi}$ and their corresponding locations $\mathbf{K} = \{j_1, j_2, ..., j_K\}$ to form the orthonormal basis matrix $\mathbf{\Phi}_{\boldsymbol{K}}$ such that the approximation $\widetilde{\mathbf{x}} = \mathbf{\Phi}_{\boldsymbol{K}}\boldsymbol{\alpha}_{\boldsymbol{K}}$ is optimal. One approach is to use the greedy based orthogonal matching pursuit (OMP) [356] algorithm to find the best basis vectors. Another approach is based on the dimensionality reduction technique as in [282] which states that the approximation error in the reconstruction can be represented as the sum of the eigen values of $\mathbf{\Phi}$, corresponding eigen vectors of which are not included in $\mathbf{\Phi}_{\boldsymbol{K}}$. If we define a covariance matrix $\mathbf{C}_{\mathbf{x}}$ formed from the set of thermal traces $\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_T}\}$ with eigen values $\{\lambda_n\}_{n=1}^{N}$, then the orthonormal basis $\mathbf{\Phi}_{\boldsymbol{K}}$ that introduces the least error in the approximation is formed from the K eigen vectors of $\mathbf{C}_{\mathbf{x}}$ with the largest eigen values $\{\lambda_n\}_{n=1}^{K}$. The minimum approximation error $\xi = \mathbf{E}\left[|\mathbf{x} - \widetilde{\mathbf{x}}|^{\mathbf{2}}\right]$ can be represented as [282]:

$$\xi = \sum_{n=K}^{N} \lambda_n. \tag{4.19}$$

Note that as the number of basis vectors K increases, approximation error ξ decreases in Eq. (4.19). In other words, the reconstruction error can be reduced by forming the basis matrix $\mathbf{\Phi}_{\boldsymbol{K}}$ with the eigen vectors corresponding to the dominant eigen values of $\mathbf{C}_{\mathbf{x}}$, as well as by increasing $K \leq M$.

4.4.4 Greedy Sensor Placement and Allocation

In this stage, sensor allocation and placement is carried out using the total number of sensors computed in the previous stage. We allocate the best and most accurate sensors to the most crucial locations. The sensor placement algorithm provides the sensor location in order of their importance and we map the sensor types to these location according to their accuracy. The placement algorithm iteratively finds the sensor locations using a greedy approach [285] in polynomial time. The algorithm picks the location based on highest temperatures iteratively by computing the orthogonal components at available location and then picking the location with highest orthogonal norm. Once the locations are found, the thermal profile is reconstructed using Eq. (4.15). The algorithm is summarized in Alg. 4.1.

The sensor placement approach in Alg. 4.1 is inspired by the concept of volume sampling [81]. The algorithm maintains two set $\mathbf{L}$ and $\mathbf{S}$ where $\mathbf{L}$ is the set of chosen sensor locations and $\mathbf{S}$ is the set of available sensor locations. Initially, the chosen sensor locations set $\mathbf{L}$ is initialized to a null set, and the set $\mathbf{S}$ is the initialized to the set of all possible sensor locations. In the next step, the algorithm picks the location with the highest temperature based on the Euclidian norm. The locations $\mathbf{L}$ and $\mathbf{S}$ are updated accordingly. The algorithm then iteratively computes the orthogonal components of the column vectors at the available locations and finds the highest Euclidian norm of the orthogonal components. The location corresponding to the highest Euclidian norm of the orthogonal components is included to the set of sensor locations $\mathbf{L}$ and removed from $\mathbf{S}$. The process is iterated for M locations.

Once the sensor combination and total number of sensors are selected, it is imperative to look for the mapping of the sensor of various types to the specified locations. As some sensor locations contribute more information to the reconstruction process than the others, it is logical to place the higher accuracy sensors to more information rich locations. The sensor placement algorithm returns placement of the sensors in order of their importance. We rank the sensors according to their accuracy and map the most accurate sensors to the most dominant locations. Another opportunity (or concern) that arises because of varied operating range of different sensors, need to be addressed during the sensor mapping process. A reasonable approach is to divide the full processor die into regions of hotter to colder thermal zones. For example, the integer register units (IRU) region of most processors (specifically Alpha processor) exhibit higher temperature than that of the cache memories. Sensors with high dynamic ranges can be mapped to these elevated regions while the lower range sensors can be placed in the relatively colder blocks.

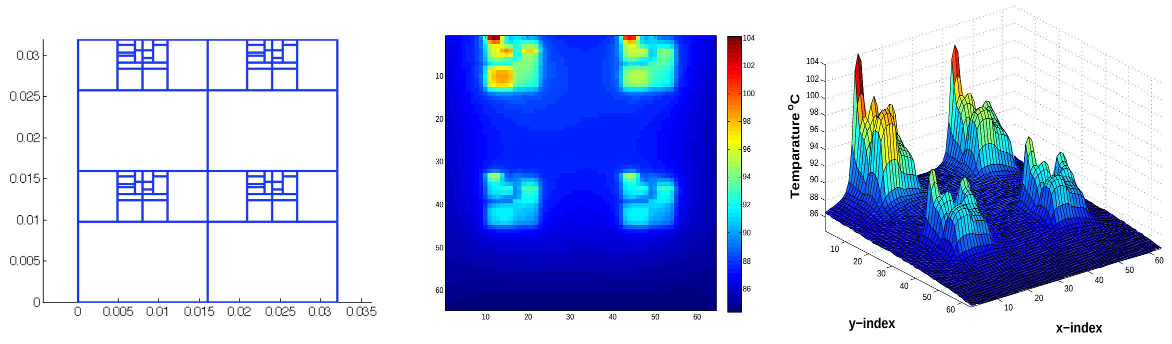
4.4.5 Heterogeneous Sensor Fusion and On-chip Prediction

The on-line prediction of the full thermal profile is performed considering the sensor heterogeneity in sensing accuracy and area-power characteristics either using a ordinary least square (OLS) or generalized least square (GLS) approach as described in Section 4.3.2. The algorithmic representation of the on-line prediction using heterogeneous sensor fusion is described in Alg. 4.2. This prediction stage is implemented as a run-time module in the OS such that the dynamic thermal management (DTM) can effectively exploit accurate thermal behavior of the chip and take appropriate step (for example, frequency thrilling, task migration, or DVFS) in order to avoid any thermal emergency in the chip.

4.5 Experimental Setup

We evaluate the effectiveness of our methodology, by setting up a tool chain that simulates the temperatures for single and multi-core architectures of up to 4 cores at 65nm technology node as shown in Fig. 4.8. We utilize HotSpot [147] for thermal simulation and McPAT [218, 219] to estimate the power for each block of the processor.

We use the Alpha 21264 processor as our baseline core. The Alpha 21264 is an out-of-order speculative execution core that is commonly used as a test-bench core in thermal management research [147, 227]. HotSpot takes in the floor-plan of the processor and the workload that will run on each core to produce the steady state and transient temperatures at each location of the grid. Using workload instruction traces, dynamic power traces for each micro-architectural unit are calculated and then fed together with the floor-plan into the thermal simulator to compute both the transient and the steady-state temperature. We use a total of 25 benchmarks from the SPEC 2000 benchmark suite to randomly allocate these workloads to a core in the multi-core architecture. The combination of these workloads assigned to different cores provide a rich set of thermal traces to characterize the thermal profile of the multi-core processor. We discretize the thermal profile by using a grid of $W = 64$ and $H = 64$ and used 25 benchmarks and their combinations to generate total of $T = 3194$ traces in our analysis.

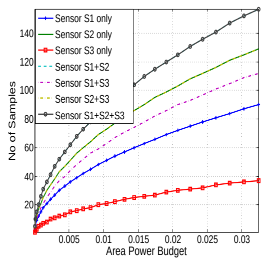


(a) (b) (c)

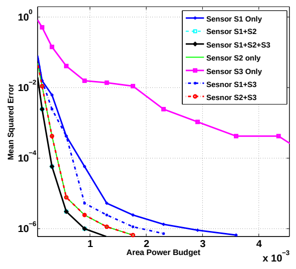
Figure 4.8: (a) Multicore Alpha processor floor-plan (b) original thermal profile with SPEC 2K benchmarks (c) Recovered thermal profile from noisy sensor measurements.

4.6 Experimental Results

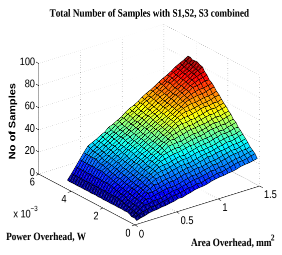
We present results for the set of thermal maps for single and multi-core architectures using the diverse set of heterogeneous temperature sensors as tabulated in Table 4.2. Although there is no limitation, we only consider three types of sensors $\{S_1, S_2, S_3\}$ from Table 4.2 in the following example. Fig. 4.9(a) shows the effect of sensor heterogeneity on the total number of samples that can be collected for given sensor power and area overhead. The x-axis represents the product of the area and power. Recall that, sensor S_1 is used for an area limited design and sensor S_2 for a power limited design. For a typical case of area and power budgets in between the two extremes of area or power limited design, neither S_1 nor S_2 is the optimal sensor type. In fact, to show the impact of choosing a particular sensor type, we plot the number of samples that can be collected for each type and their combination for a given overhead in Fig. 4.9(a). As more number of samples directly improves the measurement and reconstruction accuracy, we observe from Fig. 4.9(b) that sensor S_1 is better than S_3 and sensor S_2 is better than S_1 as well as S_3 for this area power design overhead. However, the combination of three sensor types $\{S_1, S_2, S_3\}$ accommodates more samples than any individual sensor type. Consequently, the heterogeneous combination of the sensors outperforms any individual sensor type in accommodating more sensors to improve the thermal monitoring process. Note that sensor S_2 would provide a better reconstruction compared to S_1 or S_3 if a single sensor were to be selected and hence HSPF can be used to select a suitable sensor type for a given area power overhead.



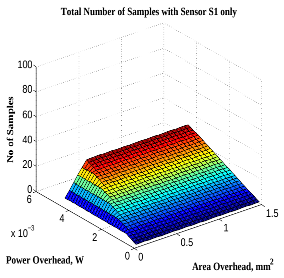
(a)



(b)



(c)



(d)

Figure 4.9: Design space exploration for area power trade-off (a) number of samples that can be used in reconstruction for given Area Power Budget for various sensor combinations (equal weights for all types) (b) MSE with different combination of sensors for given area power overhead. (c) Number of samples that can be collected for different design corners and trade-offs with mix of sensors type S1,S2, S3 (d) with S1 only.

The impact of increased number of samples and their combination for the given area-power overhead is reflected in the MSE and is shown in Fig. 4.9(b). A significant improvement in accuracy is observed for different combinations of the sensors for a given sensor area and power budget. The number of samples that can be collected for different design corners and trade-offs is shown in Fig. 4.9 (c) and Fig. 4.9(d). As seen from Fig. 4.9(c) the heterogeneous mix of sensors $\{S_1, S_2, S_3\}$ can collect more samples by accommodating more number of sensors than the best area-efficient sensor type S_1 at all area power design corners. Fig. 4.10(a) shows the saving in overhead for a given accuracy requirement as well as improvement in accuracy for a given overhead in comparison with two state-of-the-art techniques, namely k-LSE[251] and EigenMaps [282]. As they do not specify the type of the sensor to be used, we used the best area efficient sensor S_1 for the given overhead for these two methods, and the sensor combination $\{S_1, S_2, S_3\}$ for HSPF. It is clear from Fig. 4.10 that for a specified reconstruction accuracy, HSPF can save die area or power for the sensors by appropriately choosing the right mix of sensors. As observed from Fig. 4.10, HSPF provides accuracy improvement of around 10-100$\times$ for same overheads compared to the k-LSE and and EigenMaps. This improvement in turn can provide better detection of hotspots and worst case temperature gradient. Besides, the total execution time for HSPF is 358.9 sec and that of EigenMaps is 8099.7 sec, an improvement of $22.56\times$, when implemented in Matlab running on Intel i7, 2.4GHz machine.

4.6.1 Effect of Approximation and Sensor Accuracy on Prediction Accuracy

The effect of both approximation and sensor noise on the reconstruction mean square error is bounded by the condition number κ of $\mathbf{\Phi_s}$ and the noise energy of $\| \epsilon \|_2$ as [282]:

$$\frac{\| \hat{x} - x \|_2}{\| x \|_2} = \mathcal{O}(\kappa^2(\mathbf{\Phi_s})) \| \epsilon \|_2 . \tag{4.20}$$

If the condition number is close to one, the matrix is well conditioned which means its inverse can be computed with good accuracy. If the condition number is large, then the matrix is said to be ill-conditioned. Practically, such a matrix is almost singular, and the computation of its inverse, or solution of a linear system of equations is prone to large numerical errors. The numerical error that is introduced is often much larger than that what is introduced by ill-conditioning of $\mathbf{\Phi_s}$. The error ε_t introduced due to either truncation or inadvertent ill-conditioning in Section 4.3.2 is much higher than that of the contribution ε_s due to sensor accuracy. Bounding the sensor noise energy in Eq. 4.20 by the weakest sensor accuracy, we can bound the maximum error contribution due to sensor noise in MSE Eq. 4.20. In other words, the error contribution due to sensor inaccuracy is not as significant as basis truncation or ill-conditioning, under certain conditions. Thus, we can effectively gain in accuracy by slightly trading sensor accuracy by allowing many weak, inaccurate sensors but eventually gaining more by increasing the number of basis vectors in the reconstruction. More number of sensors result in inclusion of the dominant eigen vectors in the reconstruction and remove the repercussions significantly in lieu of moderate increase in error due to sensor inaccuracy. Exploiting this tradeoff is cardinal to the accuracy improvement in HSPF. This trade-off is also depicted in the eigen values of the basis matrix $\mathbf{\Phi}$ as shown in Fig. 4.10b where the magnitude is of the order of 10^7 within 50 eigen values. If we accommodate 10 accurate sensors in a given area-power budget, we will be able to use 10 eigen vectors as in Fig. 4.10c in the reconstruction and thus the approximation error introduced would be equal to the sum of rest of the eigen values – roughly > 10^1 as marked in Fig. 4.10b. On the other hand, if 41 relatively inaccurate sensors are used in the same area power budget, they will be able to cover 41 dominant eigen vectors and thus reduce the error to roughly > 10^{-2}. So, there is factor of 10^3 error reduction due to increasing the dominant basis vectors in the approximation even though the sensor inaccuracy has increase to say by an order of 10^1. Overall, there is a improvement of 10^2 in the reconstruction with heterogeneous mix of relatively inaccurate sensors.

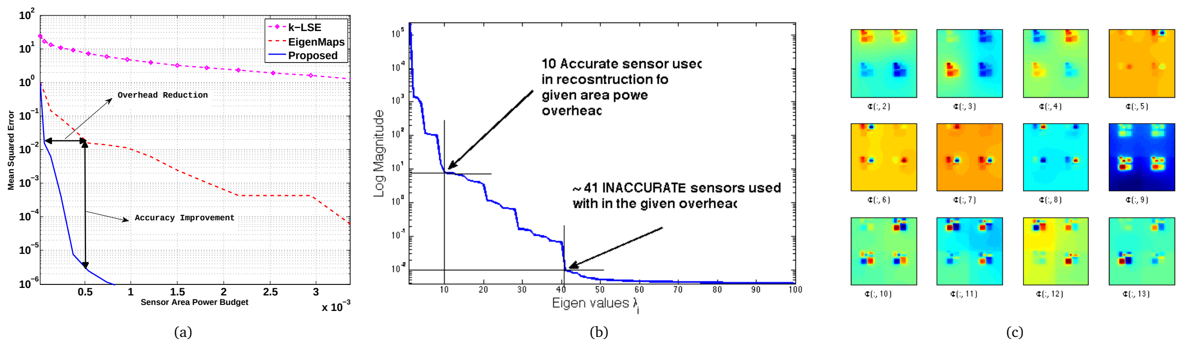


Figure 4.10: (a) Comparison of reconstruction error with state-of-the-art methods. Both k-LSE [251] and EigenMaps [282] use sensor type S1 while proposed HSPF uses sensor combination S1,S2, and S3. (b) Eigen values and their magnitudes (c) Eigen Vectors for first 12 dominant eigen values corresponding to the basis matrix $\mathbf{\Phi}$ formed from the covariance matrix $\boldsymbol{C_x}$ of the thermal traces.

4.6.2 Computational Complexity of HSPF

The computational complexity of choosing M sensors from N locations is the combinatorial C_M^N and that of choosing K coefficients (or basis vectors) among N possible combinations is C_K^N. The combined combinatorial complexity is C_M^N*C_K^N which is proportional to $(N!)^2$. This combinatorial complexity makes HSPF a very difficult problem to solve.

4.6.2.1 Predictor Computational Complexity

The predictor deployed for thermal sensor fusion and full chip prediction is dictated by the number of sensors, the dimension of the sensing matrix Φ_s and the computational complexity of the matrix inversion $(\Phi_s^{'}\mathbf{V}^{-1}\Phi_s)^{-1}$. If n_s is defined as the order of the matrix $\Phi_s^{'}\mathbf{V}^{-1}\Phi_s$, then the complexity of the predictor is bounded by $\mathcal{O}(n_s^3)$.

4.7 On-chip Self-Awareness Trends and Overhead in CPSoC

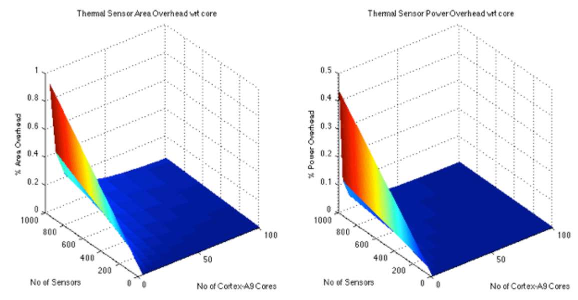
In this Section, we present the trend and the overhead of realizing on-chip self-awareness in CPSoC, by considering the different types of sensors, sNoC, architecture and topologies, as well as technology nodes.

4.7.1 On-chip Sensors Trends

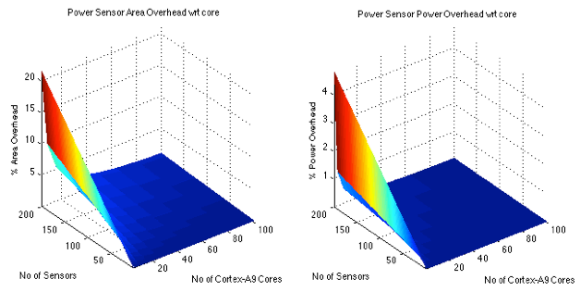
Table 4.3: On-chip Sensors Survey

Sensor	Tech. Node	Area	Power	Accuracy	Range	Resolution / Rate	Reference
Thermal	65nm	0.01 mm^2	150μW@1.0V	-5.1∼+3.4 ^{o}C	0∼60 ^{o}C	0.139 ^{o}C	[79]
	0.35μm	0.175 mm^2	0.49mW@3.3V	-0.7∼+0.9 ^{o}C	0∼100 ^{o}C	0.16^{o}C	[72]
	0.35μm	0.6 mm^2	36.7μW@3.3V	-0.25∼+0.35^{o}C	0∼90^{o}C	0.0918^{o}C	[71]
	0.7μm	4.5 mm^2	247μW@3.3V	$\pm$0.1^{o}C	-55∼125^{o}C	0.01^{o}C	[270]
	65 nm	0.1 mm^2	10μW@1.2V	$\pm$0.2^{o}C	-70∼125^{o}C	0.03^{o}C	[316]
	45nm		250-360 μW	$\pm$1^{o}C			[92]
Leakage	45nm/1.2V (SOI)	140x140 μm^2	120 uW	$\pm$10 %	27-100^{o}C/ load 0-5mA	10uA/0.5us	[44]
	90nm/1.2V	83x73 μm^2	0.66mW @80^{o}C	$\pm$3%			[186]
Power	45nm/1.2V (SOI)	140x140 μm^2	120 uW	$\pm$10 %	27-100^{o}C/ load 0-5mA	10uA/0.5us	[44]
	90nm/1.2V	83x73 μm^2	0.66mW @80^{o}C	$\pm$3%			[186]
	0.13 μm	0.01mm^2	180 μA@$1V$ 180μW		0-50mA/-23-100^{o}C	per 80ns	[45]
Aging (NBTI)	45nm/1.1V	148.0826*10^{-6} mm^2	18.57 μW (stress mode) 30.86 μW (meas. Mode)				[188]
	45nm	60*10^{-6} mm^2	12 μW				[257]
	45nm	78*10^{-6} mm^2	12.2 μW				[11]
	45nm	62*10^{-6} mm^2	15 μW				[62]
	65nm /1.2 V	38.04*10^{-6} mm^2					[173]
	130nm /1.2 V	33792*10^{-6} mm^2					[190]
	130nm/1.2V/3.3V	150*10^{-6} mm^2	469.5 μW				[176, 175]
	130nm	308*10^{-6} mm^2	500 nW (stress mode) 4.5 nW (meas. Mode)				[332]
Aging (TBDI)	65nm /1.2/2.5V	38040 *10^{-6} mm^2 214x551 μm^2				<1ps	[175]
	130nm/1.2 V	20x20 Array 555x225*10^{-6} mm^2					[177]
	130nm	150*10^{-6} mm^2	469.5 μW (stress mode) 14.03 μW (meas. Mode)			NA/100 us	[332]
Voltage	65nm	2700 μm^2	50 μW	85 mV		@5ns	[41]
	90nm		778 -338μW	50mV -10mV		@(1.85ns - 872ns)	[186]
	0.35 μm	24950μm^2	0.28 μW				[241]
Current	0.18um/1.8v	140x140 μm^2	1131.411mW			@100KHz	[365]
	0.7 um	170x170 μm^2					[343]
Delay	65nm	5inv+2mux		<1ps			[90]
	65/90nm	1 Inv					[46]
	180nm	2 Inv		<1ps			[34]
Critical Path	65m	90x36 μm^2		3.4-4.8 ps (std); 16 ps max			[101]
	65m	90x38 μm^2		20 mV/bit		12 bits / 1 sample@4-5Ghz	[102]

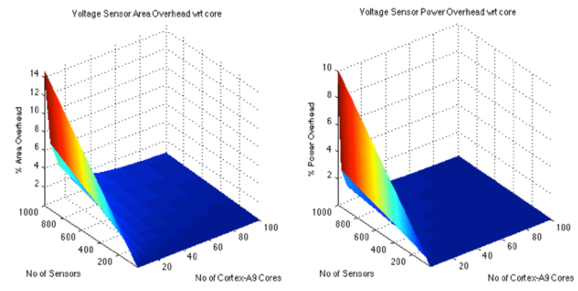
The specifications of different types of on-chip sensors are tabulated in Table 4.3. We show the overheads of using different types and number of sensors with respect to the platform cores. Based on the analysis, we show that the overhead of including the large number (for 1000 sensors of 5 types) is less than 7.3% of the area and 1.6% of the power with respect to 16 A9 platform cores. The power impacts of adding over thousands of sensors are shown in Fig. 4.12 and is found to be less than 0.3%. The thermal impact of adding the sensors are marginal as the power contribution to the overall architecture is less than 0.3% for a 16-core architecture.



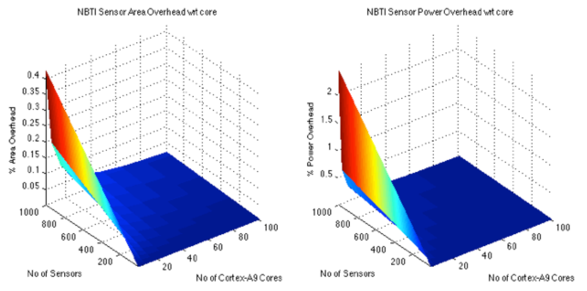
(a) Thermal sensors % overhead with respect to platform A9 cores.



(b) Power sensors % overhead with respect to platform A9 cores.



(c) Voltage sensors % overhead with respect to platform A9 cores.



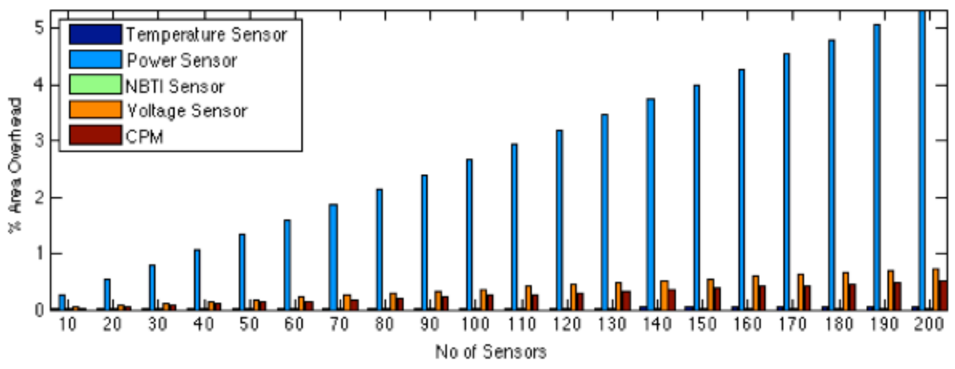
(d) NBTI sensors % overhead with respect to platform A9 cores.

Figure 4.11: Area-power overhead of on-chip sensors.

Note that since most of the sensors considered in the CPSoC platform can be implemented using the standard CMOS process, there are no special technology/manufacturing requirements. Mixed signal design can be used to include specialized sensors. However, virtualizing and fusing several sensors can avoid such specialized sensors need for CPSoC. For instance, although the power sensors (and few types of thermal sensors) use mixed signal design requiring an OpAmps, ADC etc., we overcome these limitations of process and custom design requirements by using virtual sensing. We use simple ring oscillators (RO) based delay sensors as a proxy for different sensors (e.g., temperature and power) and accurately estimates their values while saving substantial sensor area, power, design complexity and cost.

4.7.2 SensorNoC Implementation and Overheads

Since CPSoCs are sensor-actuator rich platforms, the information generated by these sensors need to be aggregated and processed at the collection point. CPSoC's introspective sentient units (ISU) serve these functions by collecting, monitoring, and processing



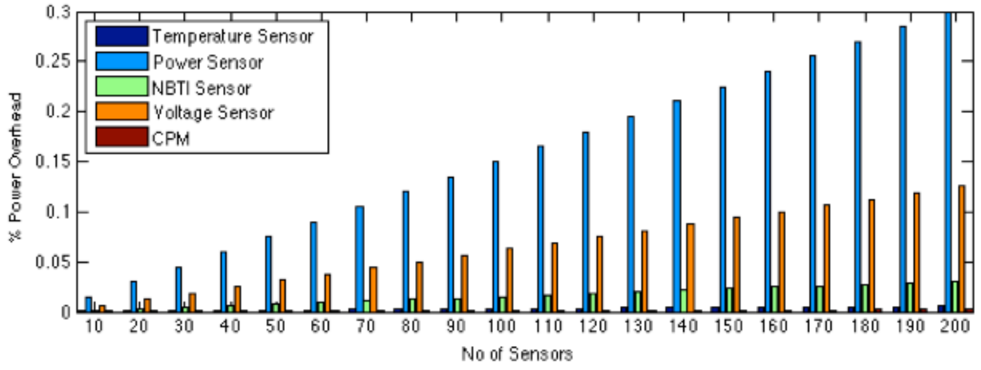


Figure 4.12: Area-power overhead of different sensors types with respect to Cortex-A9 core at 40nm technology node.

the sensing data and make meaning about the system's present states and context as well as future states. Each ISU is a processor based system that is interfaced to the sNoC where the virtual sensing approach is performed. The information from the ISUs are distributed to the computational cores. The placement and configuration of the ISUs can be determined based on the overheads and communication bottlenecks. A small core (e.g., Cortex M3) may be sufficient for low processing requirements. On the other hand, when aggressive processing is required to model and predict various phenomena in real-time, larger cores (e.g., A9) can be used. As an example, see the distribution of several ISUs across the fabric as shown in Fig. 3.11. The overheads of using different number of ISUs are shown in Fig. 4.13 where we observed that the area and power overhead is less than 1% for 8 Cortex-M3 cores with respect to the platform cores of 16xA9 cores.

The overall overhead of the CPSoC (considering 1000 sensors, consisting of 5 types, the sNoC and the ISU) is within 7.3% of the area and 1.6% in power budget of a 16 core platform made of Cortex A9. We expect that these overheads will only get smaller as

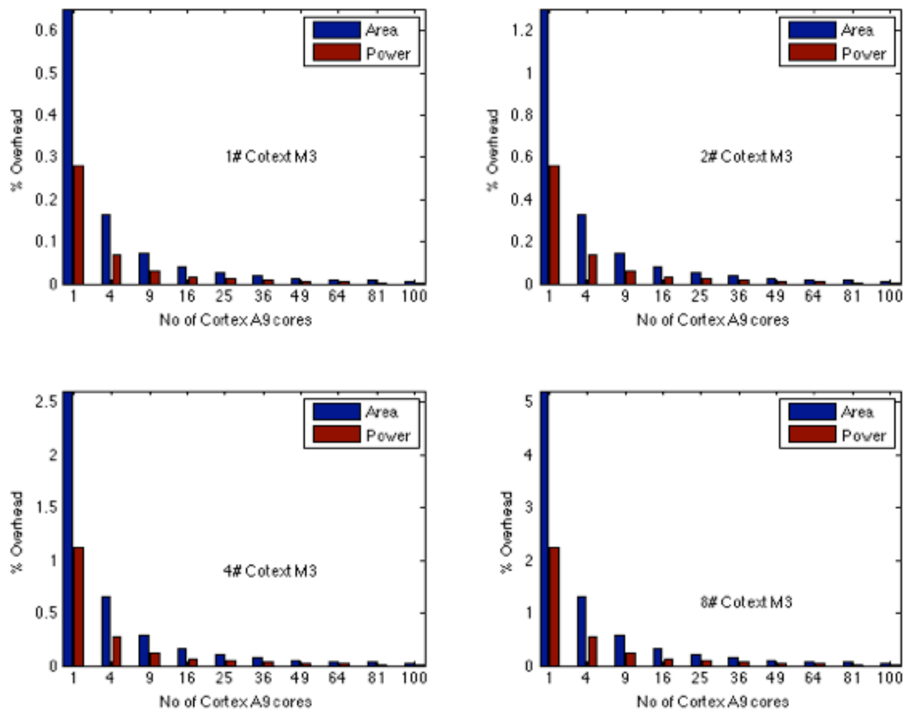


Figure 4.13: Area-power overhead of dedicated sensing data processing cores (Cortex-M3) with respect to the A9 cores.

we scale to larger platforms. Furthermore, we illustrate mechanisms to reduced these overheads using virtual sensing, hardware-software codesign, sensor repurposing and fusion in the subsequent sections.

4.7.2.1 sNoC Topology

A low-overhead, low-complexity dedicated approach for the collection and use of monitoring data for diagnosis, detection, debugging, and quality of service (power, performance, reliability, fault tolerance, security) is essential for resilient execution of applications on large multicore platforms. Sensor networks in emerging large scale embedded and computing systems are essential and to some extent are already existing (albeit in ad hoc manner) for reading critical system variables such as temperature in addition to the traditional network-on-chip [40]. Even though recent trends suggest that on-chip monitoring is a very promising foundation for investigating internal and environmental effects, building systems cost-effectively as well as in a predictable manner is a major engineering challenge. Specifically, on-chip sensor networks will need to consider a scalable topology, placement of network managers and sensors, sensing

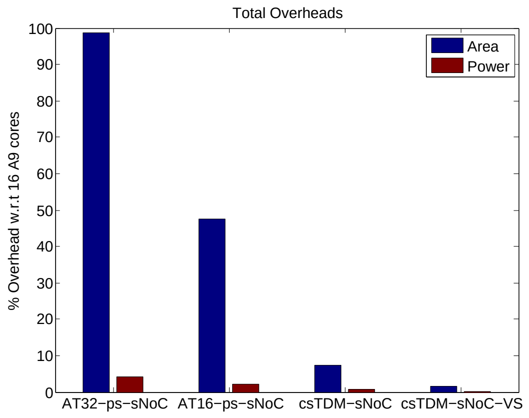


Figure 4.14: Comparison of total overheads of the introspection architecture in CPSoC (a) packet switched sNoC using Aggregation Tree of 32 nodes (b) packet switched sNoC using Aggregation Tree of 16 nodes (c) custom circuit switched TDM network (d) custom circuit switched TDM network with virtual sensing.

and network efficiency, colocation/coexistence with on-chip data networks and interacting interfaces for bandwidth, congestion, and area-power efficiency. The requirements from such a sensor network (see Fig. 3.11) will depend on sensing and actuation latency demands, overheads compared to simple design margining schemes, the number of sensors and sensor fusion scheme. These will dictate suitable topologies and signaling mechanisms. As the number of sensors are large, we consider a high fanout network such as aggregation tree (AT) as one of the promising topology as shown in Fig. 4.15a and Fig. 4.15b. The specification and resource usage of these two network topologies is tabulated in Table 4.4 and the overhead for 16 and 32 node sNoC in comparison to the platform cores (16 A9 cores) is shown in Fig. 4.16. This overhead when compared with 16 x A9 cores is relatively higher. As the number of sensors expected in these platform is going to be higher, we explore other network architectures that are relatively more resource efficient in the subsequent sections.

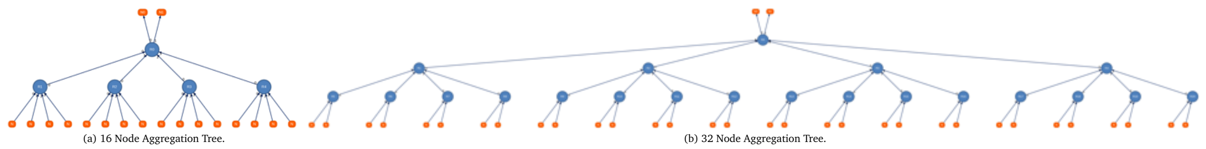


Figure 4.15: Aggregation Tree based sNoC topology.

4.7.2.2 Custom Circuit Switched sNoC

Many of the on-chip sensors are low bandwidth sensors and thus the sensor data can be time division multiplex over a router. However, as packet switching introduces

Table 4.4: Resource Utilization and Overheads of Aggregation Tree based sNoC.

Config. Name	Configuration						Area	Power	% Overhead w.r.t. 16 A9 core	
	No Tx Ch.	No Rx Ch.	Tree Fan Out	No VC	Flit Width	Flow Control			Area	Power
AT-16a	16	2	4	8	64	Credit	302273.1200 μm^2	1.5803e+03μW	0.7205	0.0316
AT-16b	16	4	4	8	64	Credit	310075.653901μm^2	1.7172e+03 μW	0.7391	0.0343
AT-32	32	2	4	8	64	Credit	1274994.786999 μm^2	6.0992e+03 μW	3.0392	0.1218

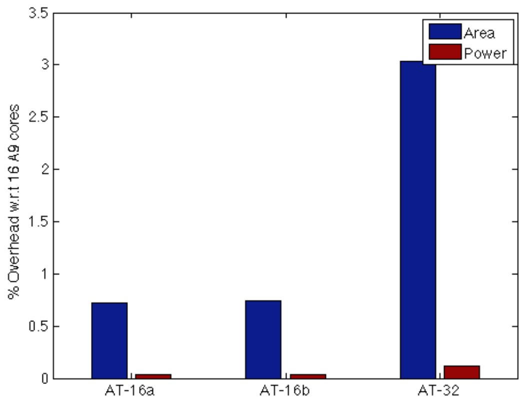


Figure 4.16: Aggregation Tree (AT) based sNoC overheads with respect to cortex A9 cores of the platform.

considerable overhead, we consider a simplified custom circuit switched architecture as shown in Fig. 4.17 for the sNoC. We consider two candidate topologies (ring and star) for the custom based on the cycle latency. If the cycle latency is comparable to the number of sensors, ring topology may be sufficient. On the other hand, if a low latency is desired the star network as shown in Fig. 4.17 is better candidate.

We implemented a circuit switched ring topology sNoC as shown in Fig. 3.11 in for 16 and 32 channels at each node/ router. The channels are time division multiplexed (TDM) for low bandwidth (slow varying phenomena) sensors such as aging. The comparison of the area and power with respect to the packet switched aggregation tree based sNoC is tabulated in Table 4.5. It is observed that the custom circuit switched sNoC can save area and power by two orders of magnitude.

Table 4.5: Circuit Switched TDM sNoC Area Power Estimation

SL No	Tech Node	Channel No	Channel Width	Area	Power	% Saving w.r.t. respective packet switched sNoC	
						Area	Power
1	32nm	16	8	2365.287712 μm^2	33.5115 μW	127.7955×	47.1569×
2	32nm	32	8	4.75e3 μm^2	68.01 μW	268.4200×	89.6809×

We also show that the overhead of the networks (packet switched and circuit switched) for the sNoC. For an aggregation tree based network of 32 nodes, the overhead is less

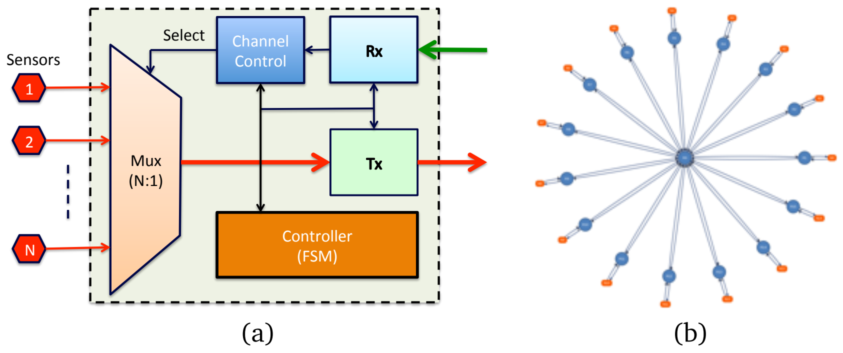


Figure 4.17: (a) Custom TDM Router for sNoC (b)Star topology for circuit switched sNoC. The channels are time division multiplexed (TDM).

than 3% with respect to 16xA9 platform cores. This is drastically reduced (268x in area and 89x in power) when a custom circuit switched network is used.

4.7.3 Thermal Sensor Overhead Estimation and Reduction

Full-chip runtime monitoring of thermal phenomena requires a large number of sensors that cost area as well as power. In order to further reduce the overhead incurred in the sensor design, design space exploration of the sensor front-end core, calibration lookup table (LUT), and readout logic is required. As an example, we consider the design of the sensor at two technology nodes for different sensor configurations. We estimated (as shown in Table 4.6) the area and power consumption of each component of the sensors. We observed that the LUT used for the sensors are the most area and power consuming part of the sensing system. For a large number of thermal sensors, the resource consumed by the LUTs are considerable compared to the sensing heads. By implementing the LUT in software instead of hardware, combined area and power overhead of the sensors as shown in Fig. 4.18 can be reduced considerably (almost over $\sim 10\times$). This also provides the flexibility to modify the LUT considering the variability of each die, during the calibration of each sensor.

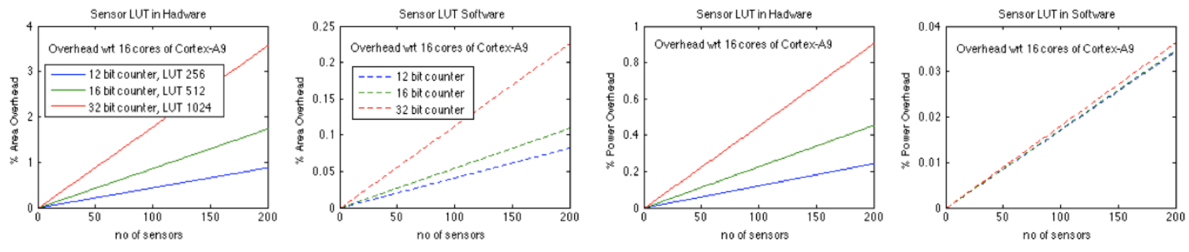


Figure 4.18: Virtualizing the calibration lookup table in software reduces the sensing overhead.

Table 4.6: Thermal sensor design space exploration.

Thermal Sensors Configuration & DSE							
SL No	Tech. Node	Sensor front end core		Sensor Calibration LUT		Total	
		Area	Power	Area	Power	Area	Power
1	32 nm	34.576623e-6 μm^2 (12 bit counter)	1.7253 μW	335.751567e-6 μm^2 (LUT256)	10.4937 μW	3.7033e-04 μm^2	12.2190 μW
2	32 nm	45.923567e-6 μm^2 (16 bit counter)	1.7409 μW	683.102252e-6 μm^2 (LUT512)	21.0713 μW	7.2903e- 04 μm^2	22.8122 μW
3	32 nm	94.848312e-6 μm^2 (32 bit counter)	1.8239 μW	1392.077308e-6 μm^2 (LUT1024)	43.5564 μW	0.0015 μm^2	45.3803 μW
4	45 nm	6.0661e-05 μm^2 (12 bit counter)	2.7561 μW	0.00058904 μm^2 (LUT256)	16.763 μW	0.0006497 μm^2	19.5190 μW
5	45 nm	8.0568e-05 μm^2 (16 bit counter)	2.7810 μW	0.0011984 μm^2 (LUT512)	33.660 μW	0.001279 μm^2	36.4410 μW
6	45 nm	0.0001664 μm^2 (32 bit counter)	2.9136 μW	0.0024422 μm^2 (LUT1024)	69.579 μW	0.0026316 μm^2	72.4920 μW

4.8 Related Work

Full chip thermal monitoring and reconstruction suffers from an inherent trade-off between accuracy of monitoring and implementation overhead (in terms of sensor die area and power) incurred in the monitoring infrastructure. Both the accuracy of the reconstruction and the overhead incurred in the monitoring infrastructure are directly impacted by the number of sensors, their locations, and accuracy [282]. Specifically, full chip thermal reconstruction accuracy predominantly depends on the number of on-chip sensors [251, 282], their placement [251, 282], as well as sensor accuracy and error characteristics [388]. Although improvement in reconstruction accuracy was studied by several computational means like interpolation [233, 245, 285], correlation of the sensor measurements, and transformed domain processing (e.g., FFT [83], DCT [251]), a tacit assumption in all these works [169, 233, 245, 251, 83, 282] is that sensors characteristics are homogeneous which is no longer valid for nanometer technologies in presence of variability [388, 54]. These works [169, 233, 245, 251, 83, 282] have neither considered the sensor error characteristics nor did they exploit the trade-offs in sensor characteristics to improve reconstruction accuracy. In our work, we exploit these trade-off characteristics among different sensor types with regard to the reconstruction accuracy and show that by using mix of many weak and inaccurate sensors along with few robust and accurate ones, the full-chip reconstruction accuracy can be improved by an order of magnitude.

The sensor placement work is close to the recent work of Ranieri et al. [282] which addresses the problem of sensor allocation and placement for a given set of homogeneous sensors and architectures. Ranieri et al. [282] do not consider the effect of variability induced heterogeneity or the sensor accuracy directly in the thermal profile reconstruction. The method in [282] does not specifically account for the sensor area and power overheads. While [282] improves the reconstruction accuracy by exploiting the correlation among the thermal maps, our work jointly considers the sensor accuracy, area, and power overhead and exploits the trade-off in different sensor types along with the correlation information among the thermal maps. Our work differs from [282] in several ways. First, our approach is more generic and can be applied for both homogeneous and heterogeneous sensor types where as the approach in [282] is only applicable for homogeneous sensors. As the thermal sensor accuracy can vary substantially due to process variations at different location of the die [388], the methods in [282] fail to account these effects. Our HSPF approach can easily be applied in such scenarios. Second, the reconstruction approach in [282] is based on ordinary least square (OLS), whereas our reconstruction approach is based on generalized least square (GLS) which is more accurate and efficient for general and heteroscedastic noise distribution. Third, our greedy sensor placement algorithm is relatively faster than that of [282] as we directly find the best M locations one after the other instead of rejecting $N-M$ (where $M \ll N$) futile locations. Fourth, we provide efficient solutions that can provide improvement of the order of $10\times-100\times$ in the reconstruction accuracy for the same sensing area and power overhead and execution speedup of over $20\times$.

4.9 Summary

This chapter presented the design, implementation, and architecture of distributed on-chip sensor networks (sNoC) for different kind of on-chip sensors that assist in achieving self-monitoring and awareness in emerging SoC. The architecture, design, and implementation of such networks for full chip characterization of dynamic phenomena are challenging as monitoring and reconstruction suffers from an inherent trade-off between accuracy of monitoring and implementation overhead (in terms of sensor die area and power). Both the accuracy of the reconstruction and the overhead incurred in the monitoring infrastructure are directly impacted by the number of samples, sensor locations, and accuracy of the sensors. To effectively address this inherent trade-off in thermal monitoring, we proposed a new method called heterogeneous sensor placement and fusion (HSPF) that exploits the flexibility and trade-off in area and power characteristics of varied thermal sensors to perform a heterogeneous sensor fusion and placement for precisely recovering the full-chip thermal map. Unlike the state-of-the-art sensor allocation and placement techniques that use a single type of homogeneous sensor, our HSPF approach finds the best combination or the mix of heterogeneous sensors for given sensor area and power budget along with their placement such that the full-chip thermal characterization error is minimized. Instead of quantifying the overheads in terms of total number of sensors, HSPF directly considers the sensor accuracy, area, power overhead of each sensor type in its formulation and thus provides a more flexible and accurate approach. HSPF exploits the trade-off in area and power of several sensor types to directly improve the accuracy of the full-chip thermal reconstruction. Simulations and experimental results show an improvement of 10-100$\times$ in reconstruction accuracy with three sensor types and execution speedup of over $20\times$ in comparison to a state-of-the-art technique using the most area efficient sensor type. On the other hand, when large number of sensors are included, the overall overhead of the CPSoC considering 1000 sensors (each of 5 types), the sNoC and the ISU is within 7.3150% of the area and 0.6476% of the power budget of an equivalent 16-core ARM Cortex A9 platform. Using virtual sensing of power the overall overhead is further reduced to 1.6834% of the area and 0.2529% of the power budget of the 16-core platform.

Algorithm 4.1 Greedy Heterogeneous Sensor Placement (GHSP)

Input: Thermal characterization traces $\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_T}\}$, Sensor types R, Sensor Specifications, $\mathbf{V}$, Sensor Area Budget AB, Sensor Power Budget PB
Output: Total no of sensors M, Sensor combination $\mathbf{m}$, Sensor location $\mathbf{L}$, Coefficient Location $\mathbf{K}$,Sensing matrix $\mathbf{\Phi_s}$ and the Basis Matrix $\mathbf{\Phi}$

```
1. Solve the heterogeneous sensor selection ILP in Eq. (4.18) and compute $M$ for the given
   sensor area power budget
2. Estimate $\mathbf{C_x} = \mathbf{E}[\mathbf{x}[\mathbf{i}]\mathbf{x}[\mathbf{j}]]$ from the set of thermal traces $\mathbf{X}$. $\mathbf{X}$ is normalized to reflect a
   zero mean process.
3. Compute the eigen values of $\mathbf{C_x}$ and the corresponding eigenvectors to form the basis $\mathbf{\Phi}$
4. Construct $\mathbf{\Phi_K}$ from $\mathbf{\Phi}$ corresponding to the largest eigen values given by coefficient loca-
   tion $\mathbf{K}$ such that $\mathbf{\Phi_K} = \mathbf{\Phi}(:, \mathbf{K})$
5. Greedy Sensor Allocation & Placement
   (a) Let $\mathbf{L} = \emptyset$ and $\mathbf{S} = \{1, .., N\}$
   (b) $s_1 = \arg\max_{s_1 \in S} \| \mathbf{X_{\{s\}}} \|_\mathbf{2}$
   (c) Let $\mathbf{L} = \mathbf{L} \cup \{s_1\}$ and $\mathbf{S} = \mathbf{S} - \{s_1\}$
   (d) For $k = 2..M$ do
         i. Project $\mathbf{X_S}$ into the column space of $\mathbf{X_L}$ : $\overline{\mathbf{P}} = \mathbf{X_L}\mathbf{X_L^\dagger}\mathbf{X_S}$
        ii. Find the orthogonal components $\mathbf{N} = \mathbf{X_S} - \overline{\mathbf{P}}$
       iii. Let $s_i = \arg\max_{s_i \in S} \| \mathbf{N_{\{s\}}} \|_\mathbf{2}$
        iv. Let $\mathbf{L} = \mathbf{L} \cup \{s_i\}$ and $\mathbf{S} = \mathbf{S} - \{s_i\}$
       end
```

Algorithm 4.2 Heterogeneous Sensor Fusion and On-line Prediction

Input: Total no of sensors M, Sensor location $\mathbf{L}$, Sensor types R, Sensor combination $\mathbf{m}$, Sensor Accuracy (Variance) $\mathbf{V}$, Basis $\mathbf{\Phi}$ and Coefficient Location $\mathbf{K}$, Sensor Measurement $\mathbf{x_s}$
Output: Sensing (Prediction coefficient) matrix $\mathbf{\Phi}_{\boldsymbol{s}}$, Predicted Thermal Profile $\hat{\mathbf{x}}$

```
1. Solve the heterogeneous sensor selection ILP in (4.18) and compute $M$ for the given sensor area power budget
2. Estimate $\mathbf{C_x} = \mathbf{E}\left[\mathbf{x}[\mathbf{i}]\mathbf{x}[\mathbf{j}]\right]$ from the set of thermal traces $\mathbf{X}$. $\mathbf{X}$ is normalized to reflect a zero mean process.
3. Compute the eigen values of $\mathbf{C_x}$ and the corresponding eigenvectors to form the basis $\mathbf{\Phi}$
4. Construct $\mathbf{\Phi}_{\boldsymbol{K}}$ from $\mathbf{\Phi}$ corresponding to the largest eigen values given by coefficient location $\mathbf{K}$ such that $\mathbf{\Phi}_{\boldsymbol{K}} = \mathbf{\Phi}(:, \mathbf{K})$
5. Construct the Sensing Matrix $\mathbf{\Phi}_{\boldsymbol{s}} = \mathbf{\Phi}(\boldsymbol{L}, \boldsymbol{K})$ and sensor variance scaling matrix $\mathbf{V}$
6. Reconstruct the thermal map $\widehat{\mathbf{x}} = \mathbf{\Phi}_{\boldsymbol{K}}\left[(\mathbf{\Phi}_{\boldsymbol{s}}'\mathbf{V}^{-1}\mathbf{\Phi}_{\boldsymbol{s}})^{-1}\mathbf{\Phi}_{\boldsymbol{s}}'\mathbf{V}^{-1}\right]\mathbf{x_S}$
```

Chapter 5

Cross-Layer Predictive Model Building

Predictive modeling is the process of developing a mathematical model to estimate and predict future behavior of a system by analyzing historical and current data [199]. The predictive modeling approaches can broadly be grouped into statistical and machine learning approaches. Statistical predictive model includes regression analysis and time series analysis. Regression model [297] describes the relationship between a response variable, and one or more predictor variables. Depending on the specific problem, regression techniques can be sub-categorized as linear regression (multiple, step-wise, multivariate regression models etc.), generalized linear model (logistic regression, multinomial regression, Poisson regression), and non-linear regression. The above mentioned regression models need not have time information; the data used for model building need not be in a time sequence, and can be picked randomly. Thus, data and model are oblivious of time. On the other hand, time series model [329] is one in which a series of observations are made over a certain time interval. The main idea behind time series analysis is to use a certain number of previous sequential observations to predict future observations. Commonly used forms of these models are parametric auto-regressive (AR), auto-regressive and moving average (ARMA), auto-regressive moving-average model with exogenous inputs (ARMAX), and auto-regressive models with integrated moving average (ARIMA).

Predictive models can be parametric and non-parametric [326]. A parametric model can be parameterized by a finite number of parameters whereas a non-parametric model cannot be parameterized by a fixed number of parameters (number and nature of the parameters are flexible). A parametric model is easier to build and understand, as it requires less data to estimate and predict future behavior of a system. In contrast, a non-parametric model requires a lot more data, and has a higher model complexity. At the same time, parametric model is computationally faster than a non-parametric model. Linear regression, polynomial regression, logistic regression etc. are the examples of parametric models and K-nearest neighbor, Gaussian processes, Dirichlet process mixtures etc. are the examples of non-parametric models. In this chapter, we focus on parametric predictive models, as they are easier to build and implement.

In this chapter, we present linear regression based predictive modeling approach for power and performance prediction. We also present time series predictive modeling approach including ARMAX model and system identification approach using state-space models, which are used subsequently for different applications such as design space exploration (DSE) of emerging Heterogeneous Multicore Processors (HMPs). The chapter is organized in the following sections. Section 5.1 describes the model building approach followed by cross-layer predictive modeling approach defined in Section 5.2 where we describe properties and complexity of the model building approach. Algorithms used for estimation and prediction are described in the subsequent sub-sections. Simulation and experimental results are described in Section 5.3.1. Section 5.5 provides a brief overview of the related works.

5.1 Model Building Approach

In this section we describe the method of model building using regression, time-series, and system identification approaches. We highlight the compelling use of these models in DSE of HMPs and motivate the use of such models in run-time decision making.

5.1.1 Linear Regression Based Predictive Model

Regression analysis [199, 297] is used to predict the value of a dependent variable based on the value of one or more independent variables and to explain the impact of changes in an independent variable on the dependent variable. Assuming a data set $\{y_l, x_{l1}, \ldots, x_{lp}\}_{l=1}^{n}$ of n observations is known including the values of p predictor variables and the values of the corresponding responses. The linear regression approach builds a linear relationship between the dependent or response variable y_l and the p-vector of predictor variables x_l based on the n observations. This relationship is modeled through a disturbance term or error variable ε_l which is an unobserved random variable that adds noise to the linear relationship between the dependent variable and predictor variables. Given the response $y_i (i \in [1..n])$, the corresponding predictor variables $x_{lk} (l \in [1..n], k \in [1..p])$, and the variable coefficients $\beta_k (k \in [1..p], \beta_k \geq 0)$ the regression model can be expressed as:

$$\begin{gathered} y_l = \beta_0 + \beta_1 x_{l1} + \cdots + \beta_p x_{lp} + \varepsilon_l, \qquad l = 1, \ldots, n, \\ y_l = \beta_0 + \textstyle\sum_{k=1}^{p} \beta_k x_{l,k} + \varepsilon_l, \quad l = 1, \ldots, n,\ k = 1, ..., p \\ y_l = \mathrm{x}_l \boldsymbol{\beta} + \varepsilon_l, \end{gathered} \tag{5.1}$$

where $x_{l1}, x_{l2}, \ldots, x_{lp}$, are the predictor variables or independent variables, β_0 is the intercept which implies the response when all the variables are set to zero, $\mathrm{x}_l \boldsymbol{\beta}$ is the inner product between predictor vector $\mathbf{x}_l = [\ 1 \quad x_{l1} \quad x_{l2} \quad \ldots \quad x_{lp}\ ]$ and parameter vector $\boldsymbol{\beta} = [\ \beta_0 \quad \beta_1 \quad \beta_2 \quad \ldots \quad \beta\ ]^{\mathrm{T}}$ where T denotes the transpose. The regression model for n observations including p predictor variables can be represented in matrix form:

$$\boldsymbol{y} = \mathbf{X} \boldsymbol{\beta} + \boldsymbol{\varepsilon}, \tag{5.2}$$

where $\boldsymbol{y} = [y_1, y_2, ..., y_n]^{\mathrm{T}}$, design matrix$\mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \ldots & x_{1p} \\ 1 & x_{21} & x_{22} & \ldots & x_{2p} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{n1} & x_{n2} & \ldots & x_{np} \end{bmatrix}$, $\boldsymbol{\varepsilon} = [\varepsilon_1, \varepsilon_2, ..., \varepsilon_n]^{\mathrm{T}}$.

The Eq. 5.2 is solved by least square method that minimizes the error $\boldsymbol{\varepsilon}$ to obtain the estimated value of the coefficient $\hat{\boldsymbol{\beta}}$:

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^{\mathrm{T}} \boldsymbol{X})^{-1} \boldsymbol{X}^{\mathrm{T}} \boldsymbol{y}. \tag{5.3}$$

The estimated linear regression model is given by:

$$\hat{\boldsymbol{y}} = \mathbf{X}\hat{\boldsymbol{\beta}}. \tag{5.4}$$

5.1.2 Time Series Predictive Model

A time series [329] is a sequence of data points measured over a continuous time interval using equal spacing between every two consecutive measurements. Time series forecasting is used to predict future values based on previously observed values.

Time series regression [329, 315] is a statistical method for predicting a future behavior of dynamic systems from experimental or observational data. A time series analysis is often described by multiple linear regression (MLR) models of the form: $y_t = X_t\beta + e_t$, to get an estimate of a linear relationship of the observed response y_t to the design matrix X_t which includes columns for contemporaneous values (existing or occurring in the same period of time) of current and past observations of predictors ordered by time t. The partial regression coefficients in β represent the marginal contributions of individual predictors to the variation in y_t when all of the other predictors are held fixed. The term e_t represents the residual which is defined as the differences between predicted and observed values of y_t. These differences are due to changes in β, measurement errors (changes in X_t), and model misspecifications (for example, omitted predictors or nonlinear relationships between X_t and y_t). It is usually assumed that e_t is generated by an unobservable innovations process with stationary covariance ,$C_T = Cov(\{e_1, ..., e_T\})$, for any time interval of length T. Under some basic assumptions about X_t, e_t, and their relationship, reliable estimates of β are obtained by ordinary least squares (OLS). A linear time-series model can be a polynomial or state-space model. Some particular types of models are parametric auto-regressive (AR), auto-regressive and moving average (ARMA), auto-regressive moving-average model with exogenous inputs (ARMAX), and auto-regressive models with integrated moving average (ARIMA).

5.1.2.1 Auto-regressive and moving average (ARMA) model

In the statistical analysis of time series, auto-regressive moving-average (ARMA) model [129] is a tool for understanding and predicting future values in this series in terms of two polynomials, one for the auto-regression and the second for the moving average. The ARMA model for a single-output time-series with input, $u(t)$, the output $y(t)$, and the noise $e(t)$ is expressed as $A(q)y(t) = C(q)e(t)$, where $A(q)$ represents the auto-regressive term, and $C(q)$ represents the moving average term, q is the time-shift operator.

Auto-regressive moving-average model with exogenous inputs (ARMAX model) [129] is expressed as:

$$y(t) + a_1y(t\text{-}1) + \ldots + a_{n_a}y(t - n_a) = b_1u(t\text{-}n_k) + \ldots + b_{n_b}u(t - n_k - n_b + 1)$$
$$+c_1e(t\text{-}1) + \ldots + c_{n_c}e(t\text{-}n_c) + e(t), \tag{5.5}$$

where $y(t)$ represents system output at time t, n_a represents number of poles, n_b represents number of zeroes plus 1, n_c represents number of C coefficients, n_k represents system delay, $y(t-1), \ldots, y(t-n_a)$ represent previous outputs on which the current output depends, $u(t-n_k), \ldots, u(t-n_k-n+1)$ represent previous and delayed inputs on which the current output depends, $e(t-1), \ldots, e(t-n_c)$ represents white-noise disturbance value. A more compact way to define the difference equation is:

$$A(q)y(t) = B(q)u(t\text{-}n_k) + C(q)e(t). \tag{5.6}$$

The variables $A(q)$, $B(q)$, $C(q)$, are polynomials with respect to the time-shift operatorq^{-1} and defined by the following equations:

$$A(q) = 1 + a_1q^{-1} + \ldots + a_{n_a}q^{-n_a}, \tag{5.7}$$

$$B(q) = b_1 + b_2q^{-1} + \ldots + b_{n_b}q^{-n_b+1}, \tag{5.8}$$

$$C(q) = 1 + c_1q^{-1} + \ldots + c_{n_c}q^{-n_c}, \tag{5.9}$$

where n_a is the order of $A(q)$, n_b is the order of $B(q)$ plus 1, n_c is the order of $C(q)$. If data is a time series, which has no input channels and one output channel, then ARMAX calculates an ARMA model for the time series $A(q)y(t) = e(t)$. ARMAX is useful when the load disturbances enter at the input.

5.1.3 System Identification for Predictive Model Building

System identification [1, 118] is a methodology for building mathematical models of dynamic systems using measurements of the system's input and output signals. In a dynamic system, the values of the output signals depend on both the instantaneous values of its input signals and also on the past behavior of the system. System identification uses the input and output signals measured from a system to estimate the values of adjustable parameters in a given model structure. System Identification methodology builds models using time-domain input-output signals, frequency response data, time series signals, and time-series spectra. Time-domain data consists of the input and output variables of the system recorded at a uniform sampling interval over a period of time, while frequency domain data represents measurements of the system input and output variables recorded or stored in the frequency domain. System Identification process requires a model structure, which is a mathematical relationship between input and output variables that contains unknown parameters and apply the estimation methods to determine the numerical values of the model parameters. Examples of model structures are transfer functions with adjustable poles and zeros, state space equations with unknown system matrices etc. The model structure can be represented as a set of equations or state-space system and the values of its parameters can be estimated from data. The System Identification process estimates model parameters by minimizing the error between the model output and the measured response. The output y_{model} of the linear model is given by: $y_{model}(t) = Gu(t)$, where G is the transfer function. To determine G, the difference between the model output $y_{model}(t)$ and the measured output $y_{meas}(t)$ is to be minimized. The minimization criterion is a weighted norm of the simulation error or prediction error, $v(t)$, where: $v(t) = y_{meas}(t)$-$y_{model}(t)$.

5.1.3.1 State-space models

State-space representation [253, 127] is a mathematical model that uses set of input, output and state variables to describe a system by a set of first-order differential or difference equations. The state of the system can be represented as a vector within that space. The most general state-space representation of a linear continuous time-variant system with p inputs, q outputs and n state variables is written as follows:

$$\begin{aligned} \dot{\mathbf{x}}(t) &= A(t)\mathbf{x}(t) + B(t)\mathbf{u}(t), \\ \mathbf{y}(t) &= C(t)\mathbf{x}(t) + D(t)\mathbf{u}(t), \end{aligned} \tag{5.10}$$

where $\mathbf{x}(\cdot)$ is the state vector, $\mathbf{x}(t) \in \mathbb{R}^n$; $\mathbf{y}(\cdot)$ is the output vector, $\mathbf{y}(t) \in \mathbb{R}^q$; $\mathbf{u}(\cdot)$ is the input or control vector, $\mathbf{u}(t) \in \mathbb{R}^p$; $A(\cdot)$ is the state or system matrix, $dim[A(\cdot)] = n \times n$; $B(\cdot)$ is the input matrix, $dim[B(\cdot)] = n \times p$; $C(\cdot)$ is the output matrix, $dim[C(\cdot)] = q \times n$; $D(\cdot)$ is the feedthrough or feedforward matrix (in cases where the system model does not have a direct feedthrough, $D(\cdot)$ is the zero matrix), $dim[D(\cdot)] = q \times p$. The state-space model representation for different system types are shown in Table 5.1.

In Chapter 6, we show temporal prediction of the thermal hotspots and power consumption of multicore systems ahead of time using a state-space model considering the discrete-time equivalent of a linear time-invariant system and use the predictions to make proactive decision in emerging SoCs.

5.2 Cross-Layer Predictive Modeling for Emerging SoCs

In this section, we describe an approach to build predictive models for emerging SoC architectures considering multiple layers of the system stack. The predictive modeling techniques described in Section 5.1 are generic and can be applied in many scenarios. However, as selection of the informative and independent features is a crucial

Table 5.1: State-space model representation for different system types.

System type	**State-space mode**
Continuous time-invariant	$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + B\mathbf{u}(t)$ $\mathbf{y}(t) = C\mathbf{x}(t) + D\mathbf{u}(t)$
Continuous time-variant	$\dot{\mathbf{x}}(t) = \mathbf{A}(t)\mathbf{x}(t) + \mathbf{B}(t)\mathbf{u}(t)$ $\mathbf{y}(t) = \mathbf{C}(t)\mathbf{x}(t) + \mathbf{D}(t)\mathbf{u}(t)$
Explicit discrete time-invariant	$\mathbf{x}(k+1) = A\mathbf{x}(k) + B\mathbf{u}(k)$ $\mathbf{y}(k) = C\mathbf{x}(k) + D\mathbf{u}(k)$
Explicit discrete time-variant	$\mathbf{x}(k+1) = \mathbf{A}(k)\mathbf{x}(k) + \mathbf{B}(k)\mathbf{u}(k)$ $\mathbf{y}(k) = \mathbf{C}(k)\mathbf{x}(k) + \mathbf{D}(k)\mathbf{u}(k)$
Laplace domain of continuous time-invariant	$s\mathbf{X}(s) = A\mathbf{X}(s) + B\mathbf{U}(s)$ $\mathbf{Y}(s) = C\mathbf{X}(s) + D\mathbf{U}(s)$
Z-domain of discrete time-invariant	$z\mathbf{X}(z) = A\mathbf{X}(z) + B\mathbf{U}(z)$ $\mathbf{Y}(z) = C\mathbf{X}(z) + D\mathbf{U}(z)$

step for effective predictive modeling, capturing the domain specific knowledge of the emerging SoCs is fundamental to the model accuracy and usefulness. While several literatures exist on predictive model of performance and power consumption of homogeneous architectures [331, 208, 157, 119], however, to the best of our understanding, cross-layer predictive modeling approach for emerging heterogeneous multicore systems has not been proposed. Design space exploration of HMPs is much more complex and challenging than that of homogeneous architectures since it involves re-evaluating architecture and application options along with the operating systems (OS) implications [78]. A straight forward extension of the above mentioned works is not directly applicable for emerging HMPs as they are targeted towards homogeneous multicore processors without considering the operating system. The exploration is performed using architectural and program space parameters without considering the OS scheduling by either using micro-architecture or cycle-accurate simulation. In order to consider the operating system in the cross-layer predictive modeling approach, full-system cycle-accurate architectural simulators are indispensable tools for evaluating complicated and subtle design trade-offs with respect to large design spaces, and handling various design constraints. In this thesis, we consider an extended full system cycle accurate HMP simulator [313] based on Gem5 [49] along with the Linux OS as the infrastructure for our cross-layer predictive model building. We build a predictive model of the full system considering parameters from all the layers of system stack in order to capture the HMP performance and power characteristics. Unlike the state-of-the-art approaches [167, 208, 378, 155, 209, 86, 268, 74, 75] that focused either on unicore or homogeneous multicores, our predictive approach specifically focusses on the cross-layer predictive model of the heterogeneous multicore processor while considering key parameters of the application, architecture, and the operating system in the evaluation of the approach.

In this section, we describe a cross-layer predictive modeling methodology for emerging SoC architectures. The approach is divided in two phases. In the training phase, known data (or training set) are used to identify the predictive model configuration as depicted in Fig. 5.1. We use special benchmarks for coverage of the design space. On the other hand, during the prediction phase, a predictive model is used to forecast the unknown system response as illustrated in Fig. 5.2. The training phase of the cross-layer predictive modeling approach captures the architectural design spaces and behaviors at different levels of abstraction to achieve efficiency (e.g., reducing simulation time by trimming down the large design space into a small finite set of points) and accuracy (gradual refinement of the abstraction models). This approach, illustrated in Fig. 5.1, is motivated by the platform based approach [182, 271] with the difference that the hardware architecture platform and the mapping strategy are varied along with diverse spectrum of applications for given system level constraints. We use the modeling and optimization techniques proposed in [268] to iteratively update the predictive models (as shown in Fig. 5.1) of different parts of the system stack as discussed in the subsequent sections.

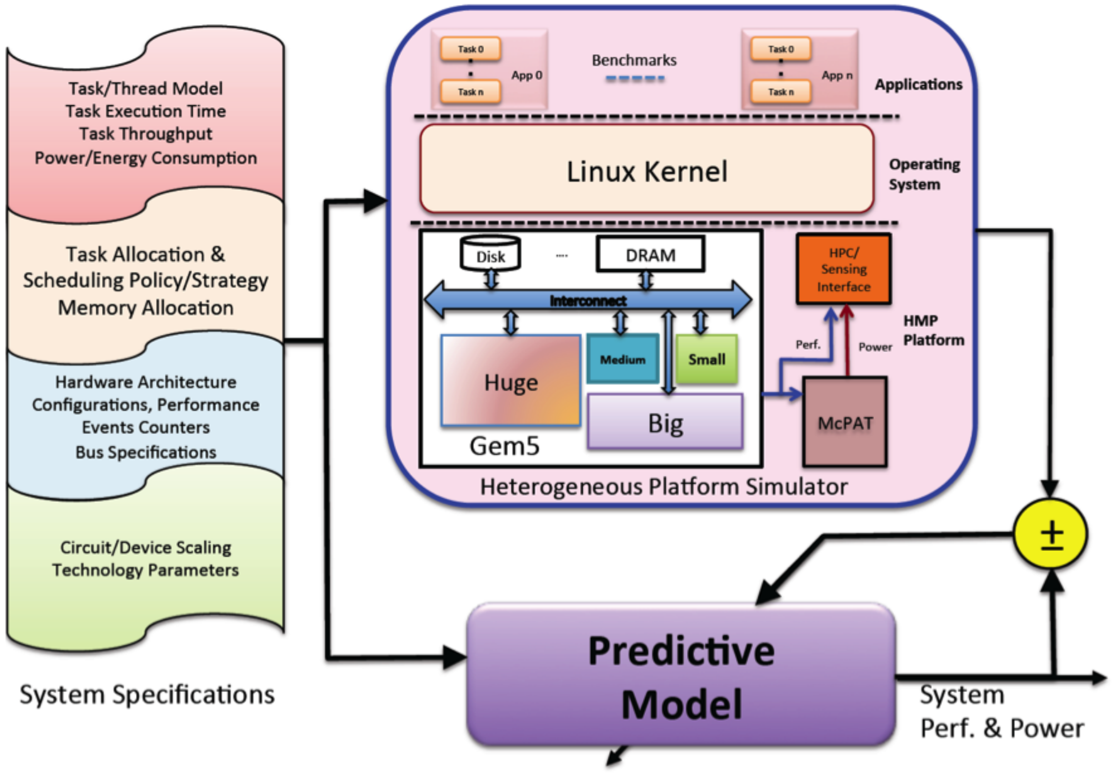


Figure 5.1: Training Phase : Cross-Layer Predictive Model Building.

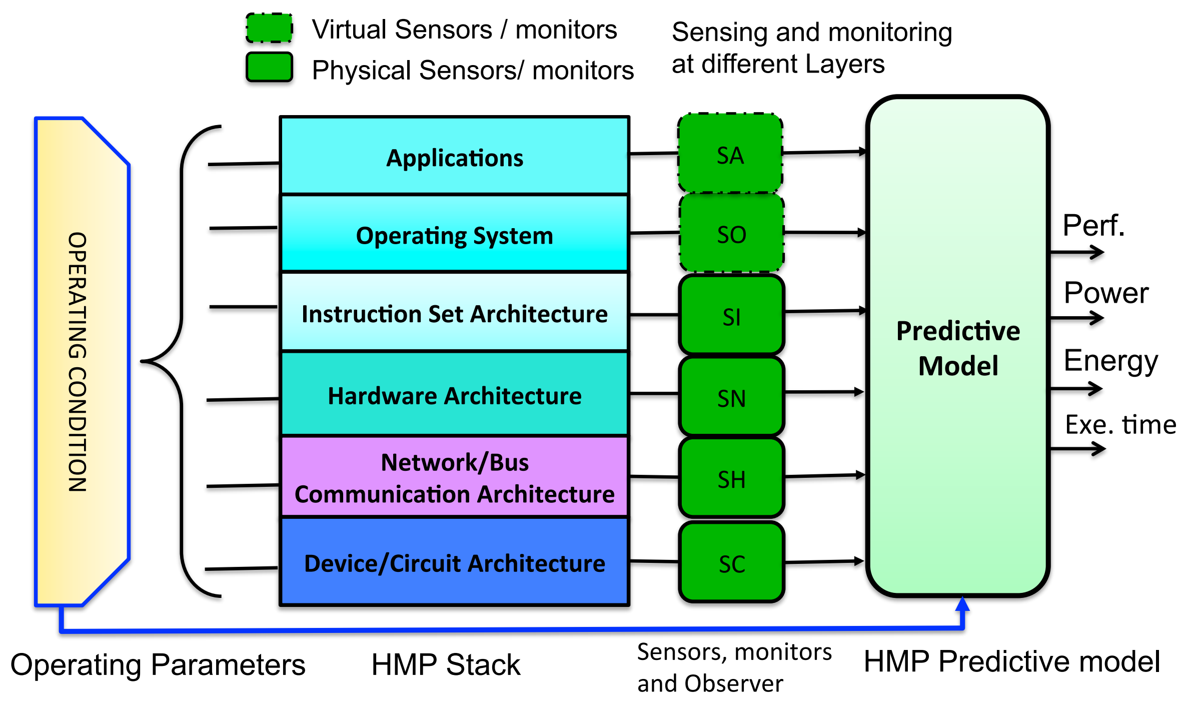

Virtual Sensors / monitors
Physical Sensors/ monitors
Sensing and monitoring at different Layers
OPERATING CONDITION
Applications
Operating System
Instruction Set Architecture
Hardware Architecture
Network/Bus Communication Architecture
Device/Circuit Architecture
SA
SO
SI
SN
SH
SC
Predictive Model
Perf.
Power
Energy
Exe. time
Operating Parameters
HMP Stack
Sensors, monitors and Observer
HMP Predictive model

Figure 5.2: Prediction Phase : use of the cross-layer predictive model showing use of features from different layers of the stack during prediction.

5.2.1 Application and Workload Models

We model the workload and their diversity (program phases, CPU, memory, and IO intensive workloads) using a task (thread) based model and expose the performance and power/energy characterization matrices to the OS where:

- $V = \{v_i\}$ is the set of tasks (or interchangeably used for threads). v_i stands for task i where $1 \leq i \leq m$ and m is the number of tasks. Without loss of generality, we use task or group of tasks to represent workload/application. Let N_i represent the computational workload (measured in terms of number of instructions) of task v_i.

- $\mathbf{S} = [\mathbf{ips_i}] = \{ips_{ij}, 1 \leq i \leq m, 1 < j \leq n\}$ is the average throughput matrix (measured in terms of instructions per sec) when executing the tasks on different processors. $ips_{ij}(= IPC_{ij} * F_j)$ represents the average throughput when task v_i executes on processor p_j and is the product of the IPC_{ij} (instruction per cycle) and the frequency of the core F_j. The IPC_{ij} can be measured directly from the processor's built-in performance counters [116].

- $\mathbf{\Gamma} = [N_i/s_{ij}] = \{\tau_{ij}, 1 \leq i \leq m, 1 < j \leq n\}$ is the average execution time (or the time span) matrix. τ_{ij} in average execution time of task v_i on processor p_j.
- $\mathbf{P} = [\mathbf{pw_i}] = \{pw_{ij}, 1 \leq i \leq m, 1 < j \leq n\}$ is the average power consumption matrix of tasks executing on different processors. $\mathbf{pw}_i = \{pw_{ij}, 1 \leq j \leq n\}$ represents a vector of all the average powers of task v_i executing on each processor. pw_{ij} represents the average power of task v_i executing on processor p_j and it varies with time. The power consumption pw_{ij} of a task v_i can be computed by using combination of performance counters [116].
- $\mathbf{\Xi} = [\varepsilon_{ij}] = \{pw_{ij} \times \tau_{ij}\}$ is the average energy consumption defined as the product of the average power consumption and execution time.

Table 5.2: Heterogeneity-Aware Task Allocation Strategies for Given HMP Composition

Sl No	Platform Design Goal	Allocation	Problem Definition	Objective Function	Nomenclature
1	Performance Maxmization (PerfMax)	$minD$	Find $\mathbf{\Psi_D}$ $\exists\, J_D$ is minimized	$t_{opt} = min\{J_D\} = min\{max\{t_j\}\}$ $J_D = max\{t_j\};\ t_i = \sum_{i=1}^{k} \tau_{ij}$ $1 \leq j \leq n$	t_j represents total execution time of the tasks in processor p_j
2	Energy Minimization (EnergyMin)	$minE$	Find $\mathbf{\Psi_E}$ $\exists\, J_E$ is minimized	$\Xi_{opt} = min\{J_E\};\ J_E = \sum_{j=1}^{n} \xi_j$ $\xi_j = \sum_{i=1}^{k} \varepsilon_{ij} = \sum_{i=1}^{k} pw_{ij}.\tau_{ij};$ $1 \leq j \leq n$	ξ_j represents sum of total energy consumed by k task in processor p_j
3	Power Minimization (PowerMin)	$minED$	Find $\mathbf{\Psi_{ED}}$ $\exists\, J_{ED}$ is minimized	$J_{ED} = min\{J_E.J_D\}$	Energy Delay Product
4	Energy Efficiency Maximization (EEMax)	$minED^2$	Find $\mathbf{\Psi_{ED^2}}$ $\exists\, J_{ED^2}$ is minimized	$J_{ED^2} = min\{J_E.J_D^2\}$	Energy Delay Square Product

5.2.2 Heterogeneity-Aware Task Allocation Model

The task allocation problem of multicore processor consists of finding an optimal distribution of tasks on a set of processors $PE = \{p_1, p_2, ..., p_n\}$. It is assumed that each processor runs independently, but can only run one task at any instant of time. We call an assignment of all tasks $V = \{v_1, v_2, ..., v_m\}$ to available processors $PE = \{p_1, p_2, ..., p_n\}$ a "schedule" $\mathbf{\Psi}$ represented as:

$$\begin{gathered} \Psi = \{\Psi_j, 1 \leq j \leq n\} \\ \Psi_j = \{\psi_i, 1 \leq i \leq m\}, \forall \psi_i \in V = \{v_1, v_2, ..., v_m\}, \end{gathered} \tag{5.11}$$

where Ψ_j represents the schedule of the set of task for the processor p_j and ψ_i represents a task among the set of tasks $V = \{v_1, v_2, ..., v_m\}$ that is mapped to processor p_j. A schedule as defined in Eq. (5.11) will result in a total execution time and power distribution consumption as a function of the task allocation taking into account the heterogeneity of processing elements and workload. In other words, for different allocation strategies, the total execution time and energy consumption in the multicore processor system will be different. Thus, the cross-layer DSE determines a schedule $\mathbf{\Psi}$ for the given set of tasks that meets an objective or a performance index as defined in Table 5.2.

5.3 Predictive Modeling of Performance and Power of Different Core Types

The predictive model based on response surface model (RSM) as described in [2, 268] is a closed-form analytical expression suitable for predicting the quality of nonsimulated design points. Predictive model techniques are typically introduced to decrease the time due to the evaluation of the system-level objective function $\mathbf{J}(\mathbf{x})$ for each architecture $\mathbf{x}$. A response surface model for the function $\mathbf{J}(\mathbf{x})$ is an analytical function $\mathbf{r}(\mathbf{x})$ such that

$$\mathbf{J}(\mathbf{x}) = \mathbf{r}(\mathbf{x}) + \epsilon \tag{5.12}$$

where ϵ is the estimation error. Typically, an appropriate predictive model for $\mathbf{J}(\mathbf{x})$ is such that it has some desired statistical properties such as a mean of zero and small variance. The working principle of predictive model is to use a set of simulations generated by design of experiments (DoE) in order to build the response model of the system. A typical predictive model based flow involves : a training phase, in which known data (or training set) are used to identify the predictive model configuration; and a prediction phase, in which the predictive model is used to forecast the unknown system response. In this work, we use linear regression techniques to construct the predictive model by taking into account the interaction between the parameters and the quadratic behavior with respect to a single parameter using the general model discussed in [268].

In order to concisely encapsulate the effects of performance, power, and workload behavior we need an effective approach to determine and represent the performance, power, and the energy matrices described in Subsection 5.2.1. We use combination of measurement and on-line prediction to construct these matrices. Estimation as well as prediction of the performance and power matrices are possible as there is a direct correlation between the behavior of different core types. The key idea behind the estimation and prediction of execution time and power matrix relies on the fact that the performance of a task on one core is correlatable to the performance in another core (with same-ISA and memory hierarchy) with a good degree of accuracy. By measuring the performance of the task in one processor we can predict the performance in another processor. The execution time τ_{ij} of a task v_i on the processor p_j can be defined as

$$\begin{gathered} \tau_{ij} = \frac{N_i}{IPC_{ij} * F_j} = \frac{N_i}{ips_{ij}} \\ IPC_{ij} = 1/CPI_{ij}. \end{gathered} \tag{5.13}$$

We develop core specific performance (throughput) predictors and then combine them to obtain performance prediction of the combined total platform. The average throughput IPC_{ij} is for a given task v_i running on processor p_j is predicted by using a linear predictor

$$IPC_{ij} = \Phi_j * X_{ij}^T \tag{5.14}$$

where Φ_j is constant vector of a predictive model [360, 15] , $X_{ij}^T = [x_{1i}, x_{2i}, .., x_{qi}]_j^T$ is a characterization vector of core architectural features and hardware counter (cycle counters, instruction counters, performance degradation events) values that is used to predict the performance for the core p_j for the task v_i. The cross-layer features and hardware architecture counters are used in the prediction. We currently use the following static features and dynamic hardware performance counters:

- **Hardware Architecture Features:** Issue width (I_w),LQ/SQ size (LSQ),IQ size (IQ),ROB size (ROB),Int/float Regs (IFR),L1\$I size (KB) (L_{1I}),L1\$D size (KB) (L_{1D}),Freq. (MHz) (F),Voltage (V), Core Area (a).

- **Performance Events Counters:** We measure the following events which are known to derive the performance of a core [15, 360]: mispredicted branches, which are used to compute the branch misprediction rate (m_B); instruction/data L1 caches and TLBs misses and hits, which are used to compute the L1 instruction miss rate (m_{L1I}), L1 data cache miss rate (m_{L1D}), instruction TLB miss rate (m_{ITLB}), data TLB miss rate (m_{DTLB}), and Context switch counters (Cw).

- **Cycle and Instruction Counters:** We sample the amount of *busy cycles* (cy_{Busy}), *idle cycles* (cy_{Idle}), and *sleep cycles* (cy_{Sleep}) of a core. *Busy cycles* represent the time a core spends doing computation, *idle cycles* capture idling time due to pipeline stalls or cache misses, and *sleep cycles* capture the time a core spends in a quiescent state. We use the total amount of *committed instructions* (I_{total}), *committed load and stores* (I_{mem}), and *committed branches* (I_{branch}).

Similarly, we also compute the power consumption of each task for all the cores by measuring the power consumption in a core and suitably scaling it by the scaling factor among the cores using a linear predictor described below:

$$pw_{ij} = \Theta_j * X_{ij}^T \tag{5.15}$$

where $\Theta_j = [\theta_1, \theta_2, ..., \theta_q]_j$ is a constant vector obtained by fitting the data of the benchmarks, and $X_{ij}^T = [x_{1i}, x_{2i}, .., x_{qi}]_j^T$ is the architecture feature and hardware counters (cycle counters, instruction counters, performance degradation events). The computational complexity and accuracy of the predictors for sample benchmarks are shown in Table 5.4.

5.3.1 Predictive Model Evaluation Results

To demonstrate the power and efficacy of our cross-layer predictive modeling approach we perform several experiments that are representatives of recent heterogeneous multicore architectures (for example ARMs big.LITTLE chip [120]) and quantify the benefits of different architectural configurations. We use different classes of Alpha Processors [181, 203] whose specifications are given in Table 5.3 to construct realistic HMP in Gem5 [49] by specifying the multi-layer architectural features. Note that the performance of the processors in terms of average IPC, Area, and Power are normalized with respect to the smallest EV4 (Alpha 21064) core. We also observe that the asymmetric increase of approximately $82\times$ in chip area just to double the performance of a EV8 core with respect to EV4 core. This asymmetry (or heterogeneity) in scaling is essentially exploited by HMPs to achieve performance, power, and energy efficiency for a given area budget.

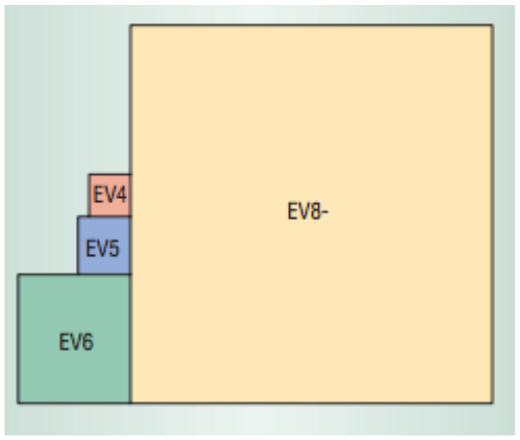
We apply the cross-layer approach as described in Fig. 5.1 by considering chip/die area budget of four Alpha 21264 (EV6) processors as the system-level constraint. We find all the possible distinct combinations with three classes of processors (EV4, EV5, and EV6) that meet the area budget and number the 37 possible candidate configurations as shown in Fig. 5.3. To represent a diverse set of workload we select 8 Mediabench-II algorithms and PARSEC benchmarks as representative workloads and measure their execution time and power consumption as shown in Fig. 5.4 using combination of Gem5 [49] and McPAT[218, 219] respectively. We measure the performance and power values for each processor core type as shown in Table 5.3 in the full system mode. To consider the effect of varying workload and other micro-architectural effects, we vary the number of threads in the benchmark program, with different inputs in the cycle accurate full system Gem5 simulation to create the training data. We consider each of these benchmarks as a single threaded task. To simulate the effect of diverse multi-threaded workloads on the platform we use PARSEC benchmarks or gradually increase the number of single threaded tasks and perform the allocation for a given platform. We use the combination of these benchmarks to form new composite tasks (e.g., JPEG compression followed by AES encryption) and compute the execution time and power consumption as the sum of the individual benchmarks respectively. This allows us to test architecture configurations with more than 100 cores (e.g., area budget of 4 EV8 results in 46428 configuration with as many as 330 EV4 cores).

We perform an initial set of simulations generated by DoE in order to build the response surface model of the system for four design objectives, i.e., make-span/delay, power, energy, energy-delay product (EDP), and energy-delay square product (ED^2P) as listed in Table 5.2. We use these initial simulation points to construct the predictive models by using linear regression and obtain the coefficients of the expression in Eq. (5.14) by performing least square fitting of the data. Table 5.4 shows the computational complexity and accuracy of the predictors in comparison to a full system simulation (over two orders of magnitude at maximum prediction error of 10%) of the platform for few benchmarks. Fig. 5.5 shows the evaluation of the cross-layer predictive model accuracy (within 5% error) for EV6 and EV4 cores for different benchmarks.

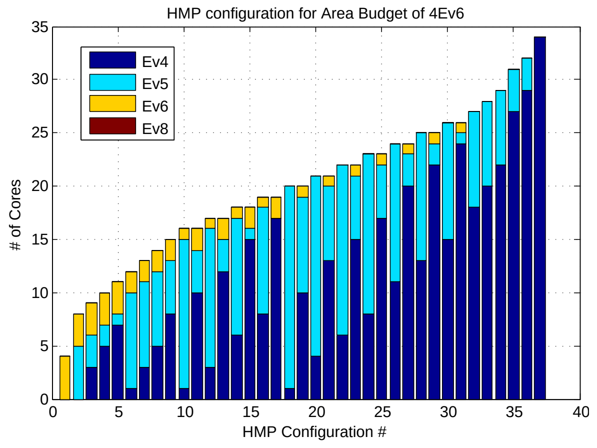
Table 5.3: Alpha Processor Cores Performance, Area and Power [203]

Alpha Core	Issue Width	I-Cache	D-Cache	Branch Prediction	# MSHRs	IPC*	Area*	Peak Power(W)	Avg. Power(W)	Power*
EV4	2	8KB, DM	8KB, DM	2KB, 1-bit	2	1.00	1.00	4.97	3.73	1.00
EV5	4	8KB, DM	8KB, DM	2K-, gshare	4	1.30	1.76	9.83	6.88	1.84
EV6	6	64KB, 2 Way	64KB, 2 Way	Hybrid, 2 level	8	1.87	8.54	17.8	10.68	2.86
EV8	8	64KB, 4 Way	64KB, 4 Way	Hybrid, 2×EV6 size	16	2.14	82.2	92.88	46.44	12.45

* Normalized versus EV4; All cores scaled to 0.1 μm, at 2.1 GHz; IPC based on SPEC CPU benchmarks.



(a)



(b)

Figure 5.3: (a) Relative core sized for the Alpha processor cores. (b) HMP configurations for area budget of 4×EV6. Total of 37 configurations numbered from 1 to 37 from left to right.

5.4 Application of Predictive Models in DSE of HMPs

In this section, we present a cross-layer approach for exploring and configuring a HMP for a given goal under system level constraints (such as equal area or power budget). Single-ISA based heterogeneous multicore processors (HMP) are increasingly considered as an attractive design alternative to homogeneous multiprocessors because of their superior performance, power, and energy efficiency while providing the flexibility of using the same software (binaries) and development tools across cores for a range of applications. HMPs can effectively address complex requirements of diverse applications by executing workloads (or tasks) in the most appropriate core types to meet

Table 5.4: Execution Time and Prediction Model Performance on Intel i7 2.4 GHz Machines.

Benchmarks	HMP Config	Gem5 Full System Simulation Time	Prediction Model Execution time	Prediction Error
H.264	#1 (4 cores)	> 2 days	< 1 sec	<5%
Bodytrack	#2 (8 cores)	> 4 days	< 1 sec	<5%
Blackscholes	#10 (16 cores)	> 7 days	< 1 sec	<8%
Fluidanimate	#10 (16 cores)	> 7 days	< 1 sec	<8%
Mix of above	#36 (32 cores)	>10 days	< 1 sec	<10%

competing and conflicting objectives / figures of merits (e.g., performance, power, energy, throughput, area, cost etc.) [27, 203, 201, 135]. Since different workloads (e.g. CPU bound, integer-intensive, floating-point intensive, memory intensive etc.) require different resources, a key issue is to determine and select the right types and number of cores (processing elements) for an allocation strategy that maps the workload (or tasks) to right core type such that the type of workload will best benefit from the given platform. The selection of number and type of cores is not straightforward when the applications executed by these HMPs face diverse workload characteristics. When designing such a system, a chip architect must decide how to distribute the available limited system resources, such as area and power, among all the processor cores.

In this example use case, we apply the cross-layer predictive models developed using the approach discussed in Section 5.2. Unlike the state-of-the-art approaches, we jointly consider features of the application, operating system (task allocation strategies), and hardware architecture while deploying computationally efficient predictive models (of performance and power) in composing the HMP platform resources (number and types of cores). Our predictive cross-layer approach enables the designer to comparatively evaluate and select the most promising (e.g., energy and performance efficient) HMP configuration in over two order of magnitude less simulation time especially during the early design and verification stages when the design space is at its largest. Our model building methodology combines the design of experiments (DoEs) [2] and predictive model techniques to predict the quality of the nonsimulated design points thereby speeding up the exploration process while reducing the number of required simulations. While the DoE phase generates an initial plan of experiments used to create a coarse view of the target design space to be explored by simulations, the predictive model, which is a closed-form expression of objective (figure of merit) space as a function of the parameter space is useful during the design space exploration (DSE) phase to quickly converge to the Pareto set of the multi-objective problem without executing lengthy simulations. We use the modeling and optimization techniques proposed in [268] to iteratively update the predictive models (as shown in Fig. 5.1) while simulating different parts of the system stack as discussed in the subsequent sections.

5.4.1 DSE Problem Formulation for the HMPs

We consider a shared memory HMP architecture as shown in Fig. 5.6 consisting of K types of core represented using a set $\mathbf{\Pi} = \{\pi_1, \pi_2, ..., \pi_K\}, \pi_i \neq \pi_j, K > 1$ having corresponding areas $\mathbf{\Lambda} = \{a_1, a_2, ..., a_K\}$. Let the set of all processing elements be $PE = \{p_1, p_2, ..., p_n\}$ where p_j is an instance of a core type in $\mathbf{\Pi}$. The HMP consists of the core combinations $\mathbf{C} = \{n_1, n_2, .., n_K\}$ such that the total number of processors in the HMP is $n = \sum_{l=1}^{K} n_l$, where n_l is the number of processors of type π_l and the total area $A = \sum_{i=1}^{K} a_i * n_i$. Let A_{max}, P_{max} be the respective maximum die area and allowable power consumption in the design of the HMP. Our goal is to find the HMP core combinations $\mathbf{C}$ such that a platform objective $\mathbf{J}$ (e.g., power/energy efficiency) is optimized under system constraints (such as area or power) as below:

$$\begin{array}{ll} \underset{\mathbf{C}}{Maximize} & (\mathbf{J}) \quad . \\ s.t & A \leq A_{max} \end{array} \tag{5.16}$$

As the design space for HMP composition problem in Eq. (5.16) is extremely large, we make a few assumptions and approximations to reduce the space. First, we assume that the number of core types K is small (<5). Second, as there is a hard constraint on system area and power resources, the composition space of $\mathbf{C}$ can be reduced to a set of possible configurations $\mathbb{C} = \{C_1, C_2, .., C_N\}$. Then our problem is to chose one of these C_i for a given set of workloads and OS level workload allocation strategy that optimizes the system goal $\mathbf{J}$.

5.4.2 HMP Configuration Selection

Once the response surface of the system goal is formed using the predictive models, different search heuristics can be used to find the most suitable HMP configuration. As an example, we search for the configuration that performs the best in most cases as the number of threads (or load) is increased. Other heuristics can also be used to select the configurations from the Pareto front.

5.4.3 Experimental Results for DSE

In this section, we present the DSE results using the cross-layer predictive model for selecting the HMP configurations for given platform goals. Our cross-layer DSE shows that an allocation strategy that performs well with one architecture configuration does not perform equally well for another architecture configuration and there is a rich design space to exploit for a specific solution. With the variability in number of

Table 5.5: Architectural Composition with Different System Goals and Allocation Strategies.

Goals/Objective	J_D (minD)	J_E (minE)	$J_{ED}(minED)$	$J_{ED^2}(minED^2)$
$\mathbf{C_{PerfMax}}$	#1 (4×EV6)	#1(4×EV6)	#1(4×EV6)	#1(4×EV6)
$\mathbf{C_{EnergyMin}}$	#37(34×EV4)	#9(8×EV4,5×EV5,2×EV6)	#37(34×EV4)	#37(34×EV4)
$\mathbf{C_{PowerMin}}$	#9(8×EV4,5×EV5,2×EV6)	#37(34×EV4)	#2(5×EV5,3×EV6)	#2(5×EV5,3×EV6)
$\mathbf{C_{EEMax}}$	#9(8×EV4,5×EV5,2×EV6)	#37(34×EV4)	#9(8×EV4,5×EV5,2×EV6)	#9(8×EV4,5×EV5,2×EV6)

tasks, the objective functions of each architectural combination with system goal $minD$ is shown in Fig. 5.7. The predictive models demonstrate the relative merit for heterogeneous multicore processor configurations for the same area budget and different allocation strategies. Furthermore, we compare the allocation strategies with a heterogeneity oblivious random allocation with variability in number of task and the execution time as shown in Fig. 5.8. The joint impact of considering the workload variability (with variations in number of tasks and intra task execution time variations) with allocation strategies shows that almost all the HMP configuration will underperform by as much as 50 % and 70% respectively in terms of energy delay product (EDP) and energy delay square product (ED^2P) if a heterogeneity agnostic allocation policy (e.g., random policy as in vanilla Linux Kernel) is used. Thus, heterogeneity-aware allocation strategies are crucial for almost any HMP platform configurations and their impact is significant as the system is loaded with more tasks. We search for the best performing architectures as given in Table 5.5 as the most preferable architectures (most frequently occurring) for different system goals using a given allocation strategy with the given equal area budget constraints. We observe that for the given area budget, the architectural combination #9(8×EV4,5×EV5,2×EV6) with 8 EV4 cores, 5 EV5 and 2 EV6 cores has superior performance in terms of EDP and ED^2P.

5.5 Related Work

HMPs that integrate a mix of small power-efficient cores and big high-performance cores are very attractive alternative to homogeneous multiprocessors because they have the potential for higher performance and reduced power consumption. Contemporary mobile phones have already embraced hardware core heterogeneity, for instance, ARM big.LITTLE architecture [120], and NVidia's Kal-El [252] that have cores of different strengths in one cache coherence domain with the same instruction set architecture (ISA). Architectures with more than two core types are already a reality (e.g., NVidia's Kal-El [252] that integrates four high performance cores, one low performance core, and many GPU cores) and this trend towards heterogeneity is only expected to grow further in the future. Processor cores in a heterogeneous multicore system can differ in static micro-architecture to dynamic behavior or modes of operation (e.g., frequency or operating voltage). Broadly the vast space of HMPs can be classified by core type (strength/size/number/ISA) and heterogeneity levels, as shown in Fig. 5.6.

HMPs provide architecturally diverse cores with drastically different power-performance trade-offs that can be exploited for system efficiency. Consequently, HMPs and their architecture is an active area of research. Although several works have studied HMPs and their runtime systems [201, 36, 322, 223, 15, 70, 360], none of them addresses the problem of platform selection and composition for a given system level constraint such as area or power. Selection and composition of the platform is important at the early stage of the design and a chip architect must decide how to distribute the available limited system resources, such as area and power, among all the processor cores. Moreover, since HMPs are inherently designed as a multi-layered system, either gross approximation or complete neglect of any layer and its features can affect the behavior, misrepresent the intricate multi-layer trade-offs and interactions, as well as misguide the design and composition process. Therefore, in this chapter, we presented a cross-layer approach for configuring a HMP under system level constraints (such as equal area or power budget) as an optimization problem.

The closest to our work is the approach in [390, 389] where an optimization framework based on Lagrange multiplier is presented to allocate system area resource to accelerators such that the total execution time is minimized. First, the approach in [390, 389] is applicable to specialized accelerator units that can be easily synthesized/designed to trade-off area and performance at finer granularity. On the other hand, processor cores are much more complex; they can not be easily designed to trade-off area-performance at finer granularity. Core trade-off characteristics are discrete (resulting in only few core types) and thus a continuously differentiable objective function can not be used for HMPs compositions as used in accelerators. Second, the resource allocation in [390, 389] does not consider operating system and allocation/mapping of task and context switching that is intrinsic in HMPs. In fact, any allocation/mapping approaches (e.g., as used in vanilla Linux Kernel) that are not heterogeneity and workload aware, will have a drastic impact on the HMP platform performance and power [201, 36, 195] and thus in the composition of the HMP platform. In contrast, our approach considers full system OS (vanilla Linux) and four heterogeneity-aware static allocation schemes (that have been shown to be effective [322]) along with realistic benchmarks while configuring the platform. Third, unlike ours, [390, 389] essentially model the performance as function of area and ignores all the other cross-layer design parameters and thus in essence can not be considered a cross-layer approach. Fourth, they [390, 389] assume closed form analytical models for the performance of the hardware accelerator but do not explain how these can be derived and used for multiple accelerators or for application that can not be accelerated. In contrast, we present a more versatile and realistic approach along with a clear methodology to build cross-layer predictive models of application and system interactions that can be used in the HMP composition.

5.6 Summary

In this chapter, we described an approach to build cross-layer predictive models for emerging SoC architectures considering multiple layers of the system stack. The generic predictive modeling techniques based on regression analysis is applied for emerging heterogeneous architectures by selecting informative and independent features. To capture domain specific knowledge of the emerging SoCs is fundamental to the model accuracy and usefulness. While several literatures exist on predictive model of performance and power consumption of homogeneous architectures, however, to the best of our understanding, cross-layer predictive modeling approach for emerging heterogeneous multicore systems has not been addressed. We used these predictive models to investigate the interactions and influence of heterogeneity of hardware architectures (configurations, number and types of cores), multi-objective allocation strategies along with diverse types of workloads under system level constraints (such as equal area or power budget). We presented a principled approach to build cross-layer predictive models of application and system interactions that can be used in the HMP compositions. Our proposed cross-layer approach quantifies the relative merits of one architectural configuration and allocation strategy over others and helps in selecting most promising heterogeneous architectures. Our predictive cross-layer approach enables the chip architect and designer to comparatively evaluate and select the most promising (e.g., energy and performance efficient) HMP configuration in over two order of magnitude less simulation time especially during the early design and verification stages when the design space is at its largest.

Algorithm 5.1 Heterogeneity-aware static task allocation using simulated annealing (SA)

SA Input Params: Temperature T, Temperature schedule c, Maximum number of iterations $Iter_{max}$

Input Data: HMP config $\mathbf{C}$, Throughput Matrix $\mathbf{S}$, Power Matrix $\mathbf{P}$, Execution Time Matrix Γ, Energy Matrix Ξ

Output: Allocation $\mathbf{\Psi}$

1. Set an initial solution $\mathbf{\Psi} = \Psi_0$
2. Obtain a new solution $\mathbf{\Psi'} = \mathbf{\Psi}$ and randomly perturb one of the elements $\mathbf{\Psi'_{ji}}$ of $\mathbf{\Psi'}$. The so-called Bolz-mann generating scheme accomplishes this:

$$\begin{aligned}&idx = j * m + i\\&idx = [idx + \sqrt{T} \times rand()] \,\mathbf{mod}\, (n * m)\\&j' = idx \,\mathbf{mod}\, n,\ i' = (idx - j')/m\\&swap(\mathbf{\Psi'_{ji}}, \mathbf{\Psi'_{j'i'}})\end{aligned}$$

   where $rand()$ generates uniformly distributed random integer numbers.

3. Evaluate the objective function $\mathbf{J}(\mathbf{C}, \mathbf{S}, \mathbf{P}, \Gamma, \mathbf{\Xi})$ for $\mathbf{\Psi'}$
4. Accept (set $\mathbf{\Psi} = \mathbf{\Psi'}$) or reject $\mathbf{\Psi'}$. If the value of the objective function is lower than before the perturbation, always accept. If it is higher, then accept according to the probabilistic rule

$$accept\, if\; rand() < exp\left(\frac{E_0 - E_p}{T}\right)$$

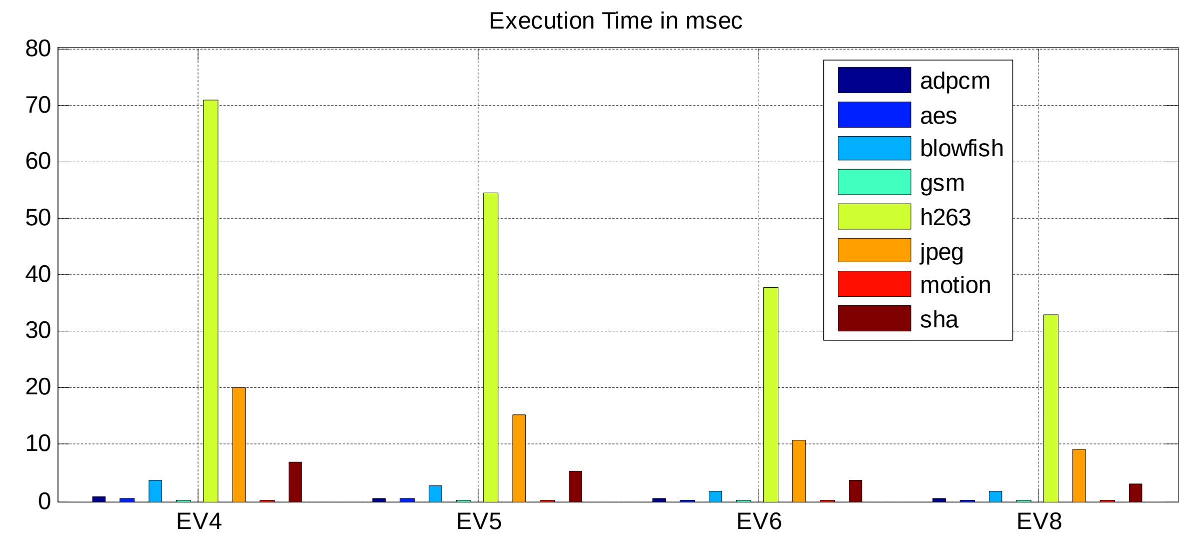
   where $E_0 - E_p$ is the difference in objective function values before and after the perturbation.

5. Decrease the temperature according to the cooling schedule:

$$T = c \times T$$

   where $0 < c < 1$ is a constant.

6. The algorithm stops when the average change in the objective function is small relative to the tolerance, or when it reaches the maximum number of iterations $Iter_{max}$, otherwise, it goes back to step 2.



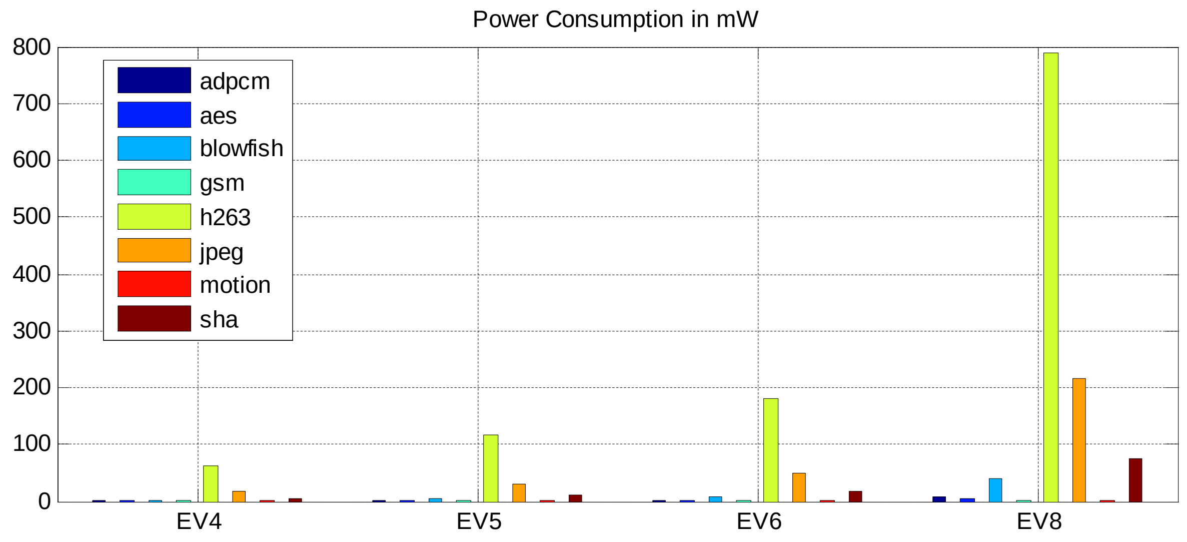
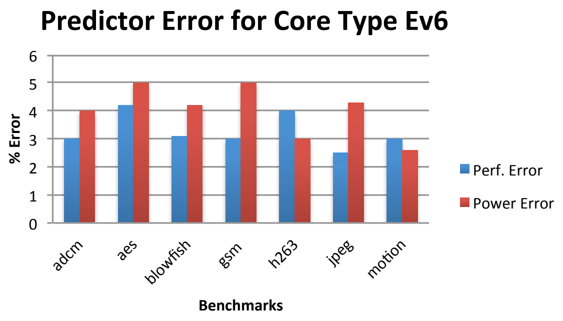
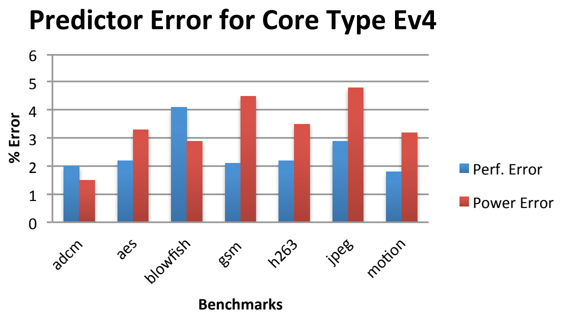


Figure 5.4: Average power and execution time for 8 benchmarks for different Alpha processors.



(a)



(b)

Figure 5.5: Predictive model evaluation (a) accuracy of the predictive model for the Ev6 core (b) Accuracy of the predictive model for Ev4.

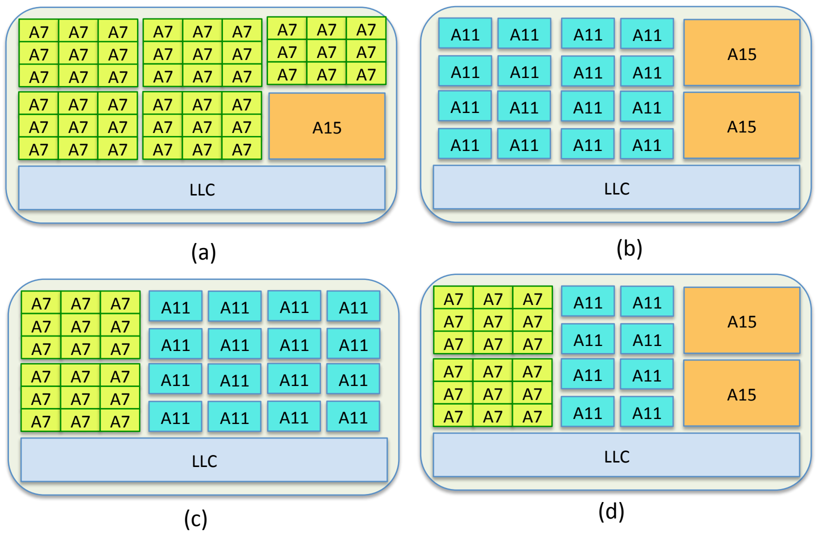


Figure 5.6: Examples of heterogeneous architectures composition for the same die area using Big (A15), Medium (A11) , and Little (A7) cores.

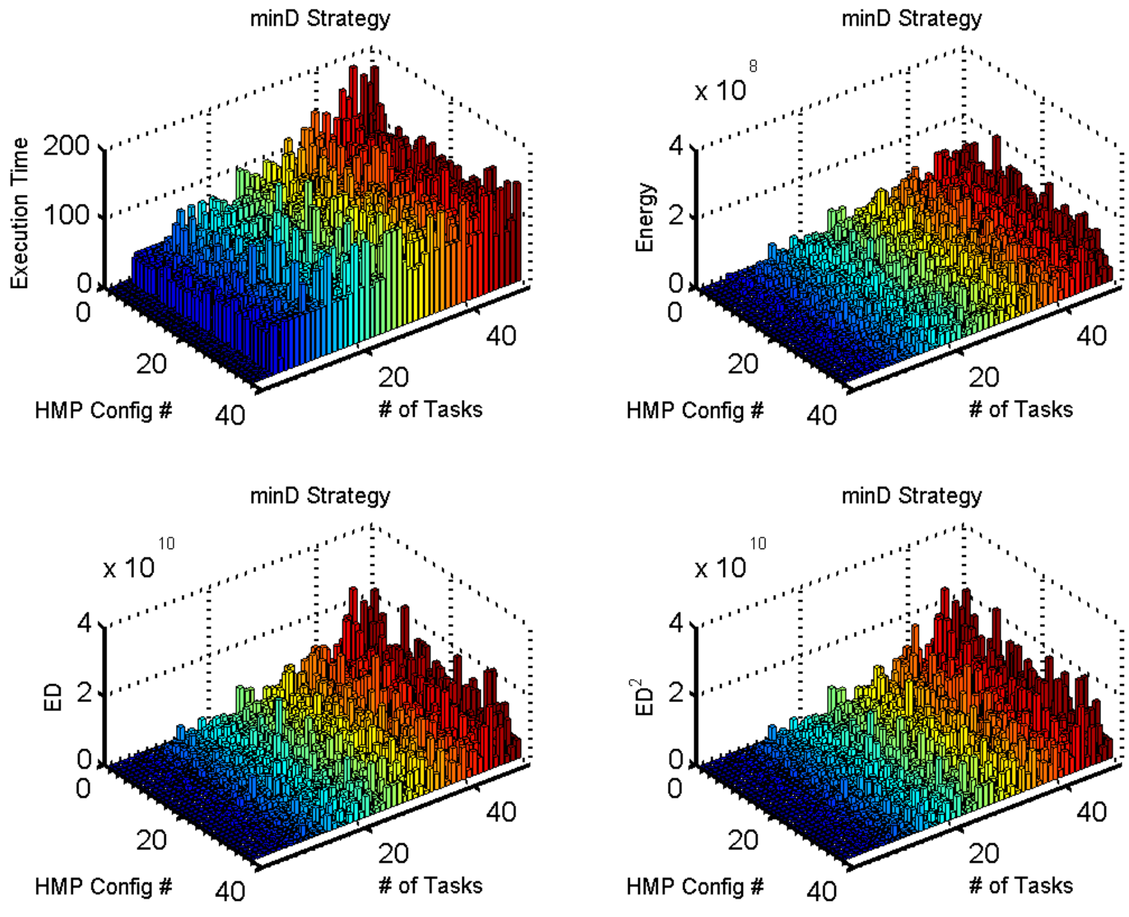


Figure 5.7: Objectives with variability in number of task for delay only task allocation strategy ($minD$) using the predictive models. Lower is better.

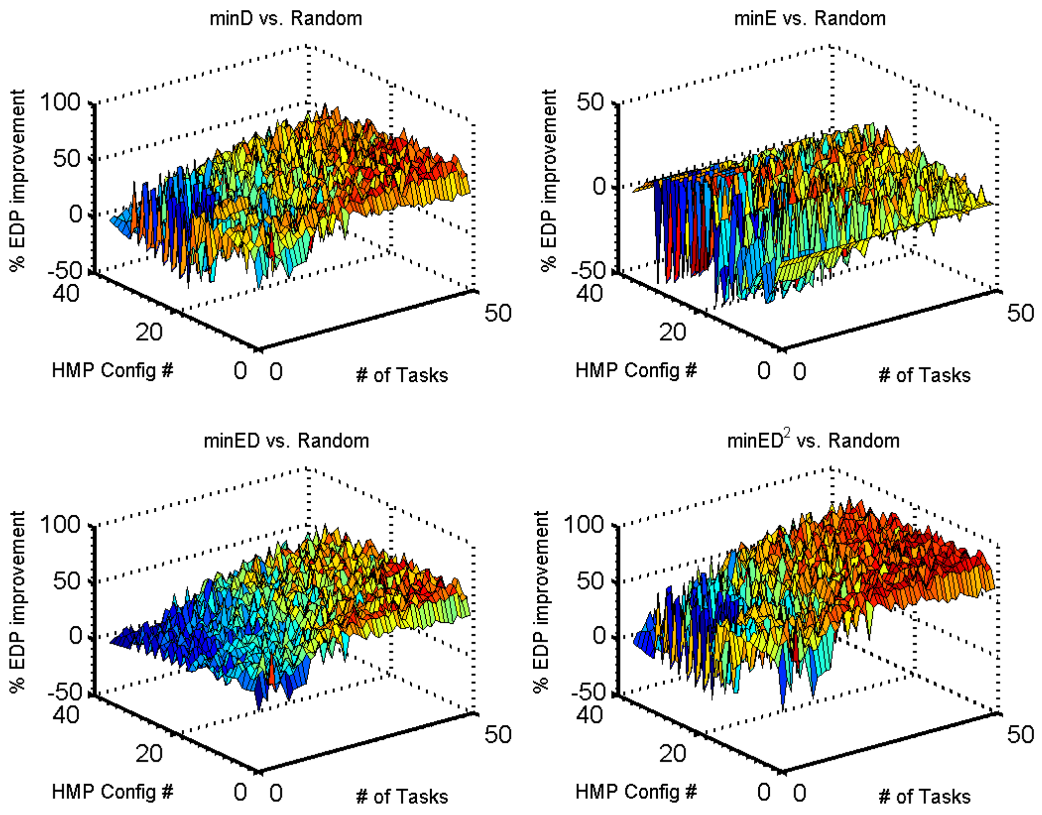


Figure 5.8: DSE with predictive models: Comparison of SA based static allocation strategies with random allocation strategy (as in vanilla Linux). Higher is better.

Chapter 6

State Estimation and Prediction Using Minimal Sensing

Temporal prediction of system states is essential for making proactive decision as well as achieving self-awareness in emerging MPSoCs. In this chapter, we show that state prediction can be made in advance (multiple epochs of time ahead) using sparse measurement and minimal sensors. We present an on-line estimation method using the sparse Kalman filters to estimate and predict the full-state of the system from sparse measurements. In this chapter, we show that temporal prediction of the thermal hotspots and power consumption of multicore systems ahead of time using state-space model and use the predictions to make proactive decision in dynamic thermal management (DTM) and improve the performance by $\sim$28% without violating the thermal limits.

The chapter is organized in the following sections. Section 6.2 defines terminologies and the preliminaries followed by the sparse observability problem defined in Section 6.3 with the description of the properties and complexity of the problem. Algorithms to solve the MSOP are described in the subsequent sub-sections. A specific example use case of thermal sensor placement and run-time in-situ thermal profile estimation and robust hotspot tracking is described in Section 6.5 supported by simulation and experimental results. Section 6.6 provides a brief overview of the related works.

6.1 Motivation

Observing and controlling a dynamical MPSoC system is of paramount importance for achieving adaptation. The ability to experimentally access and accurately observe the internal states of a system offers a means to quantitatively describe dynamic behaviors of any complex SoC. Such dynamics exist in a wide range of systems including many/multi processor thermal and heat-flow networks [147]. A necessary step towards observing a complex dynamical system is to fully understand the observability of complex SoCs with linear dynamics [225]. Consider a system of n nodes described by the following set of ordinary differential equations [224]:

$$\begin{aligned}\dot{\mathbf{x}} &= \widetilde{\mathbf{A}}\mathbf{x} + \widetilde{\mathbf{B}}\mathbf{u} \\ \mathbf{y} &= \widetilde{\mathbf{C}}\mathbf{x}\end{aligned} \tag{6.1}$$

where the vector $\mathbf{x} = [x_1, x_2, ..., x_n]^T$ stands for the states of nodes, $\widetilde{\mathbf{A}} \in \mathbb{R}^{n\times n}$ represents the coupling matrix of the system, in which a_{ij} represents the weight of a directed link from node j to i (for undirected networks $a_{ij} = a_{ji}$), $\mathbf{u_k} = [u_1, u_2, ..., u_p]^T$ are the set of controllers or control inputs, $\widetilde{\mathbf{B}} \in \mathbb{R}^{n\times p}$ is the control matrix, $\mathbf{y}$ are the measurements, and $\widetilde{\mathbf{C}} \in \mathbb{R}^{\mathbf{m}\times\mathbf{n}}$ is the measurement matrix. A system is called observable if we can reconstruct its complete internal states $\mathbf{x}$ from the measured outputs $\mathbf{y}$. Although simultaneous measurement and sensing of all the internal variables offers a complete description of the system's state, in practice experimental access is limited to only a subset of variables due to cost of the sensing infrastructure, placement restrictions, as well as unavailability of suitable and effective sensing mechanisms. Identifying a set of such crucial points that can provide complete insight into the internal dynamics of a complex SoC system is fundamental to effective and high performance emerging MPSoC design.

In this chapter, we explore this fundamental question of observing the internal dynamics of a linear dynamical system using a minimal set of observation points by using the notion of sparsity as developed in the emerging field of compressive sensing [98]. We define the minimal sparse observability problem to find the sparsest measurement vector which makes a linear dynamical complex network system completely observable. We formulate and develop analytical tools to find the minimum number of nodes (sensors) for any arbitrary type of network. The mathematical tools are then used to develop effective algorithms to find the sparsest measurement vector that enables estimation of the internal states of a complex dynamic system from experimentally accessible outputs. The developed algorithms are further used in the design of a sparse Kalman filter (SKF) to estimate the time-dependent internal states of a high performance processor system and its dynamic thermal management (DTM) and control with the minimum number of on-chip sensors.

6.2 Preliminaries and System Model

6.2.1 Dynamic System Model and Model Conversions

We consider the discrete-time equivalent of a linear time-invariant (LTI) system in Eq. (6.1) as follows:

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \\ \mathbf{y}_k &= \mathbf{C}\mathbf{x}_k \end{aligned} \tag{6.2}$$

where $\mathbf{x_k} \in \mathbb{R}^{\mathbf{n}}$ are the system states at k^{th} time instant, $\mathbf{u_k} \in \mathbb{R}^{\mathbf{p}}$ are the system inputs , $\mathbf{y_k} \in \mathbb{R}^{\mathbf{m}}$ are the measurements, $\mathbf{A} \in \mathbb{R}^{n\times n}$,$\mathbf{B} \in \mathbb{R}^{n\times p}$, $\mathbf{C} \in \mathbb{R}^{\mathbf{m}\times\mathbf{n}}$, are system matrices and $\mathbf{x_0}$ is the initial state of the system. Note that the discrete-time system matrices are obtained for the sampling time t_s as [228, 126]:

$$\begin{aligned} \mathbf{A} &= e^{\widetilde{\mathbf{A}}*t_s} \\ \mathbf{B} &= \int_0^{t_s} e^{\widetilde{\mathbf{A}}(t_s-\tau)}\widetilde{\mathbf{B}}d\tau. \end{aligned} \tag{6.3}$$

6.2.2 Observability and Controllability of a System

The system described by Eq. (6.2) is said to be controllable if it can be driven from any initial state to any desired final state in finite time, which is possible if and only if the $n \times np$ controllability matrix

$$\mathbf{Q_c} = \left[\mathbf{B}, \mathbf{AB}, \mathbf{A^2B}, ..., \mathbf{A^{n-1}B}\right] = \mathtt{C}(\mathbf{A}, \mathbf{B}) \tag{6.4}$$

has full rank, that is:

$$rank(\mathbf{Q_c}) = n. \tag{6.5}$$

This represents the mathematical condition for controllability, and is called Kalman's controllability rank condition [171, 228].

Observability, on the other hand, requires us to establish a relationship between the outputs $\mathbf{y_k}$, the state vector $\mathbf{x_k}$, and the inputs $\mathbf{u_k}$ in a manner that we can uniquely infer the system's complete initial state $\mathbf{x_0}$. The linear dynamic system described by Eq. (6.2) is said to be observable if we can reconstruct the system's complete internal state from its outputs, which is possible if and only if the $nm \times n$ observability matrix

$$\mathbf{Q_o} = [\mathbf{C^T}, (\mathbf{CA})^\mathbf{T}, ..., (\mathbf{CA^{n-1}})^\mathbf{T}]^\mathbf{T} = \mathcal{O}(\mathbf{A}, \mathbf{C}) \tag{6.6}$$

has full rank, that is:

$$rank(\mathbf{Q_o}) = n. \tag{6.7}$$

The controllability and observability of a LTI dynamic system is related by the following duality property as described in Theorem (3).

Theorem 3. *A linear dynamical system described in Eq.* (6.2) *is observable (controllable) if and only if the dual system*

$$\begin{aligned} \mathbf{x_{k+1}} &= -\mathbf{A^T x_k} + \mathbf{C^T u_k} \\ \mathbf{y_k} &= \mathbf{B^T x_k} \end{aligned} \tag{6.8}$$

is controllable (observable) [171].

Proof. Substituting the system matrices in Eq. (6.4) produces the observability matrix Eq. (6.6) of the dual system and vice versa. See [171, 228, 63] for the detailed proof. □

6.3 Problem Formulation

We define the minimal sparse observability problem using the above definitions of observability and controllability as follows.

6.3.1 Minimal Sparse Observability Problem (MSOP)

For the dynamic system defined by Eq. (6.2), the minimal sparse observability problem is defined as the sparsest measurement matrix $\mathbf{C}$, i.e., with smallest number of nonzero entries in $\mathbf{C}$, for which the system described by Eq. (6.2) is completely observable. We use the following theorems to show that MSOP is a NP-hard problem.

Theorem 4. *For any $p \geq 1$, finding matrix $\mathbf{B} \in \mathbb{R}^{n\times p}$,with smallest number of nonzero entries in $\mathbf{B}$ such that the system $\mathbf{x_{k+1}} = \mathbf{Ax_k} + \mathbf{Bu_k}$ is controllable is NP-hard.*

Proof. See [255] □

Theorem 5. *The minimal sparse observability problem is NP-hard. In other words, for any $m \geq 1$, finding the matrix $\mathbf{C} \in \mathbb{R}^{m\times n}$ with the smallest number of nonzero entries that will make the system in Eq. (6.2) observable is NP-hard.*

Proof. We use the duality theorem in Theorem (3) to construct a dual system $(-\mathbf{A^T}, \mathbf{C^T}, \mathbf{B^T})$ as in Eq. (6.8). We then use Theorem (4) to prove the NP-hardness of finding the sparsest $\mathbf{C^T}$ matrix that will make the dual system controllable. Since the minimal controllability of the dual system is NP-hard, hence the sparsest $\mathbf{C}$ that will make the original system observable is NP-hard. Hence the minimal sparse observability problem is NP-hard. □

Theorem 6. *For any $p \geq 1, m \geq 1$ finding matrix $\mathbf{B} \in \mathbb{R}^{n\times p}$,$\mathbf{C} \in \mathbb{R}^{m\times n}$ with smallest number of nonzero entries (in $\mathbf{B}$ and $\mathbf{C}$) such that the system $\mathbf{x_{k+1}} = \mathbf{Ax_k} + \mathbf{Bu_k}$ and $\mathbf{y_k} = \mathbf{Cx_k}$ is both controllable and observable is NP-hard.*

Proof. The proof follows from Theorem (4) and (5). For details see [255]. □

6.3.2 Greedy Solution to MSOP

Since the minimal controllability problem as described in [255] and the minimal sparse observability problems are NP-hard, polynomial time optimal solutions are unreachable. A randomized and deterministic algorithm was proposed for the minimal controllability problem in [255]. We extend the algorithm in [255] and propose a Greedy algorithm for the minimal sparse observability problem. The algorithm for the minimal sparse observability is presented in Alg. s(6.1) and Alg. (6.2). A constrained version of the MPOP algorithm with location constraints is listed in Alg. 6.3.

6.4 Sparse Kalman Filter (SKF)

In this section, we describe an approach to estimate and predict the dynamic states from the sparse output measurement obtained from the location identified by the MSOP approach using a Kalman filter as illustrated in Fig. 6.1. The states of the system in presence of noise and variability is given by:

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k + \mathbf{w}_k \tag{6.9}$$

where $\mathbf{x_k} \in \mathbb{R}^{\mathbf{n}}$ are the system states (i.e., the temperatures of each block) at epoch or time instant k, $\mathbf{u_k} \in \mathbb{R}^{\mathbf{p}}$ are the system inputs (i.e., power consumption at each block),

Algorithm 6.1 Minimal Sparse Controllability

Greedy Minimal Controllability Algorithm.
Input: System matrix $\mathbf{A}$
Output: Sparse $\mathbf{B}$ such that System Eq. (6.2) is Controllable

```
1. Initialize $\mathbf{B}$ to zero vector and rank difference $e_r^* = 1$
2. While $e_r^* > 1$,
    (a) For $i = 1..n$
        i. If $\mathbf{B}[i]$=0, then for $j = 1..2n+1$, set $\widetilde{\mathbf{B}}[\mathbf{i},\mathbf{j}] = \mathbf{B} + \mathbf{j} * \mathbf{v}_\mathbf{i}^\perp$ where $\mathbf{v}^\perp{}_i$ is the $i^{th}$ basis vector
        ii. $\widetilde{\mathbf{Q}}_\mathbf{c} = \mathtt{C}(\mathbf{A}, \widetilde{\mathbf{B}}[i,j])$; $\mathbf{Q}_\mathbf{c} = \mathtt{C}(\mathbf{A}, \mathbf{B})$
        iii. Set $e_r(i,j) = rank(\widetilde{\mathbf{Q}}_\mathbf{c})$-$rank(\mathbf{Q}_\mathbf{c})$
        end
    (b) Let $(i^*, j^*) \in arg\ max_{(i,j)} \{e_r(i,j)\}$ and let $e_r^* = e_r(i^*, j^*)$
    (c) if $e_r^* > 0$,set $\mathbf{B} \leftarrow \mathbf{B}$+$j^* * \mathbf{v}_{i*}^\perp$
    end
3. Output $\mathbf{B}$
```

$\mathbf{A}$,$\mathbf{B}$ are system matrices as discussed earlier. The process noise $\mathbf{w}_k$ is zero-mean, white random signals with known covariance matrices, $\mathbf{Q_k} = \mathbf{E}\left[\mathbf{w_k}\mathbf{w_k^T}\right]$. Our objective is to select minimal number of sensors and their placement such that we have minimum number of sensors in the measurement equation:

$$\breve{\mathbf{y}}_k = \breve{\mathbf{C}}\mathbf{x}_k + \breve{\eta}_k \tag{6.10}$$

where $\breve{\mathbf{y}}_\mathbf{k} \in \mathbb{R}^\mathbf{m}$ are the measurement (i.e., the temperature sensor measurement), $\breve{\mathbf{C}}$ is the minimum sensor measurement matrix, and measurement noise $\breve{\eta}_\mathbf{k}$ is zero-mean, white random signals with known covariance matrices $\breve{\mathbf{R}}_\mathbf{k} = \mathbf{E}\left[\breve{\eta}_\mathbf{k}\breve{\eta}_\mathbf{k}^\mathbf{T}\right]$. The process noise $\mathbf{w}_k$ and measurement noise $\breve{\eta}_\mathbf{k}$ are assumed to be mutually uncorrelated.

The Kalman filter maintains the states $\hat{\mathbf{x}}_{\mathbf{k}|\mathbf{k}}$ which means the estimate of $\mathbf{x_k}$ given the measurement $\mathbf{y_k}, \mathbf{y_{k-1}}, \cdots$, and the error covariance of the states, $\mathbf{P}_{\mathbf{k}|\mathbf{k}}$, is the covariance of the states $\mathbf{x_k}$ given the measurement $\mathbf{y_k}, \mathbf{y_{k-1}}, \ldots \mathbf{y_0}$. Kalman filter performs the following recursive processing as in Alg. 6.4 to estimate the states from the measurement for the given input.

Algorithm 6.2 **Minimal Sparse Observability**

Greedy Algorithm for Minimal Sparse Observability Problem.
Input: System matrix $\mathbf{A}$
Output: Sparse $\mathbf{C}$ such that system in Eq. (6.2) is Observable , Minimum number of Sensor n_s, Sensor Locations

```
1. Compute the system matrices for the dual system
   $\overline{\mathbf{A}} = -\mathbf{A^T}$,
2. Find the sparse control matrix for the dual system using the Alg. (6.1)
   $\overline{\mathbf{B}} =$ minimal_sparse_controllability($\overline{\mathbf{A}}$)
3. Compute the controllability matrix of original system as
   $\mathbf{C} = \overline{\mathbf{B}}^T$
4. Output
   (a) Minimum No of Sensors $n_s = rank(\mathbf{C^T})$
   (b) Sensor Locations are independent rows of $\mathbf{C^T}$
   (c) Measurement matrix $\mathbf{C}$
```

6.5 Run-time Thermal Estimation and Hotspot Tracking

We use this SKF approach for runtime thermal estimation and hotspot detection of the MPSoC.

6.5.1 Thermal Dynamic Model of MPSoC

The thermal behavior of the multi/many core processor is modeled using heat-flow dynamics [335]. The heat-flow dynamics describe the temperature values at different locations on the die depending on various factors such as power consumption of functional units, layout of the chip and the package characteristics. The differential equations describing the heat flow have a form dual to that of electrical current, represented using lumped values of thermal R and C networks, and forms the basis for commonly used micro-architectural thermal models [335, 147]. This complex dynam-

Algorithm 6.3 Constrained Minimal Sensor Observability (CMSO)

Greedy Algorithm for Constrained Minimal Sensor Complete Observability (CMSCO) Algorithm.
Input: System matrix $\mathbf{A}$, Location Constraints S_c that are to be avoided
Output: Sparse C such that System is Observable, Sensor Location $\mathbf{L}$

```
1. Initialize $\mathbf{C}$ to zero matrix and rank difference $e_r^* = 1$
2. Let $S = \{1..n\}$ be the set of all possible sensor location
3. Remove the Constraints locations $S_c$ to form the valid set of location $S_v = S - S_c$
4. Let $n_v = |S_v|$
5. While $e_r^* > 1$,
   (a) For $i = 1..n_v$
       i. If $\mathbf{C}[S_v(i), S_v(i)]$=0, then set $\widetilde{\mathbf{C}} = \mathbf{C} + \mathbf{randn} * \mathbf{v}^{\perp}_{\mathbf{S_v(i)}} * (\mathbf{v}^{\perp}_{\mathbf{S_v(i)}})^{\mathbf{T}}$ where $\mathbf{v}^{\perp}{}_i$ is
          the $i^{th}$ basis vector
       ii. $\widetilde{\mathbf{Q}}_{\mathbf{o}} = \mathcal{O}(\mathbf{A}, \widetilde{\mathbf{C}})$; $\mathbf{Q_o} = \mathcal{O}(\mathbf{A}, \mathbf{C})$
       iii. Set $e_r(i) = rank(\widetilde{\mathbf{Q}}_{\mathbf{o}})$-$rank(\mathbf{Q_o})$
       endfor
   (b) Let $(i^*) \in arg\ max_{(i)}\ \{e_r(i)\}$ and let $e_r^* = e_r(i^*)$
   (c) if $e_r^* > 0$,set $\mathbf{C} \leftarrow \mathbf{C}$+$\mathbf{v}^{\perp}_{\mathbf{S_v(i*)}} * (\mathbf{v}^{\perp}_{S_v(i*)})^T$
   endwhile
6. Output $\mathbf{C}$, Sensor Locations $\mathbf{L} = find(diag(\mathbf{C})$>0)
```

ical thermal network is represented in state space form [147, 126] with the grid cell or subsystem block temperatures as states and the power consumption of each block as inputs to this system. The outputs of this state space model are the temperatures at the sensor locations which can be observed by the temperature sensor readings. The system matrices $\mathbf{A}$ and $\mathbf{B}$ are constant and are computed based on the floor-plan of the processor and the process parameters [126]. We consider the Alpha 31386 processor and its multi-core architectures that have been extensively used in previous research works [335, 147, 126]. A quad-core Alpha 31386 processor floorplan is shown in Fig. 6.2 where each processor core has 18 functional blocks/ subsystem units. Fig. 6.3 shows the blocks in different layers of the chip and package. Note that the block in the top most layer, i.e., the die only consumes power and the blocks in the thermal interface layer, heat spreader, and the heat sinks help in dissipating the heat generated in the die.

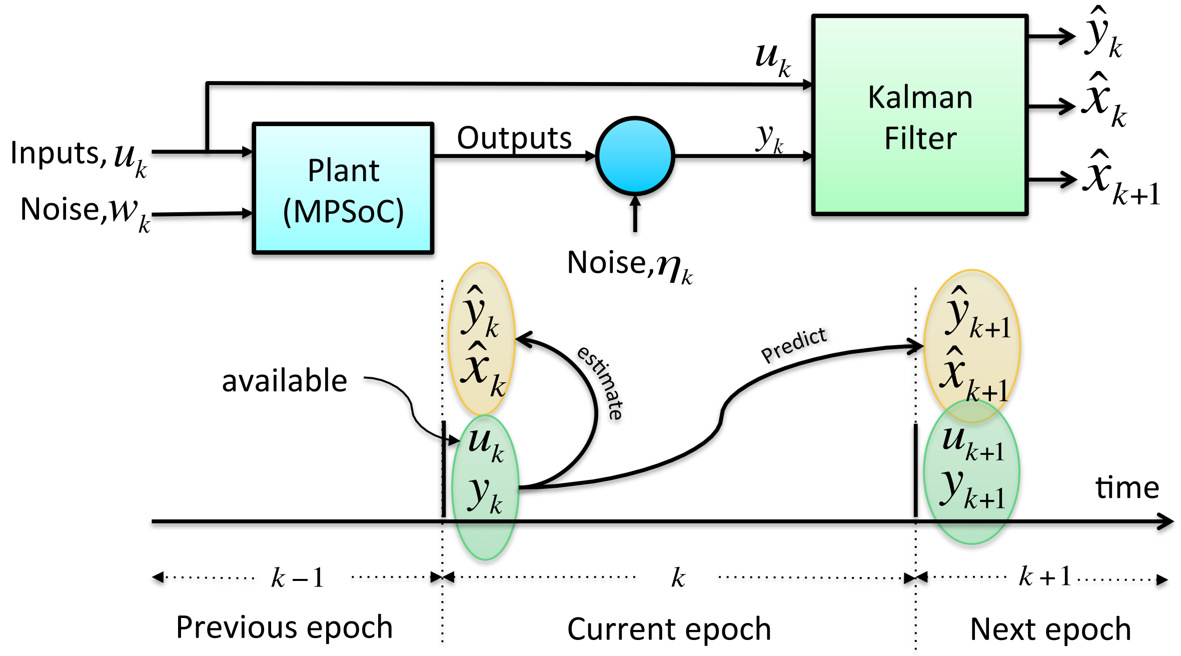


Figure 6.1: Kalman filter overview.

The equivalent RC network representation of the processor's thermal dynamic system is shown in Fig. 6.3(b).

6.5.2 Minimum Sensor Set and Their Optimal Placement

With increasing number of cores in the processor and projection of hundreds and even thousands of cores [51], the thermal dynamics of such a processor can be extremely complex with more than thousands of blocks or functional units. Consequently, it is prudent to consider them as large complex dynamic networks, requiring systematic analysis. We use the mathematical tools and algorithms developed in Section 6.3 to perform sparse observability analysis on these networks. We consider the Alpha 31386 processor and its multi-core architectures as discussed in the previous Section 6.5.1. Our objective is to determine if the thermal network of the Alpha processor is completely observable and find the sparsest measurement matrix C such that the processor network is completely observable. We use the greedy algorithm in Alg. 6.2 to find the sparsest measurement matrix C. The algorithm returns that the single-core Alpha 31386 processor thermal network is completely observable with a single sensor. Consequently, we would be able to estimate the temperature of the 18 subsystem units of the processor (resulting in 2x18+14=50 nodes on the network, see Fig. 6.3(a)) and autonomously track them from a single sensor measurement. The placement of the sensor returned by the algorithm is the first block in the processor floorplan with a sensor gain

Algorithm 6.4 **Sparse Kalman Filter**

Minimum sensor state estimation using Kalman filter.
Input: Measurements $\breve{\mathbf{y}}_\mathbf{k}$, input $\mathbf{u_k}$, state space model $\mathbf{A}, \mathbf{B}, \breve{\mathbf{C}}, \mathbf{Q_k}, \breve{\mathbf{R}}_\mathbf{k}$, Initial Values $\mathbf{x_0}, \mathbf{P_0}$
Output: State estimates $\hat{\mathbf{x}}_\mathbf{k}$, measurement estimate $\hat{\mathbf{y}}_\mathbf{k}$

```
1. Initialize the values of $\hat{\mathbf{x}}_{\mathbf{0|-1}} = \mathbf{x_0}$,$\hat{\mathbf{P}}_{\mathbf{0|-1}} = \mathbf{P_0}$,
2. Perform the following every sampling step :
   (a) Predict states and states covariance:
       i. $\hat{\mathbf{x}}_{\mathbf{k|k-1}} = \mathbf{A}\hat{\mathbf{x}}_{\mathbf{k|k-1}} + \mathbf{B}\mathbf{u_k}$
       ii. $\mathbf{P}_{\mathbf{k|k-1}} = \mathbf{A}\mathbf{P}_{\mathbf{k-1|k-1}}\mathbf{A}^\mathbf{T} + \mathbf{Q}_{\mathbf{k-1}}$
   (b) Calculate the Kalman gain, update the states and states covariance:
       i. $\mathbf{K_k} = \mathbf{P}_{\mathbf{k|k-1}}\breve{\mathbf{C}}^\mathbf{T}\left(\breve{\mathbf{C}}\mathbf{P}_{\mathbf{k|k-1}}\breve{\mathbf{C}}^\mathbf{T} + \mathbf{R}\right)^{-1}$
       ii. $\hat{\mathbf{x}}_{\mathbf{k|k}} = \hat{\mathbf{x}}_{\mathbf{k|k-1}} + \mathbf{K_k}\left(\breve{\mathbf{y}}_\mathbf{k} - \breve{\mathbf{C}}\mathbf{x}_{\mathbf{k|k-1}}\right)$
       iii. $\mathbf{P}_{\mathbf{k|k}} = (\mathbf{I} - \mathbf{K_k}\breve{\mathbf{C}})\mathbf{P}_{\mathbf{k|k-1}}$
3. Output the results:
   (a) State estimate: $\hat{\mathbf{x}}_\mathbf{k} = \hat{\mathbf{x}}_{\mathbf{k|k}}$
   (b) Measurement estimate: $\hat{\mathbf{y}}_\mathbf{k} = \breve{\mathbf{C}}\hat{\mathbf{x}}_{\mathbf{k|k}}$
```

of 0.272. Note that the complexity of the algorithm in Alg. 6.2 is determined by the rank computation of the controllability matrix $\mathbf{Q_c}$ which is $\mathcal{O}(n^3)$. The rank computation uses singular value decomposition (SVD) which has the computational complexity of $\mathcal{O}(n^3)$. To further validate and verify the completely observability of the system, we construct a Kalman filter using this single sensor observation as well as placement and track the peak temperature (hotspot) of the processor. We compare the results of the thermal hotspot tracking of the Alpha processor with that of a state-of-the-art thermal and hotspot tracking method [285].

6.5.3 SKF for State Prediction

To construct a full-state observer, we propose the use of the Kalman filtering approach using sparse measurement as discussed earlier. The thermal dynamics of the processor is modeled using the discrete linear state-space system [147, 126] in presence of variability induced process noise [54] as in Eq. 6.9. By using Alg. 6.4, the thermal

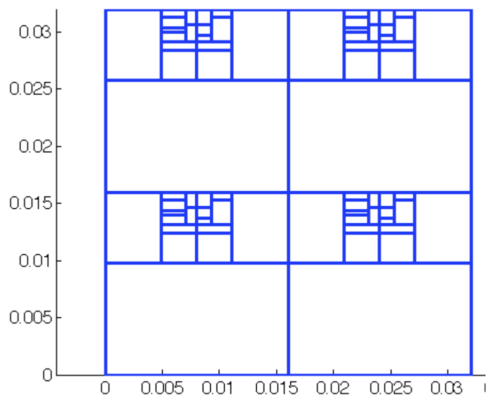


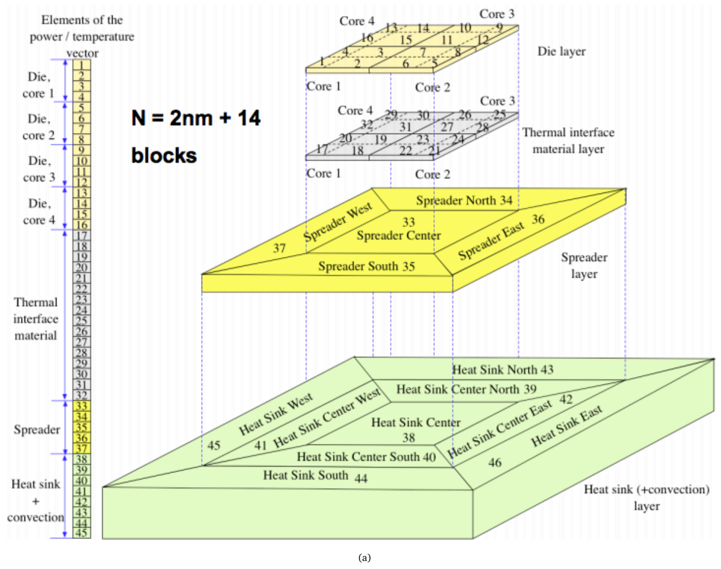
Figure 6.2: Floor plan of quad-core processor based on Alpha 31386.

and power dynamic behavior of the MPSoC system is estimated as well as predicted ahead of time in the $k+1$ epoch.

6.5.4 Run-time Thermal Awareness and Hotspot Tracking

Temperature adversely affects the power and reliability of processor systems. For safe and reliable operation, the peak temperature of the processor has to be always maintained or controlled below a safe threshold. However, with the change in workloads and phasic behavior of workloads, the power consumption of each block varies vastly. As temperature sensors along with their peripheral circuits introduce substantial overhead in silicon area and power consumption, it is extremely important to minimize the number of temperature sensors without surrendering the accuracy of thermal monitoring. On the one-hand a large number of temperature sensors are needed for accurate thermal monitoring in presence of large dynamic power variations, but on the other hand they incur substantial die area real estate and power consumption overhead. Given this scenario, the minimal sparse observability problem directly addresses this trade-off by providing the minimum number of sensors to accurately observe the thermal dynamics.

In order to experimentally verify the results, we have created a simulation framework as shown in Fig. 6.5. Fig. 6.6 shows the thermal profile estimation of the complete processor using a sparse Kalman filter presented in Alg. 6.4 with just one sensor. The thermal network for the single-core Alpha processor is observable using a single sensor and the thermal profile estimated by the SKF is very accurate. Even in presence of sensor noise, the SKF provides the best statistically possible estimation and tracking of the thermal profile. The approach presented can easily be applied to multi-core configurations; the number of sensors and their location are obtained using the algorithm in Alg. 6.2. Fig. 6.7 shows improved tracking of the hotspot in the Alpha processor



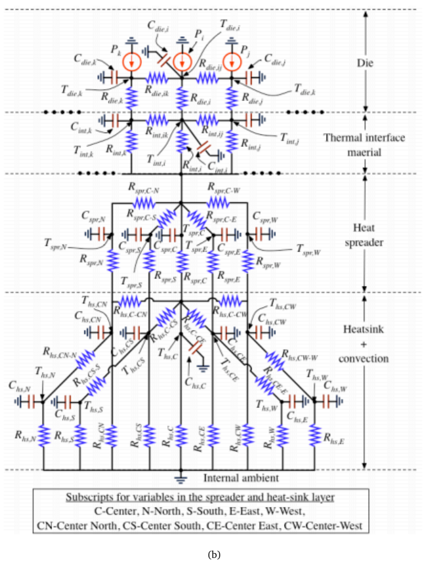

Die
Thermal interface maerial
Heat spreader
Heatsink + convection
Internal ambient
Subscripts for variables in the spreader and heat-sink layer
C-Center, N-North, S-South, E-East, W-West,
CN-Center North, CS-Center South, CE-Center East, CW-Center-West
(b)

Figure 6.3: Thermal network representation of high performance processors. (a) blocks in the different layers of the chip and their corresponding nodes in the thermal networks [335] (b) RC equivalent circuit representation of the thermal dynamic network [128].

in comparison to a state-of-the-art method [285] for all the SPEC 2000 benchmarks. As should be evident from Fig. 6.7, the minimum sensor SKF provides superior hotspot tracking that is paramount for the reliable operations of processors.

In addition, the SKF can also alleviate the variability induced process noise in deep sub-micron technologies as well as suppress sensor noise by suitably filtering the noise during the estimation steps. Fig. 6.8 shows the filtering of the sensor noise from the measurements during the estimation procedure. Such an approach could also be beneficial in the closed loop dynamic thermal management and performance improvement of the processor as illustrated in Fig. 6.9. During the dynamic thermal management operation, the processor throttles its frequency (i.e., reduces the frequency of operation and hence power by certain percentage, e.g., 20%) whenever the peak temperature crosses a specified threshold. In the absence of a accurate hotspot tracker, the peak thresholds are violated resulting in deep reliability issues and imminent damage to the processor. On the other hand, the noise in the sensor can spuriously trigger early DTM operation causing unwanted performance loss. Fig. 6.9 illustrates the scenario where the SKF enabled the DTM to filter-out the noise and spurious triggers, resulting in performance improvement of approximately 28% in this specific example scenario.

6.5.5 Overheads and Complexity

The algorithmic solution to the MSOP has the computational complexity determined by the SVD computations. The complexity of finding the sparsest measurement matrix through the SVD based rank computations results in $\mathcal{O}(n^3)$. On the other hand, the complexity of the SKF that uses the sparse measurement matrix, is also $\mathcal{O}(n^3)$. This is determined by the matrix inverse involved in the computation of the Kalman gain $\mathbf{K_k}$. In a scenario where the time scale at which the noise characteristics change is much larger than the time scale at which the thermal networks are studied (e.g., month or even years), the system and the noise covariance of the Kalman filter can be assumed to be time-invariant [330]. As a result, we can use a steady state sparse Kalman filter for which it is not necessary to compute the estimation error covariance or the Kalman gains in real-time [330], rather the gains can be replaced by constant gains. This reduces the computational overhead of the real-time thermal estimation using the SKF from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$ while still providing good accuracy. However, a calibration step may be needed prior to running the SKF with steady state gains, this can be achieved by computing the Kalman gains initially from the SKF and then switching to steady state SKF with constant gains. Furthermore, a distributed implementation [185, 254]of the SKF can be used to reduce the complexity further for scalable architecture.

6.5.6 Virtual Run-time Power Sensing Using Thermal Sensors

Here we illustrate extended use case of the state estimation approach discussed above for unknown inputs for accurate run-time power and temperature estimation and prediction using noisy thermal sensors placed at the suitable locations in the CPSoC fabric.

This scenario illustrates the virtual sensing capability of CPSoC in jointly estimating and predicting the run-time transient power consumption and accurate temperature of each subsystem unit as well as accurate temperatures from noisy temperature sensors readings using the observe-decide-act (ODA) paradigm. The virtual approach (Fig. 6.10(a)) combines a sensor-network-on-chip (sNoC) and a robust Kalman filter [145] to enable in-situ, on-the-fly run-time estimation of the power and temperature of each block/unit from on-chip noisy thermal measurements. Unlike traditional MPSoCs, the specialized sensor network (Fig. 6.10(b)) can coexist independently or combined with the core-to-core communication network (cNoC) as shown in Fig. 6.10(c). Such an architecture decouples the sensing and computation concerns and allows flexibility to independently perform on-chip sensing without interfering or burdening the core-to-core computation network.

Temperature values at different locations on the die depend on various factors such as power consumptions of functional units, layout of the chip and the package characteristics. The differential equations describing the heat flow have a form dual to that of electrical current, represented using lumped values of thermal R and Cs, and form the basis for commonly used micro-architectural thermal models in state space form [334][147].

Virtual sensing of the run-time transient power requires the thermal model which represents the relation between subsystem power and the temperature as discussed in Section 6.5.1. However, to address the emerging problem of semiconductor process variations in deep sub-micron technologies and sensor noise [388], the processor thermal dynamics is augmented with the process and measurement noise in the standard LTI model as:

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{A}_k\mathbf{x}_k + \mathbf{E}_k\mathbf{d}_k + \mathbf{w}_k \\ \mathbf{y}_k &= \mathbf{H}_k\mathbf{x}_k + \eta_k \end{aligned} \tag{6.11}$$

where $\mathbf{x_k} \in \mathbb{R}^\mathbf{n}$ are the system states (i.e., each block's temperature), $\mathbf{d_k} \in \mathbb{R}^\mathbf{p}$ are the unknown inputs (i.e., each block's power), $\mathbf{y_k} \in \mathbb{R}^\mathbf{m}$ are the measurements (i.e., the temperature sensor measurement), $\mathbf{A}_k$,$\mathbf{E}_k$, $\mathbf{H_k}$ are system matrices of appropriate dimensions. The process noise $\mathbf{w}_k$ and measurement noise $\eta_\mathbf{k}$ are assumed to be mutually uncorrelated, zero-mean, white random signals with known covariance matrices, $\mathbf{Q_k} = \mathbf{E}\left[\mathbf{w_k}\mathbf{w_k^T}\right]$ and $\mathbf{R_k} = \mathbf{E}\left[\eta_\mathbf{k}\eta_\mathbf{k}^\mathbf{T}\right]$ respectively.

To effectively construct the virtual sensing mechanism for the variability and noise corrupted dynamic system in Eq. (6.11), we use an optimal Kalman filter to provide robust estimates. The objective of the Kalman filter is to jointly estimate the unknown power input $\mathbf{d_k}$ as well as the temperature states $\mathbf{x_k}$ from noisy temperature measurements in the presence of both process and measurement noise as described in Alg. 6.5. Robust Kalman filters [145] have been shown to provide unbiased estimates of the states and unknown inputs and guarantee global optimality and minimum variance of the estimates [145]. Consequently, virtual sensors constructed using such filters will provide the statistically best solution for the thermal and power estimation of the subsystems.

To validate our approach we created a CPSoC simulation platform called CPSoCSim [305]. We used the Alpha 31386 processor and its compilation tools that have been extensively used in earlier research work [334, 147]. We use the SPEC and PARSEC benchmarks and their corresponding power traces to generate the thermal profile using the Hotspot simulator [147]. We assume that the temperature of the blocks are measured with noisy sensors and are collected by the sNoC so that these can be processed to produce thermal and the power estimates at each subsystem unit of the processor. Fig. 6.11 shows the actual temperature obtained from Hotspot [147] versus the temperature obtained from the virtual sensor. The estimation of the power of each units and run-time tracking of total power consumed by the processor are shown in Fig. 6.12(a) and Fig. (b) respectively. Although the estimation error is a function of process and measurement noise, for typical scenarios the temperature and power estimate errors are less than 1% and 5% respectively. Note that no power sensors were used in the whole process of power estimation of the subsystems; results were generated indirectly by computational means using virtual sensing.

6.6 Related Work

The conditions of observability and controllability of LTI systems were initially introduced by Kalman [171] and have been used extensively in control theory. Although the classical rank condition [171, 224] proposed by Kalman provides a test for checking the controllability and observability for given system matrices, the process of systematically finding these system matrices (measurement and control matrices) have not been addressed. The very recent groundbreaking work of Liu et al. [224] addressed the process of making a complex system completely controllable using few controlling or driving nodes. The approach finds the minimal number of driving nodes (or controllers) that would be necessary for driving a complex network to a specific state by using a graphical approach. Specifically, they developed a minimum input theory to efficiently characterize the structural controllability of directed networks, allowing a minimum set of driver nodes to be identified to achieve full control. In particular, the structural controllability of a directed network can be mapped into the problem of maximum matching [144, 386], where external control is necessary for every unmatched node. Liu et al. [225] extended their graphical approach to observability of complex networks in their very recent work [225] .

Although the graphical approach based on structural controllability theory offers a general tool for *directed* networks, the approach fails if the assumption of independence of free link parameters and non-symmetry of the structural matrices is violated [224]. In other words, the graphical approach can not be applied to any arbitrary complex network with structure and configurations of the link weights where the parameters (i.e., the elements of the systems matrices) are not independently varying. Specifically, for *undirected* networks, the symmetric characteristic of the network matrix accounts for the violation of the assumption of the structural matrix, even with random weights [224]. To overcome this limitation, recently a more generic algorithmic approach was proposed in [255]. Our work is motivated by the work in [255] but differs in its objective and problem formulation. The work in [255] finds the sparsest control matrix $\mathbf{B}$ for which a complex network is fully controllable whereas we find the sparsest measurement matrix $\mathbf{C}$ and minimum number of sensors as well as their locations for which the complex dynamical network system is completely observable. Our work is closest to the very recent work of Lie et al. [225] but differs in two respect. First, [225] considers a graphical approach (GA) based on structural properties of system matrices to make the system structurally observable, whereas we pursue a generic algorithmic approach to make the system completely observable. Second, the approach in [225] can not be applied to any arbitrary system with structural symmetries, whereas no such limitations hampers our approach and thus our approach can be applied to any arbitrary system.

In our approach, we also outline the design of a sparse Kalman filter to illustrate full-state observability of complex thermal networks of real high performance processors and multiprocessor system-on-chip (MPSoC).

6.7 Summary

One of the most challenging problems in emerging MPSoC is that of controlling and observing the complex dynamical behavior of power-thermal networks. Observability is fundamental to having deeper insight and self-awareness in any complex dynamic behavior of MPSoC. It is paramount in the understanding of the interplay between the complex network topology and the underlying dynamic behavior. In this work, we explored this fundamental question of observing the internal dynamics of a complex dynamic network systems using minimal set of observation points by using the notion of sparsity. We defined the minimal sparse observability problem (MSOP) to find the sparsest measurement vector and proved that the problem is NP-hard. Our main result is the development of an algorithm to find the sparsest measurement matrix that will make any arbitrary linear dynamical network completely observable. We developed effective greedy algorithms to find the sparsest measurement and used it in the design of a sparse Kalman filter (SKF) to estimate time-dependent internal states of complex dynamical networks. We applied the approach to complex thermal networks of real processor systems and demonstrated the applicability in run-time thermal profile prediction and hotspot tracking using a minimal number of on-chip sensors for effective dynamic thermal management of such processors. We illustrated an extended use case of the state estimation approach for unknown inputs for accurate run-time power and temperature estimation and prediction using noisy thermal sensors placed at the suitable locations in the CPSoC fabric.

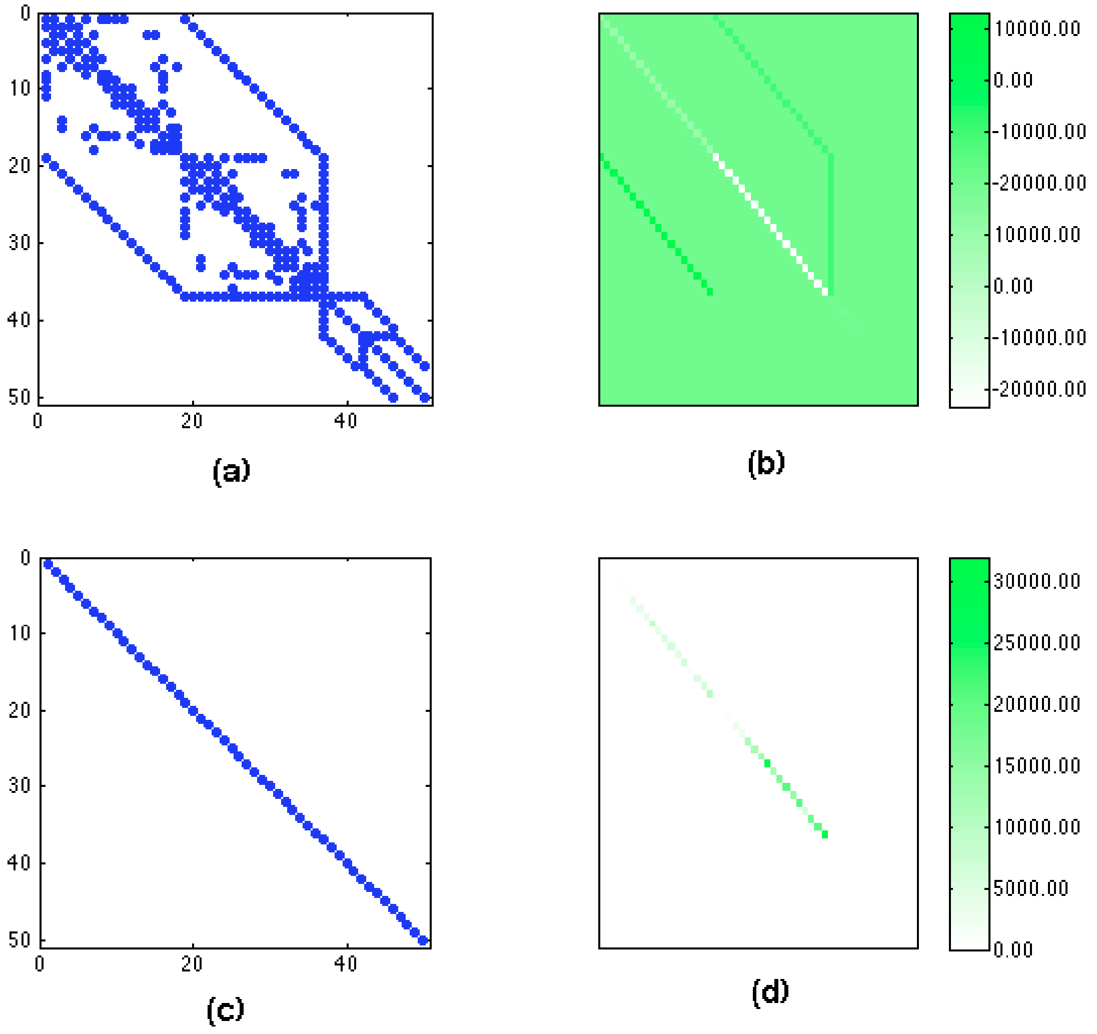


Figure 6.4: System matrices of the micro-architectural thermal model of Alpha 31386 processor. (a) sparsity pattern of the the system coupling matrix **A** (size 50×50) in continuous domain (b) representation of the coupling matrix **A** (size 50×50) in the discrete domain (c) sparsity pattern of the control matrix **B** (size 50×50) in continuous domain (d) the **B** matrix (size 50×50) in discrete domain. The matrices are obtained from the Hotspot thermal simulator [147] for the Alpha 31386 processor floorplan.

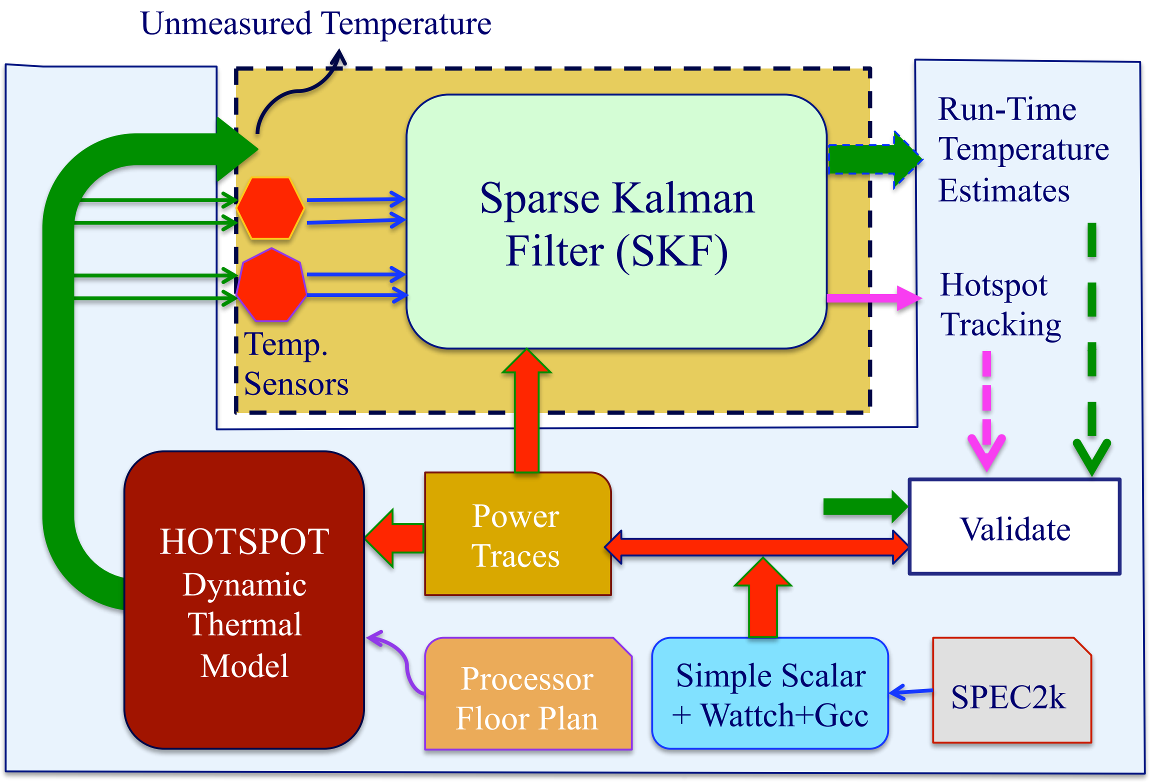

Unmeasured Temperature
Sparse Kalman Filter (SKF)
Run-Time Temperature Estimates
Hotspot Tracking
Temp. Sensors
HOTSPOT Dynamic Thermal Model
Power Traces
Validate
Processor Floor Plan
Simple Scalar + Wattch+Gcc
SPEC2k

Figure 6.5: Simulation and validation framework for run-time thermal estimation using Sparse Kalman Filter (SKF). The SKF estimates the full chip thermal profile using minimum number of sensors while filtering the effect of sensor, measurement, and process noise.

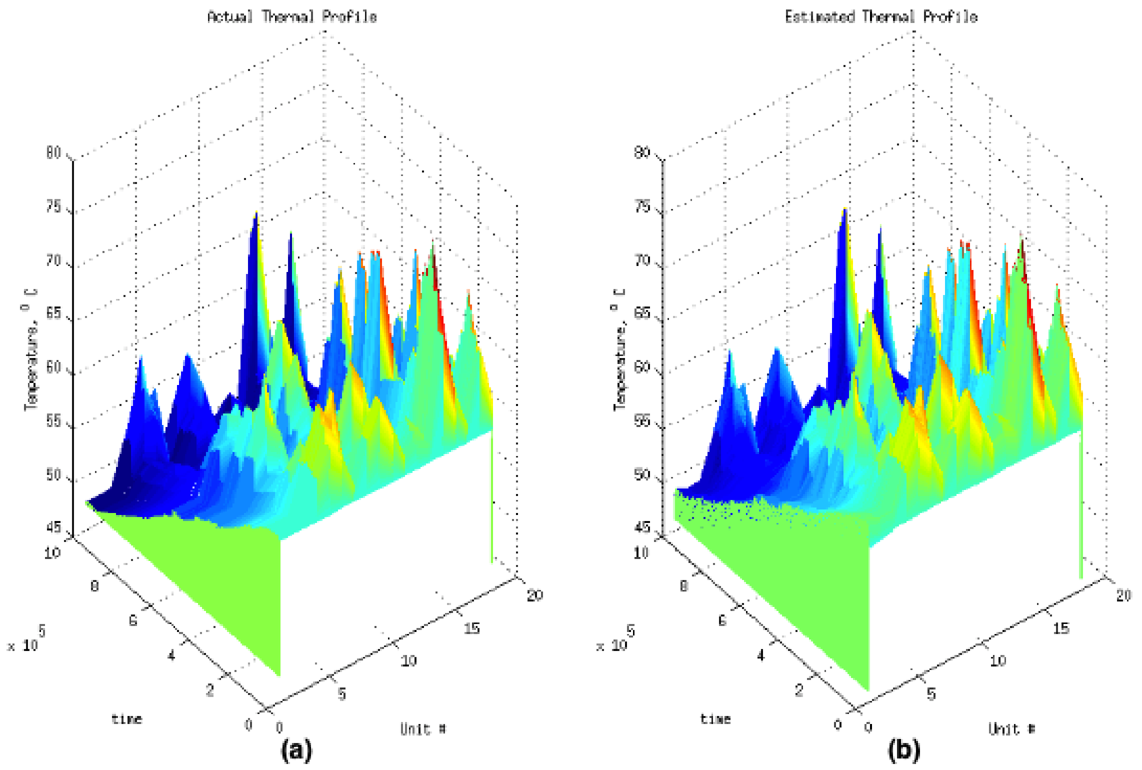


Figure 6.6: Thermal profile estimation (a) actual thermal profile (b) estimated by SKF using a single sensor for all the SPEC 2000 benchmarks. The estimated error is with in 0.3% for all the blocks.

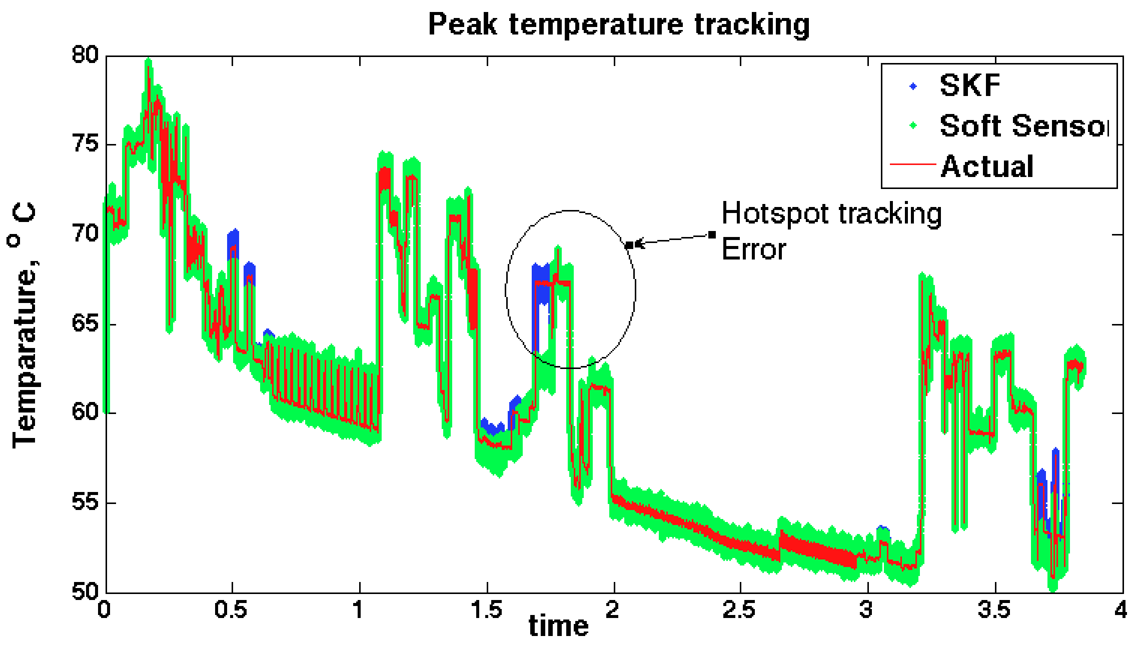


Figure 6.7: Robust hotspot tracking of the Alpha processor using SKF in comparison to the state-of-the-art hotspot tracking approach in [285].

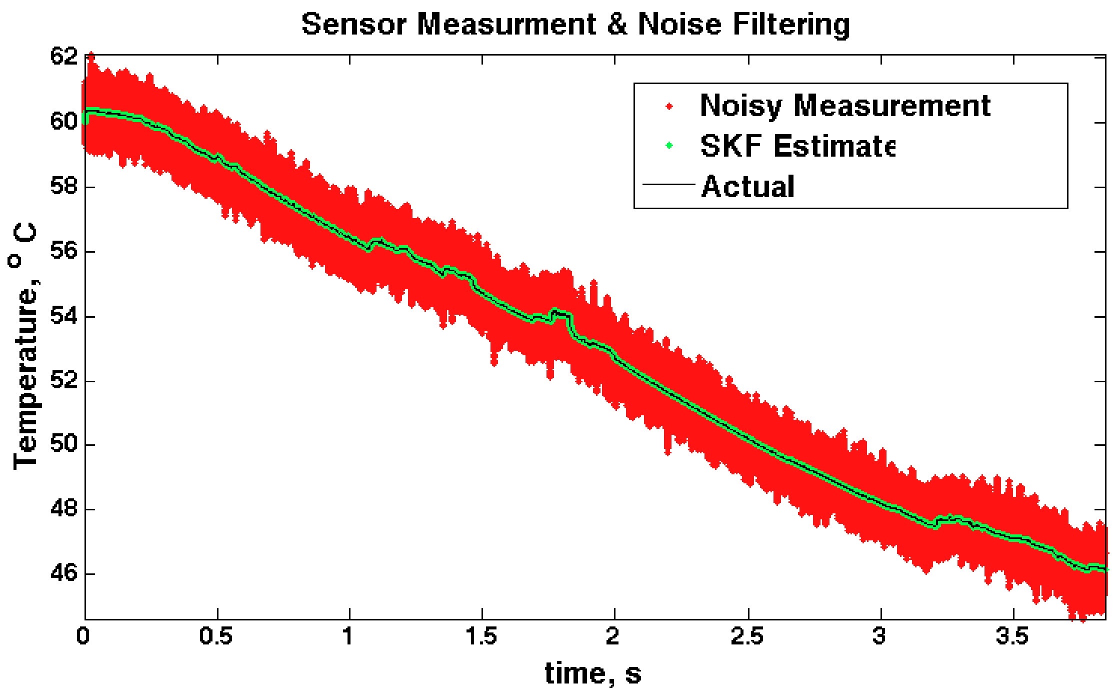
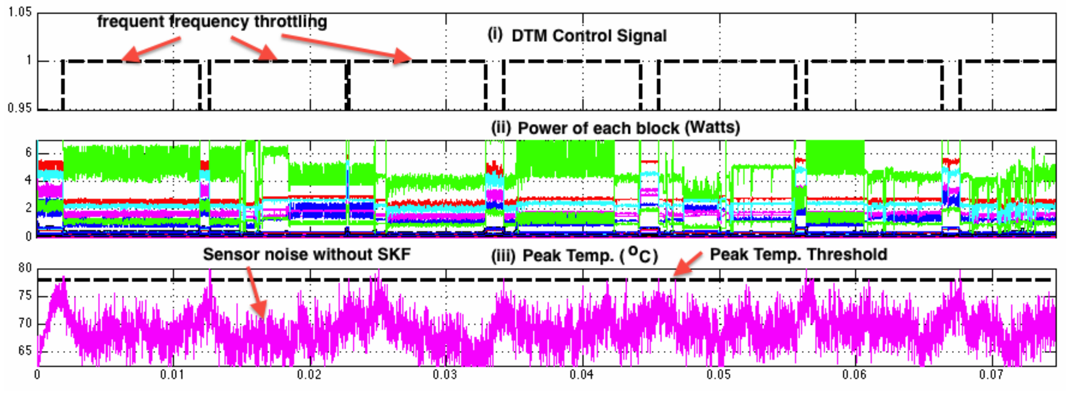
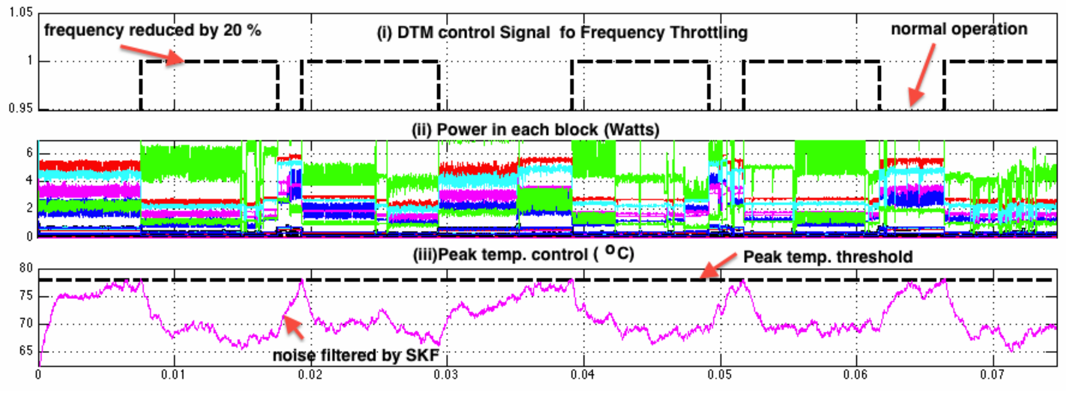


Figure 6.8: Sensor measurement (with noise variance of $\pm 1^o C$) and noise filtering with SKF. Effect of both measurement noise and variability induced process noise can be mitigated by the SKF to achieve statistically superior estimates.



(a)



(b)

Figure 6.9: DTM control of the Alpha processor with minimal sensor placement (a) DTM with direct noisy sensor reading (b) with SKF. Because of the noise filtering by the SKF, less frequent frequency throttling is initiated in the DTM, which improves performance of the processor system (approx. by 28%).

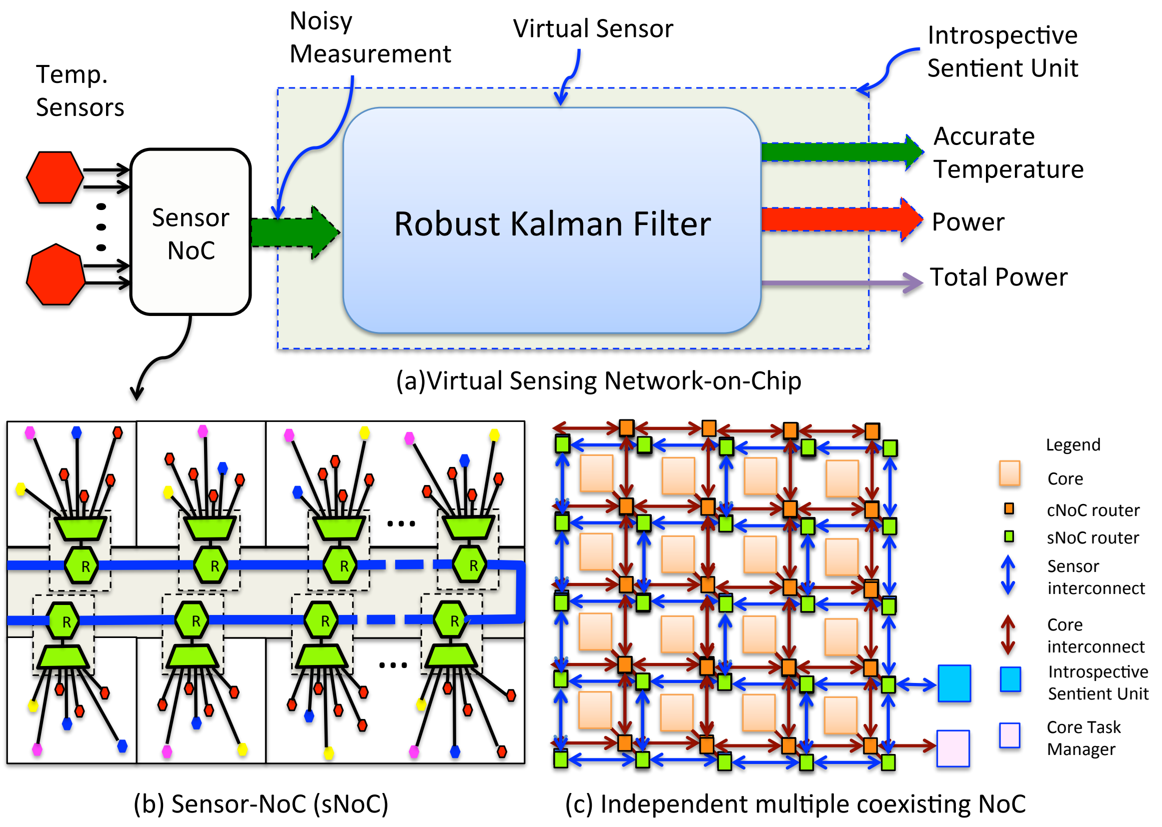


Figure 6.10: (a)Virtual Sensor Network-on-Chip Architecture (b) Sensor NoC (sNoC) (c) Independent multiple coexisting NoCs in CPSoC.

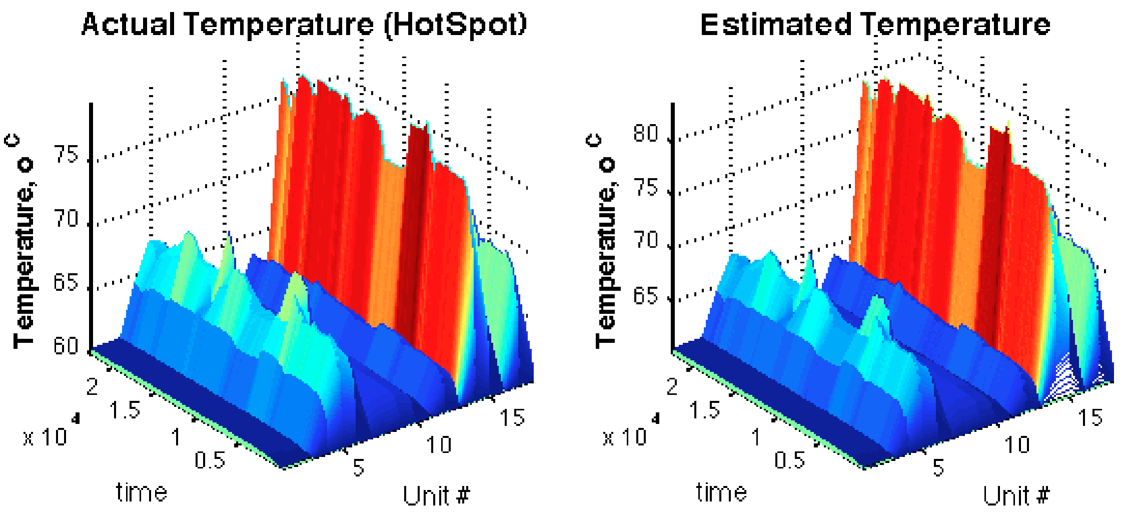


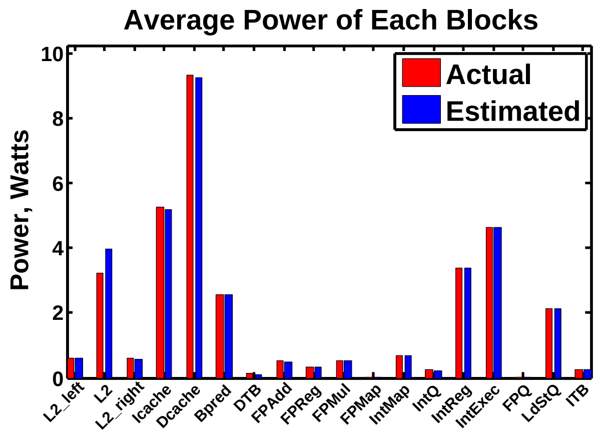
Figure 6.11: Run-time subsystem temperature estimation.

Algorithm 6.5 Virtual Sensing of Subsystem Power and Temperature

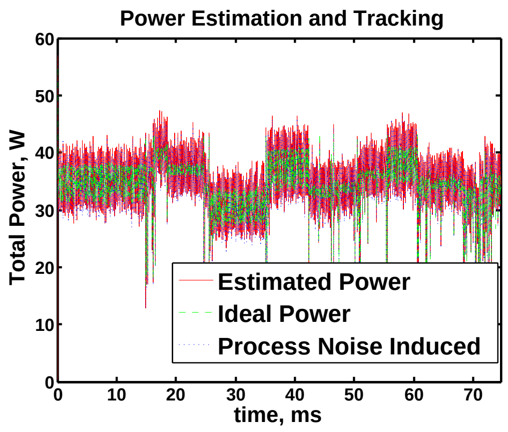
Virtual Sensing of Core and Subsystem Temperature and Power.
Input: Temperature sensor measurement from sensors $\mathbf{y}$, Thermal state space model $\mathbf{A_k}, \mathbf{E_k}, \mathbf{H_k}, \mathbf{Q_k}, \mathbf{R_k}$
Output: Subsystem Thermal Profiles, $\mathbf{\hat{x}_k}$, and Power Profile, $\mathbf{\hat{d}}_k$, Total Power, P_{total}

Perform the following every sampling step:

```
1. Initialize the values of $\mathbf{\hat{x}_k}$,$\mathbf{\hat{P}_k}$, $\mathbf{A_k}, \mathbf{E_k}, \mathbf{H_k}, \mathbf{Q_k}, \mathbf{R_k}$
2. Robust Two-Stage Kalman Filter (RTSKF):
    (a) $\mathbf{\bar{x}_{k|k-1}} = \mathbf{A_k \bar{x}_{k|k-1}}$
    (b) $\mathbf{P^{\bar{x}}_{k|k-1}} = \mathbf{A}_{k-1}\mathbf{\hat{P}_{k-1|k-1} A^T_{k-1}} + \mathbf{Q_{k-1}}$
    (c) $\mathbf{C_k} = \mathbf{H_k P^{\bar{x}}_{k|k-1} H^T_k} + \mathbf{R_k}$
    (d) $\mathbf{P^d_{k|k}} = \left\{\mathbf{E^T_{k-1} H^T_k C^{-1}_k H_k E_{k-1}}\right\}^{-1}$
    (e) $\mathbf{K^d_k} = \mathbf{P^d_{k|k} E^T_{k-1} H^T_k C^{-1}_k}$
    (f) $\mathbf{K^{\bar{x}}_k} = \mathbf{P^{\bar{x}}_{k|k-1} H^T_k C^{-1}_k}$
    (g) $\mathbf{V_k} = (\mathbf{I} - \mathbf{K^{\bar{x}}_k H_k})\mathbf{E_{k-1}}$, $\mathbf{I}$ is identity matrix
    (h) $\mathbf{\hat{d}_{k|k}} = \mathbf{K^d_k}\left(\mathbf{y_k} - \mathbf{H_k \bar{x}_{k|k-1}}\right)$
    (i) $\mathbf{\bar{x}_{k|k}} = \mathbf{\bar{x}_{k|k-1}} + \mathbf{K^{\bar{x}}_k}\left(\mathbf{y_k} - \mathbf{H_k \bar{x}_{k|k-1}}\right)$
    (j) $\mathbf{\hat{P}^x_{k|k}} = \mathbf{P^{\bar{x}}_{k|k}} + \mathbf{V_k P^d_{k|k} V^T_k}$
    (k) $\mathbf{P^{\bar{x}}_{k|k}} = (\mathbf{I} - \mathbf{K^{\bar{x}}_k H_k})\mathbf{P^{\bar{x}}_{k|k-1}}$
    (l) $\mathbf{\hat{x}_{k|k}} = \mathbf{\bar{x}_{k|k-1}} + \mathbf{V_k d_{k|k}}$
3. Output the estimates: Power Estimates , $\widehat{\mathbf{d}}$; Total Power, $P_{total}$; Temperature Estimates, $\widehat{\mathbf{x}}$
```



(a)



(b)

Figure 6.12: (a) Power estimation of subsystem units (b) Run-time total power estimation and tracking.

Chapter 7

Operating System Support for Adaptation in Emerging MPSoCs

The word “adaptation” has several meanings in the systems community. However, the most accepted notion of adaptation is defined as the ability to adjust and improve system response using feedback, the ability to respond to change along with the capability to alter itself dynamically to achieve goals. From a software engineering perspective, a system is adaptive when it allows for modifying its structure or behavior at runtime; i.e., without interrupting its service by coupling the system with an adaptation manager. On the other hand, from a control engineering viewpoint, adaptation or adaptive control adds a degree of flexibility to the control mechanism, where the controller may change its own control policies, structure, or control parameters dynamically.

In this work we reconcile both these viewpoints by defining a generalized closed-loop structure where the system monitors and detects the changes, analyzes their impact, and, if needed, plans and executes actions in response to the changes. In addition, knowledge about the system that is captured by suitable models updated at runtime is used to support adaptation. We specifically focus on the operating system support needed for adaptation of emerging heterogeneous MPSoCs for achieving system goals (e.g., energy efficiency) dynamically.

Due to increased demand for higher performance and better energy efficiency, MPSoCs are deploying heterogeneous architectures with architecturally differentiated core types. However, the traditional Linux-based operating system is unable to exploit this heterogeneity since existing kernel load balancing and scheduling approaches lack support for aggressively heterogeneous architectural configurations (e.g., beyond two core types). In this chapter, we present SmartBalance: a sensing-driven closed-loop load balancer for aggressively heterogeneous MPSoCs that performs load balancing using a sense-predict-balance paradigm. SmartBalance can efficiently manage the chip resources while opportunistically exploiting the workload variations and performance-power trade-offs of different core types. The approach builds predictive machine learning models of performance, power, and energy for capturing online behavior of different cores considering the cross-layer (software and hardware) characteristics. The approach contributes toward the design and extension of the Linux operating system with self-awareness abstractions, adaptive middleware, with predictive models to achieve system level goals such as energy efficiency. The experimental results show that exposing and adapting the system software and OS with quantitative and predictive awareness of power and throughput at the thread level can improve energy efficiency by over 50% with respect to the existing Linux Kernel load balancer and over 20% with respect to ARM’s GTS scheme. When compared to the standard vanilla Linux kernel load balancer, our per-thread and per-core performance-power-aware scheme shows an improvement in energy efficiency (throughput/Watt) of over 50% for benchmarks from the PARSEC [48, 47] benchmark suite executing on a heterogeneous MPSoC with 4 different core types and over 20% with respect to the state-of-the-art ARM’s global task scheduling (GTS) [16] scheme for octa-core big.Little architecture.

7.1 Smart Balancing : An Operating System Adaptation Mechanism

Emerging embedded devices (e.g., mobile platforms) face diverse multi-threaded workloads along with conflicting needs of high energy efficiency and performance, necessitating a move towards heterogeneous MPSoC platforms with architecturally differentiated cores providing attractive power-performance benefits [53] [120] [252]. Architectures with different core types are already a reality (e.g., NVidia's Kal-El [252] and ARM's big.LITTLE [120]) and this trend towards heterogeneity is only expected to grow further in the future [53, 223]. Unfortunately, handling heterogeneity comes at the cost of a complex OS scheduler and load balancer. Both time-varying characteristics of the application and OS workloads as well as the heterogeneous features of the platform need to be smartly managed. Existing OS kernels' scheduling and load balancing schemes are openly accepted to be inefficient for such systems [121]. For instance, the vanilla Linux kernel load balancer evenly distributes the workload among cores even if the cores have distinct processing capabilities, which can result in serious performance and energy efficiency loss. Even though there have been some recent efforts to address this important issue (e.g., the IKS [272] and the GTS [16] Linux extensions), the solutions have been limited to the very specific case of ARM's big.LITTLE with two core types. This limitation not only discourages hardware vendors from introducing new architectural improvements with diverse core types, but also defeats the purpose of versatile OS support for heterogeneous MPSoC with seamless OS configurations without making major structural changes in the OS load balancer/scheduler.

In this chapter, we propose SmartBalance, a closed-loop sensing-driven opportunistic load balancer that uses on-chip sensing, estimation and prediction, and global optimization for aggressively heterogeneous MPSoCs (with more than two core types, for instance, with Big (A15), Medium (A11), and Little (A7) core types) as depicted in Fig. 7.1 (b). SmartBalance consists of three phases: **sense**, **predict**, and **balance** that are executed at runtime in periodic *epochs*, where each epoch covers multiple Linux scheduling periods as shown in Fig. 7.1(c). Unlike the open-loop standard Linux load balancer seen in Fig. 7.1 (a) which distributes the thread evenly, our closed-loop feedback-driven approach makes judicious decisions to distribute the threads smartly (i.e., matched to the core type) so as to best achieve the system goal(s) (e.g., energy efficiency) in a smart way. SmartBalance leverages existing OS code bases by reusing and refactoring the legacy Linux kernel code to improve energy efficiency. The key contributions of our approach are:

- We present SmartBalance: a closed-loop load balancing approach under a sense-predict-balance paradigm that efficiently manages chip resources while opportunistically exploiting diverse workload and core performance-power characteristics.

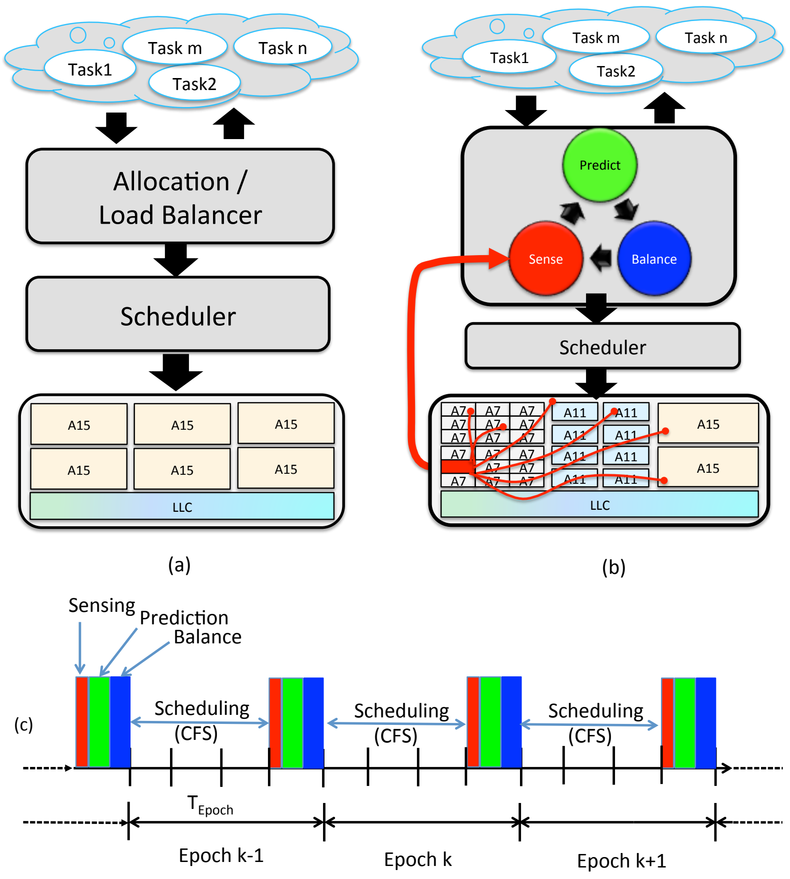


Figure 7.1: Load balancing with (a) standard Linux for homogeneous MPSoCs, (b) SmartBalance closed-loop sense-predict-balance approach for aggressively heterogeneous MPSoCs, (c) timing relations of the phases in SmartBalance. Each epoch covers several linux CFS scheduling periods.

- We expose workload (performance, power) variability to the OS to exploit per-thread workload characteristics (IPC, power, and utilization) in each core for aggressively heterogeneous architectures (as defined in Chapter 2). We introduce estimation and prediction models to calculate the performance and power impact of executing each thread on different heterogeneous cores without performing sampling at each core type as depicted in Fig. 7.2.

- We demonstrate a working prototype and make available the reference implementation for the Linux 2.6.x kernel as an open source project at: https://github.com/santanusarma/smartbalance.

- When compared to the standard vanilla Linux kernel load balancer, SmartBalance improves energy efficiency by over 50% for PARSEC benchmarks executing on a Heterogeneous MPSoC with 4 core types and by over 20% with respect to the

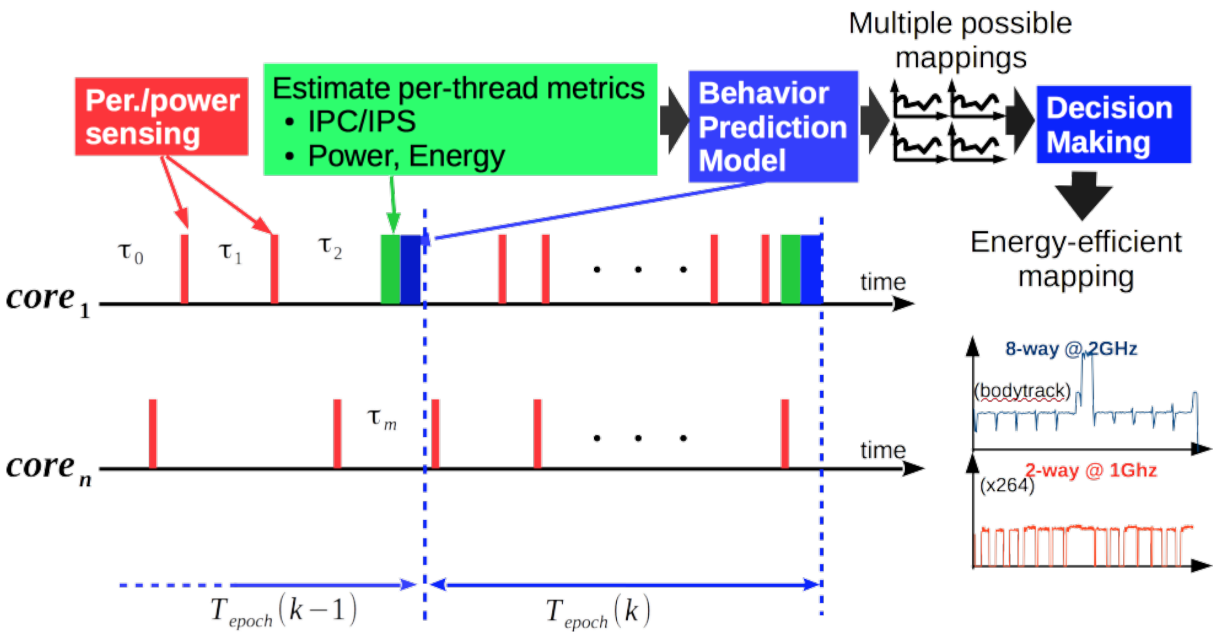


Figure 7.2: Temporal model of the SmartBalance approach.

state-of-the-art ARM GTS scheme [16].

7.2 Motivation and Related Work

Heterogeneous MPSoCs provide architecturally diverse cores with drastically different power-performance trade-offs that can be exploited by the OS. Consequently, heterogeneous MPSoCs and their run-time systems is an active area of research. Previous works focused on extracting computational performance with energy efficiency as a secondary benefit [36, 360] and recently the focus has shifted to energy efficiency [16, 189, 70, 223]. However, most of these works have been restricted to a special class of heterogeneous MPSoC architecture (e.g., homogeneous cluster of two limited core types as in ARM big.LITTLE [163, 252]) with the number of threads being scheduled limited by the number of cores [36, 201, 70, 360, 223, 15]. Within this restricted architectural model, many of them do not take into account the OS and associated issues in their simulation/analysis[201, 36, 223, 15, 70, 360]. Many of these techniques also lack thread-level awareness of both the performance and power characteristics for exploiting multithreaded workloads at a finer granularity[201, 360, 70, 223, 15, 240, 189, 272]. This limits the opportunity to perform extensive energy and performance optimizations. SmartBalance overcomes these restrictions by handling aggressive architectural heterogeneity and exploiting runtime performance-power variability. The key differences of SmartBalance with the existing state-of-the-art is summarized in Table 7.1.

The closest to our work are the approaches proposed in [189, 272, 16, 15, 240, 223].

Table 7.1: Comparative Summary of Related Work.

Reference	Scheme Generality		Per-Thread Awareness		Per-Core Awareness			Integrated & Implemented in OS
	No Core Types >2	Thread-to-core-ratio >1	IPC	Power	Util.	IPC	Power	
Chen et al., 2009[70]	✓	×	×	×	×	✓	✓	×
Annamalai et al., 2013 [15]	×	×	×	×	×	✓	✓	×
Liu et al., 2013[223]	✓	✓	×	×	×	✓	✓	×
Kim et al., 2014 [189]	×	✓	×	×	✓	×	×	✓
Poirier et al., 2013 [272]	×	✓	×	×	✓	×	×	✓
ARM GTS, 2013[16]	×	✓	×	×	✓	×	×	✓
SmartBalance [313]	✓	✓	✓	✓	✓	✓	✓	✓

The work by Kim et al. [189] improves Linaro IKS [272] by bringing the core utilization awareness to the scheduler but lacks support for architectures with more than two core types in addition to per thread awareness for finer balancing. The Global Task Scheduler (GTS) [16] proposed by ARM brings several improvements in comparison to Linaro IKS[272] both in terms of performance and generality. GTS allows a selection of either a big core or a little core instead of a core cluster as in IKS, thus enabling architecture exploitation at finer granularity. However, GTS can not directly support architectures with more than two core types without major re-engineering of the kernel. In addition, although GTS considers per-thread utilization awareness, it is unable to exploit thread level opportunity for instruction level parallelism (ILP) and the power consumption of the threads. Annamalai et al. [15] proposed a program phase-aware dynamic scheduling scheme limited to only two core types without considering the OS workload and implementation issues. Liu et al [223] proposed a dynamic thread scheduling scheme with the number of threads limited to the number of cores. They also do not provide per thread awareness for the load balancer to exploit and consider the OS implementation issues. Mogul et al. [240] considered OS issues and proposes an asymmetric architecture with special core types optimized to run OS code with improved energy efficiency, but are also limited to two core types in their scheduling and load balancing. In contrast with all of the above related work in Table 7.1, SmartBalance considers aggressively heterogeneous architectures with more than two core types while exploiting per-thread and per-core level awareness in a closed loop manner.

7.3 Heterogeneous Computing Elements and Thread Model

We consider the case of heterogeneous multi-core systems in which all cores can have different capabilities. A core type r is defined by the combination of micro-architectural features and their nominal performance and power (voltage/frequency). For example, Table 7.2 shows four core types labeled as *Huge, Big, Medium*, and *Small* differing in seven architectural feature combinations such as issue width, instruction queue (IQ) size, reorder buffer (ROB) size, number of registers, and cache size. In addition, even if the cores are identical in terms of microarchitecture but associated with different nominal frequencies, they can be considered as distinct core types. We define the set of cores as $C = \{c_1, c_2, ..., c_n\}$, the set of core types as $R = \{r_1, r_2, ..., r_q\}$, where $\gamma : C \rightarrow R$ gives type of a particular core. Each type r is characterized by a unique combination of the parameters $X = \{x_1, x_2, ..., x_p\}$, with an example shown in Table 7.2.

Table 7.2: Heterogeneous Core Configuration Parameters

Parameter	**Symbol**	**Huge Core**	**Big Core**	**Medium Core**	**Small Core**
Issue width	x_1	8	4	2	1
LQ/SQ size	x_2	32/32	16/16	8/8	8/8
IQ size	x_3	64	32	16	16
ROB size	x_4	192	128	64	64
Int/float Regs	x_5	256	128	64	64
L1$I size (KB)	x_6	64	32	16	16
L1$D size (KB)	x_7	64	32	16	16
Freq. (MHz)	F	2000	1500	1000	500
Voltage (V)*	$\underline{V_{DD}}$	1	0.8	0.7	0.6
Peak Throughput*	IPC	4.18	2.60	1.31	0.91
Peak Power (W)*	P_{Total}	8.62	1.41	0.53	0.095
Area (mm^2)*	A	11.99	5.08	3.04	2.27

* Estimated by the Gem5 [49] and McPAT modeling framework [218, 219] for a 22nm node based on PARSEC [48, 47] benchmarks.

SmartBalance assumes each core can run multiple threads and perform multitasking. We assume processes are encapsulating threads similar to the Pthread model, so there is no formal or explicit dependency between threads. Within the Linux scheduling subsystem, processes and threads are all treated as *task entities* and scheduled independently. For uniformity, in this work the term *thread* is used interchangeably for both single-threaded processes and for threads of the same process. We consider a total of m threads to be mapped to n cores without restricting the number of threads to the number of cores. Threads can enter and leave the system at any time and their total execution time is unknown. The set of threads to be optimized contains all threads active at the beginning of each SmartBalance Epoch.

Throughout the work, we use the following notations:

- $V = \{t_i, 1 \leq i \leq m\}$ is the set of all the threads to be allocated during an epoch. The thread set $\Psi_j(k) = \{t_i, 1 \leq i \leq m_j\}, \forall t_i \in V = \{t_1, t_2, ..., t_m\}$ is m_j threads mapped to core c_j at the k^{th} epoch such that the "allocation"

$$\mathbf{\Psi(k)} = \{\Psi_j(k), 1 \leq j \leq n\} \tag{7.1}$$

where total threads $m = \sum_{j=1}^{n} m_j$.

- We define the throughput characterization matrix that is exposed to the OS as:

$$\mathbf{S}(\mathbf{k}) = [\mathbf{s_i}(\mathbf{k})] = [ips_{ij}(k)] \tag{7.2}$$

  as the average throughput (in instruction per second (IPS)) of threads executing on different cores at k^{th} epoch. $\mathbf{s_i}(\mathbf{k}) = \{ips_{ij}(k), 1 \leq j \leq n\}$ represents the vector throughput when executing thread t_i on different cores. $ips_{ij}(k)$ represents the average throughput of thread t_i executing on core c_j. $IPS_j(k)$ is the average throughput of core c_j when executing its allocated threads during the epoch.

- Similarly, we define a power characterization matrix that is exposed to the OS as:

$$\mathbf{P}(\mathbf{k}) = [\mathbf{p_i}(\mathbf{k})] = [p_{ij}(k)] \tag{7.3}$$

  as the average power consumption of threads executing on different cores during an epoch k. $\mathbf{p}_i(k) = \{p_{ij}(k), 1 \leq j \leq n\}$ represents a power vector corresponding to a thread t_i executing on different cores and $p_{ij}(k)$ represents the average power of thread t_i executing on core c_j at the k^{th} epoch. $P_j(k)$ is the total average power consumed by the core c_j during the epoch while executing the threads allocated to it.

7.4 SmartBalance Approach

SmartBalance is closed-loop load balancing approach consisting of three stages: **sensing**, **prediction**, and **balancing** that are executed at runtime in periodic epochs, where each smart balancing epoch covers multiple Linux CFS (completely fair scheduling) periods as shown in Fig. 7.3. Since we use the standard Linux CFS to perform scheduling of the threads allocated to the same core, we only focus on the three stages at the beginning of each epoch. At every epoch, SmartBalance first samples the power and the performance counters of each core. Second, using the measurements, it estimates the individual impact or contribution of each thread towards the performance and power of its current core. Third, the individual per-thread contribution in performance and power are used to predict the per-thread performance and power in other core types. These per-thread performance and power characteristics are represented in a systematic manner using two 2-dimensional matrices. These matrices are then used to find the allocation for the next epoch in the fourth step. Once the allocation is found, the threads are scheduled by the OS using the standard Linux CFS in the last step. This process is repeated in each epoch as shown in Fig. 7.3.

7.4.1 Sensing

During the sensing and measurement phase, *hardware performance counters (HPCs)* and power are periodically sampled per thread (at the thread context switch as shown

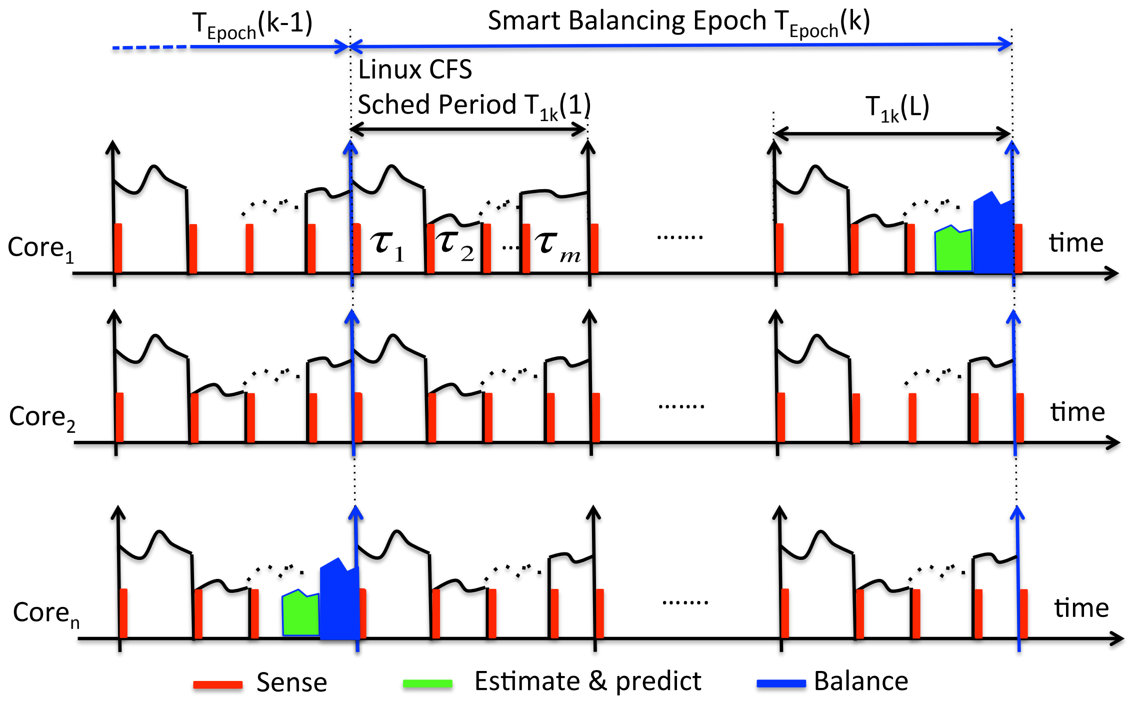


Figure 7.3: SmartBalancer epochs for workload-aware dynamic thread balancing and scheduling in heterogeneous MPSoCs. Each SmartBalance epoch covers L Linux CFS scheduling periods.

in Fig. 7.3) in order to trace the workload characteristics of the threads currently running in the system. We currently use following hardware performance counters (cycle, instruction and performance degradation events):

- **Cycle counters** sample the amount of *busy cycles* (cy_{Busy}), *idle cycles* (cy_{Idle}), and *sleep cycles* (cy_{Sleep}) of a core. Busy cycles represent the time a core spends doing computation. Idle cycles capture idling time due to pipeline stalls or cache misses. Sleep cycles capture the time a core spends in a quiescent state. In our current experimental platform and kernel implementation, a core enters this state when it has no threads to execute.

- **Instruction counters**. We sample the total amount of *committed instructions* (I_{total}), *committed load and stores* (I_{mem}), and *committed branches* (I_{branch}). These are used to calculate the share of memory instructions $I_{msh} = \frac{I_{mem}}{I_{total}}$ and the share of branch instructions $I_{bsh} = \frac{I_{branch}}{I_{total}}$.

- **Performance events counters.** We measure the following events which are known to drive the performance of a core[15]: mispredicted branches, which is used to compute the branch misprediction rate (mr_b); instruction/data L1 cache and TLB misses and hits, which are used to compute the L1 instruction miss rate ($mr_{\$i}$), L1 data cache miss rate ($mr_{\$d}$), instruction TLB miss rate (mr_{itlb}), and data TLB miss rate (mr_{dtlb}).

7.4.2 Prediction of Performance and Power

As described in Section 7.3 threads are characterized using the performance and power matrices. Estimation and prediction of the performance and power matrices are possible since there is a direct correlation between the behavior of different core types. In this section, we derive analytical models to compute these matrices using a combination of measurements and prediction among the cores, and the experimental validation of each step is presented in Section 7.6. We propose a simple methodology to predict $ips_{il}(k)$ in a different core c_l by using the measurement $IPS_j(k)$ (instructions per second) and its throughput contribution $ips_{ij}(k)$ in the core c_j, $\forall j \neq l$ as discussed below.

7.4.2.1 Performance and Power Estimation of Each Thread on a Core

In order to estimate the thread-specific performance and power characteristics, we define the following notations and timing relations as shown in Fig. 7.3. We define the load balancing epoch T_{Epoch} consisting of L CFS scheduling periods T_{jk}. Let $T_{jk}(l), l = 1..L$ be the l^{th} CFS scheduling period in the k^{th} epoch $T_{Epoch}(k)$. Let there be $m = \sum_{j=1}^{n} m_j$ threads, i.e., $\Psi_j = \{t_i, 1 \leq i \leq m_j\}, \forall t_i \in V = \{t_1, t_2, ..., t_m\}$ scheduled in a scheduling period $T_{jk}(l)$ in the core c_j. Let i^{th} thread get time share τ_{ijl} in the l^{th} scheduling period such that the scheduling period and smart balancing epoch is given by $T_{Epoch}(k) = \sum_{l=1}^{L} T_{jk}(l)$ where $T_{jk}(l) = \sum_{i=1}^{m_j} \tau_{ijl}$. Let I_{ijl} be the measured counter values for the number of instructions executed by the i^{th} thread in the l^{th} scheduling period, $\dddot{ips}_{ijl}$, $\dddot{p}_{ijl}$, and $\dddot{\epsilon}_{ijl}$ be the respective measured throughput, power, and energy consumed by the thread during the execution duration τ_{ijl} is given by $ips_{ijl} = {}^{I_{ijl}}/_{\tau_{ijl}}$ and $\dddot{p}_{ijl} = {}^{\dddot{\epsilon}_{ijl}}/_{\tau_{ijl}}$ respectively. The average total throughput and power in the k^{th} epoch for the i^{th} thread is:

$$ips_{ij}(k) = {}^{\sum_{l=1}^{L} I_{ijl}}/_{\sum_{l=1}^{L} \tau_{ijl}} = \frac{1}{L} \sum_{l=1}^{L} \dddot{ips}_{ijl} \tag{7.4}$$

$$p_{ij}(k) = {}^{\sum_{l=1}^{L} \dddot{\epsilon}_{ijl}}/_{\sum_{l=1}^{L} \tau_{ijl}} = \frac{1}{L} \sum_{l=1}^{L} \dddot{p}_{ijl}. \tag{7.5}$$

Total average throughput IPS_j and power P_j in the k^{th} epoch for all the threads for the core c_j:

$$IPS_j(k) = \sum_{i=1}^{m_j} \left(\sum_{l=1}^{L} I_{ijl} / \sum_{l=1}^{L} \tau_{ijl} \right) = \frac{1}{m_j} \sum_{i=1}^{m_j} ips_{ij}(k) \tag{7.6}$$

$$P_j(k) = \frac{1}{m_j} \sum_{i=1}^{m_j} p_{ij}(k) \tag{7.7}$$

Average core performance in terms of instruction per second for the k^{th} epoch is $IPS_j(k) = IPC_j(k) * F_j = {I^j_{total} * F_j}/{(cy^j_{Busy} + cy^j_{Idle})}$ where cy^j_{Busy}, cy^j_{Idle}, and I^j_{total} are the counters values sampled for core c_j at epoch k, and F_j is the frequency of the core.

7.4.2.2 Performance and Power Prediction for Different Core Types

In the previous section, we described how we compute $ips_{ij}(k)$ for all threads t_i executed on core c_j in the k^{th} epoch. The next step updates the remaining values of the $\mathbf{S}(k)$ matrix with predictions of performance for different core types. Some approaches use a sampling-based method[36], in which every thread is periodically executed on all core types in order to collect performance and power statistics. This imposes a high overhead in the system. Instead, we employ a prediction-based approach. The key idea behind the prediction of the throughput matrix relies on the fact that the average throughput behavior of a thread on one core is correlatable to the throughput on another core (with same ISA and memory hierarchy) with a good degree of accuracy [360]. By collecting performance information of the thread in one core we can predict the throughput in the other cores. In Section 7.4.1, we identified a set of performance counters that can be used to characterize the workload being executed by a thread. Given the aforementioned set of counters, the performance of a thread t_i that has run on core c_j at the the k^{th} epoch can be predicted on every other core c_l, $\forall j \neq l$ as follows:

$$\widehat{ipc}_{ij}(k) = \mathbf{\Theta} * X^T_{ij} \tag{7.8}$$

such that $\widehat{ips}_{ij}(k) = \widehat{ipc}_{ij}(k) * F_j$ where $X^T_{ij} = [x^{ij}_1, x^{ij}_2, ..., x^{ij}_N]^T$ represents a characterization vector of the workload of thread t_i during epoch k-1. $\mathbf{\Theta}$ is the coefficient matrix which defines impact of each metric when predicting from core c_j of type $\gamma(c_j)$ to core c_l of type $\gamma(c_l)$. In order to obtain Θ, we employ standard linear regression using the least square method in a similar approach to [15].

The final step of the estimate/predict phase of SmartBalance is to obtain the power characterization matrix $\mathbf{P}(k)$. We use the observation that the power p_{ij} a thread t_i is linearly correlated to its ipc_{ij} [220]. We use linear interpolation to relate the power consumption of the thread t_i running a core c_j of type $y = \gamma(c_j)$ to the predicted ipc_{ij} as:

$$\widehat{p}_{ij} = \alpha_1 * \widehat{ipc}_{ij} + \alpha_0 = \mathbf{\Theta_1} * X^T_{ij} + \alpha_0 \tag{7.9}$$

where α_0, α_1 are constants that provide the performance–power relationship for core type y and are obtained from offline profiling.

7.4.3 Thread Balancing and Allocation

The thread mapping problem consists of finding an optimal allocation of threads $V = \{t_i, 1 \leq i \leq m\}$ on the cores $C = \{c_1, c_2, ..., c_n\}$ of each type $R = \{r_1, r_2, ..., r_q\}$ such that an objective or cost function is optimized. We call an assignment of all threads $V = \{t_1, t_2, ..., t_m\}$ to available cores $C = \{c_1, c_2, ..., c_n\}$ an *allocation* $\mathbf{\Psi}(\mathbf{k})$. An allocation as defined in Eq. (7.1) results in an objective or cost function to be optimized while accounting for the heterogeneity of processing elements and executing threads. An objective or a cost function for the allocation problem can be defined in several ways according to the desired optimization goals. In this work, we focus on **maximizing overall energy efficiency** (i.e., IPS/Watt or Instructions per Joule) as defined below:

$$\underset{\boldsymbol{\Psi(k)}}{Maximize} \quad (J_E) \tag{7.10}$$

$$J_E = \textstyle\sum_{j=1}^{n} \omega_j \frac{IPS_j(k)}{P_j(k)} \tag{7.11}$$

where the objective J_E is the weighted sum of energy efficiency (IPS/watt) of all the cores, $IPS_j(k)$ can be calculated using Eq. (7.6) and $P_j(k)$ is given by Eq.(7.7) or the characterization matrices in Eq. (7.2) and Eq. (7.3). The weights ω_j are ideally set to 1, but can be tuned to give preference to certain cores or core types.

7.4.3.1 Optimization Methodology

Finding the optimal thread allocation is an NP-hard combinatorial problem [93] that requires heuristics exploiting specific characteristics of the problem to achieve acceptable solutions within a reasonable amount of time. In our SmartBalance context, we have a further constraint of achieving online optimization at the start of each SmartBalance epoch (Fig. 7.3). We achieve this through the runtime optimization outlined in Alg. 7.1 that deploys a modified online Simulated Annealing (SA)-based approach. SA has been shown to produce nearly-optimal global solutions while accommodating problem-specific changes without significant modifications [93]. Furthermore, SA provides tunable parameters to trade-off computational complexity for solution quality (e.g., number of iterations and precision of probabilistic functions), enabling a runtime light-weight implementation. The tunable input parameters in Alg. 7.1 can be used to trade-off computational complexity for solution quality (e.g., number of iterations and convergence speed). While a straightforward floating-point implementation of Alg. 7.1 may lead to long execution times due to the high cost of computing the probabilistic functions, we use custom fixed-point implementations of $rand$ and e^x that trade-off performance with uniformity ($rand$) and precision (e^x) without significantly compromising the quality of the final solution. The computation of the objective function is also optimized by keeping track of previous computations and obtaining a new evaluation only by performing computations induced by the latest swap on Ψ. In Section 7.6.3 we show that our runtime optimization algorithm is able to find solutions online (at the beginning of each scheduling epoch) without imposing significant overheads on the operating system.

Algorithm 7.1 Smart_Balance() Runtime Optimization

Input Parameter Options: Max. no. of iterations Opt_{max_iter}, perturbation schedule $Opt_{\triangle perturb}$, acceptance rate schedule $Opt_{\triangle accept}$, initial perturbation $Opt_{perturb}$ and acceptance rate Opt_{accept}

Input Data: Throughput Matrix $\mathbf{S}$, Power Matrix $\mathbf{P}$ (core dynamic and leakage power vectors P_D and P_L), thread utilization vector $\mathbf{U}$, initial allocation $\mathbf{\Psi}_0$

Output: Allocation $\mathbf{\Psi}$

```
1:  $\mathbf{\Psi} \leftarrow \Psi_0$                         ▷ Implemented as uni-dimensional array
2:  $iteration \leftarrow Opt_{max\_iter}$
3:  $perturb \leftarrow Opt_{perturb}$
4:  $accept \leftarrow Opt_{accept}$
5:  while $iteration > 0$ do
6:      $\mathbf{\Psi}' \leftarrow \mathbf{\Psi}$
7:      $pos \leftarrow randi(0, n * m)$                ▷ permute for new index position
8:      $pos\_new \leftarrow pos + \sqrt{perturb} * randi(-pos, n * m - pos)$
9:      swap($\mathbf{\Psi}'$, $pos$, $pos\_new$)
10:     $diff \leftarrow J_E(\mathbf{S}, \mathbf{P}, \mathbf{\Psi}') - J_E(\mathbf{S}, \mathbf{P}, \mathbf{\Psi})$
11:     if $diff > 0$ then          ▷ Always accept if new solution is better than previous
12:         $\mathbf{\Psi} \leftarrow \mathbf{\Psi}'$
13:     else             ▷ Accept worse solution with a probability proportional to $accept$
14:         $probability \leftarrow e^{\frac{-diff}{accept}}$
15:         if $randi() \mod \frac{1}{probability} = 0$ then
16:             $\mathbf{\Psi} \leftarrow \mathbf{\Psi}'$
17:         end if
18:     end if
19:     $iteration \leftarrow iteration - 1$           ▷ Update perturbation and acceptance rate
20:     $perturb \leftarrow perturb * Opt_{\Delta perturb}$
21:     $accept \leftarrow accept * Opt_{\Delta accept}$
22: end while
    ▷ $randi()$ generates an uniformly distributed integer number in the interval $[0, 2^{32})$,
    while $randi(x, y)$ generates a number in the interval $[x, y)$
```

7.5 Experimental Setup and Implementation

We created an experimental simulation platform (shown in Fig. 7.4) comprising the heterogeneous cores described in Table 7.2 using the *Gem5* performance simulator [49]. *Gem5* includes cycle-accurate models for various CPU architectures, as well as peripheral models that allow us to run full system simulations. We use *Gem5* to create heterogeneous cores by changing the architectural simulation parameters of the original Alpha 21264 superscalar architecture. Our approach is not limited by the voltage and frequency of the cores, however, to show the effect of architectural heterogeneity, we fix all cores' voltages and frequencies to predefined values. All L1 and L2 caches are private and the cores are connected to the main memory through a shared bus. For obtaining power data, we integrated the *McPAT* power model [218, 219] directly with the *Gem5* simulation framework, which allows us to obtain power sensor data at runtime. *Gem5* is also extended with a *sensing interface* which exports *McPAT* power information and other *Gem5* statistics (specifically the hardware counters) to the kernel at run-time.

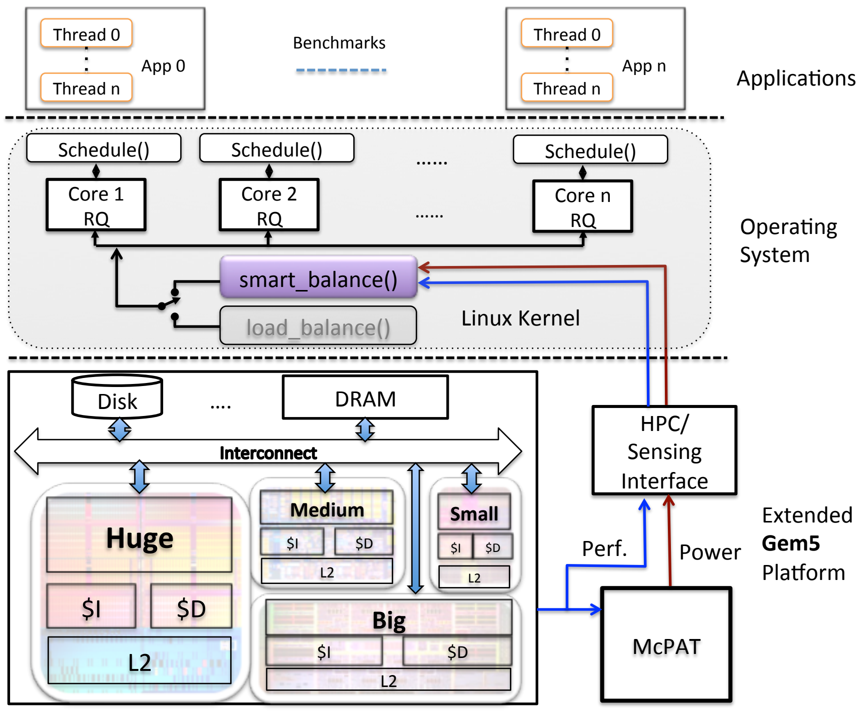


Figure 7.4: SmartBalance experimental platform using extended Gem5.

7.5.1 SmartBalance Implementation

SmartBalance replaces Linux's existing load balancing mechanism. In the vanilla kernel, load balancing is triggered by the function `rebalance_domains()`. We have reimplemented this function in order to call SmartBalance instead of the standard load balancing at the end of every epoch as shown in Fig. 7.3. We sample performance counters on a thread-by-thread basis. The sampling of these counters is done at the granularity of Linux's *schedule()* function. At the beginning of each epoch, the values of all thread-counters are collected and the performance and power matrices are built for use in the subsequent phases. Values that are unavailable are predicted as described in Subsection 4.3.1. It is worth mentioning that SmartBalance can optimize both the user and kernel threads jointly. Since the impact of the user level threads dominates that of the kernel threads, we focus on the user-level threads by identifying and marking them during their creation in the *sched_fork()* function. We assume that each user thread is allowed to run on any core, however, special constraints can easily be included by modifying the objective function in Eq. (7.11). Once a new allocation $\mathbf{\Psi}$ is found, threads may be migrated to balance the system according to the new allocation. The migration process is performed using the *set_cpus_allowed_ptr()* function already provided by the kernel.

Table 7.3: Benchmarks and their Mixes.

Benchmarks	**Mix1**	**Mix2**	**Mix3**	**Mix4**	**Mix5**	**Mix6**
PARSEC Mixes	x264$_{\mathrm{H}}$crew x264$_{\mathrm{H}}$bow	x264$_{\mathrm{L}}$crew x264$_{L}$bow	x264$_{\mathrm{L}}$crew x264$_{\mathrm{H}}$bow	x264$_{\mathrm{H}}$crew x264$_{L}$bow	Bodytrack x264$_{\mathrm{H}}$crew	Bodytrack x264$_{\mathrm{H}}$crew x264$_{\mathrm{L}}$bow

7.6 Experimental and Evaluation Results

The objective of our experiments is to evaluate the energy efficiency and effectiveness of the SmartBalance Linux kernel with respect to the baseline vanilla Linux kernel and the state-of-the-art ARM GTS policy [16] using a varied sets of benchmarks and their mixes. Our first set of experiments show results for generic HMP architectures with four core types, while our second set of results in Section 7.6.1 performs a comparison with the state-of-the-art ARM GTS policy for big.Little architectures with two core types. In order to do a comprehensive evaluation of SmartBalance, we use PARSEC benchmarks [48, 47] and their combinations for different levels of parallelization (2,4,8 threads) as shown in Table 7.3. We select a set of multithreaded benchmarks that have diverse characteristics (compute and memory intensive) from the PARSEC [48] benchmark suite as representative applications as well as create sets of synthetic benchmarks with attributes that reflect interactive/IO dependent applications.

We call these sets of multithreaded synthetic benchmarks *interactive microbenchmarks* (IMB) that provide the ability to control the load, phasic behavior, and interactivity (sleep and wait periods). The IMBs can be configured to have throughput (T) and interactivity (I) that controls the sleep/wait periods for high (H), medium(M), and low(L) values. As an example, HTHI corresponds to the high throughput and high interactiveness of the IMB configuration and all the other 8 combinations are similarly labeled in our experiments. Note that we use the x264 benchmark with different configurations (high(H)/low(L) frame processing rate) and input videos (crew/bowing) to show that a single benchmark can have different characteristics (both in IPS and power) as shown in Table 7.3.

Fig. 7.5 shows the energy efficiency of the SmartBalance approach for different sets of benchmarks when the benchmarks are run using the *Gem5* experimental platform in Fig. 7.4(a) in full system mode in a cluster computing environment. We observe that the SmartBalance kernel performs 50.02 % on average better with the interactive benchmarks (Fig. 7.5(a)), 52% with the PARSEC benchmarks (Fig. 7.5(b)) and their mixes respectively when running 2, 4, and 8 threads of each benchmarks. Overall, SmartBalance Linux kernel achieves an energy efficiency of over 50% across all the benchmarks in comparison to the vanilla Linux kernel.

7.6.1 Comparison with state-of-the-art

SmartBalance provides a simple yet tunable runtime optimization engine that overcomes the limitations of balancing scheme generality for emerging heterogeneous multicore processors (HMPs) with several core types. In order to demonstrate the relative advantage of the SmartBalance approach even for specific HMP architectures such as ARM big.Little [163] with two core types, we compare our approach with that of the state-of-the-art ARM's global task scheduling policy[16]. We create an octa-core big.Little HMP using *Gem5* and modify the Linux 2.26.x kernel to implement the ARM GTS policy. Note that the GTS policy works only for big.Little type of HMP as the policy makes a fixed utilization threshold-based binary decision to either select *a big* or *a little* core [16]. In fact, the lack of joint per-thread (finer granularity) and per-core accurate power as well as performance awareness, limits GTS from achieving energy efficiency by as much as $\sim$20% in comparison to SmartBalance for several benchmarks as shown in Fig. 7.6. On the contrary, instead of using core utilization as a proxy for deciding thread allocation for energy efficiency, SmartBalance uses accurate energy efficiency *measurements directly* –both at fine and coarse granularity during thread balancing to provide additional improvements while providing a generic and fairly scalable solution as shown in Section 7.6.3.

7.6.2 Predictor Evaluation

The runtime prediction of performance and power incurs an average error of 4.2 % and 5 % respectively as shown in Fig. 7.8 and Fig. 7.7 corresponding to the predictor coefficient matrix $\mathbf{\Theta}$ in Table 7.4. The aver age percentage error is about 70% smaller than the one reported by [15], which proposes a similar predictor. The predictor in [15]

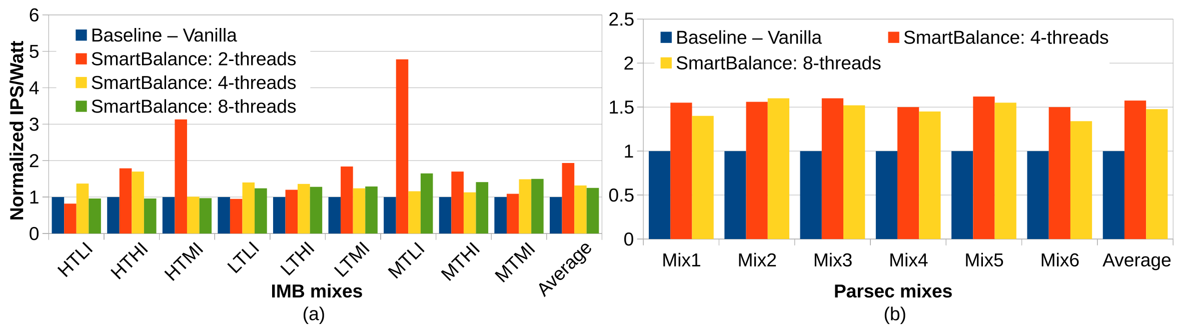


Figure 7.5: SmartBalance kernel performance with respect to the baseline vanilla Linux kernel (running the same benchmarks with same number of threads) using (a) interactive microbenchmarks (b) PARSEC benchmark and their mixes. The optimization goal is set to maximize overall energy efficiency (IPS/Watt).

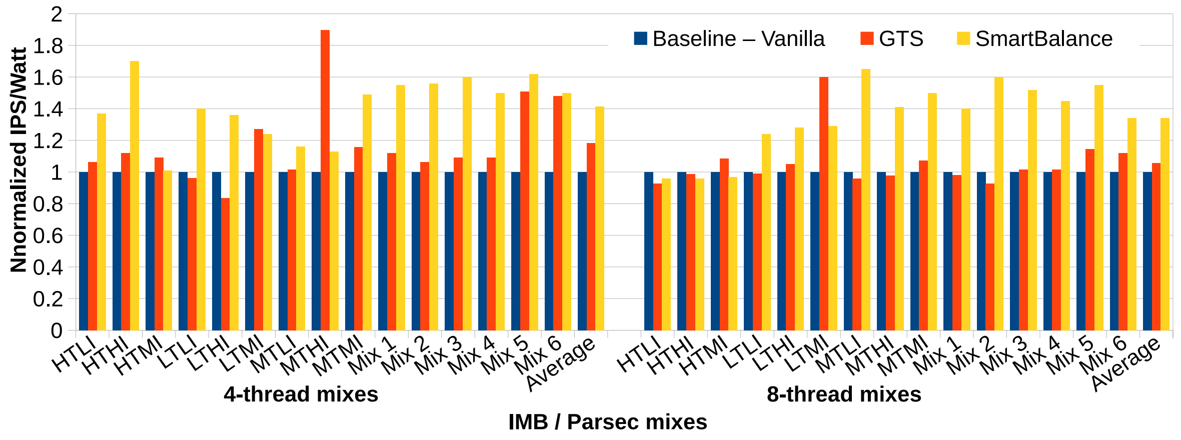


Figure 7.6: Comparison for normalized energy efficiency with respect to the state-of-the-art.

doesn't include the share of memory and branching instructions in its expression, which might explain the higher average error. The work of [223] has also employed a similar performance predictor based on the offline profiling of PARSEC benchmarks. However, differently from our work, [223] implements a binning approach in which each thread is categorized as their nearest neighbor in the PARSEC benchmarks. Compared to [223], our approach yields 71% smaller average error.

Table 7.4: Predictor coefficient matrix.

Predictor IPC	FR*	$mr_{\$i}$	$mr_{\$d}$	I_{msh}	I_{bsh}	mr_b	mr_{itlb}	mr_{dtlb}	$ipc_{i,src}$	$const$
Huge->Big	-0.006	-2.110	-0.135	-0.975	1.213	0.385	-2.928	-9.552	0.416	0.691
Huge->Medium	-0.007	-3.610	-0.106	-0.751	0.906	0.841	-0.881	-9.661	0.130	0.668
Huge->Small	-0.005	1.786	1.139	-2.017	0.125	0.035	4.958	-8.913	0.142	0.657
Big->Huge	-0.007	1.870	0.393	1.787	-2.213	-1.134	0.000	5.717	2.084	-1.017
Big->Medium	-0.003	-2.286	-0.057	-0.416	0.455	1.351	0.000	-5.173	0.326	0.415
Big->Small	-0.004	-0.046	1.394	-1.235	0.607	1.060	0.000	-23.62	0.194	0.432
Medium->Huge	-0.053	3.064	-0.385	3.882	-3.987	-6.458	-249.4	7.210	4.503	-1.835
Medium->Big	-0.016	2.145	0.025	1.401	-1.460	-1.739	-6.416	9.666	2.731	-0.987
Medium->Small	-0.008	0.995	1.080	-0.258	0.105	2.719	208.5	3.290	1.040	-0.467
Small->Huge	0.042	-1.835	-9.291	6.021	-1.238	-19.81	0.000	104.00	2.745	0.235
Small->Big	0.003	-0.630	-4.087	1.185	-0.277	-9.902	0.000	39.354	1.545	0.829
Small->Medium	-0.001	-1.035	-1.122	-0.517	0.357	-2.639	0.000	2.639	0.594	0.776

*FR if the ratio of the frequencies of source to target core

7.6.3 Overheads and Scalability

Fig. 7.9 shows average runtime measurements for each SmartBalance phase on our 4-core platform as well as extrapolated results for systems scaling from 2 to 128 cores with 4 to 128 threads. Most of the overhead originates from the optimization algorithm and thread migration (assuming 50% of the threads are migrated). However, for typical embedded platforms (e.g., quad-core mobile devices) with 2 to 8 cores, the average overhead of using SmartBalance is small (less than 1%) with respect to the 100ms balancing epoch length. Recall that we used the runtime light-weight SA-based optimization described in Section 7.4.3, which enables us to control the overhead. As shown in Fig. 7.10(a), for larger configurations we limit the number of iterations to avoid excessive overhead, therefore trading off solution quality for scalability.

7.7 Future work

One limitation of the SmartBalance approach may be argued to be the dependence on additional counters and sensors for fine-grained awareness of performance and power. We used as many as 10 counters and per-core power sensors for predictions. Although this may be viewed as a serious limitation on certain architectures, current trends and future projections suggest the inclusion of per-core power sensors, many counters, and on-chip monitors already in several existing platforms [215, 266]. For example, the recent Samsung Exynos big.Little board [266] already has power sensors for each core, GPU, and DRAM as well as extensive support for all the counters in the Linux kernel using oprofile [84]. In addition, our previous work [306, 341]– where

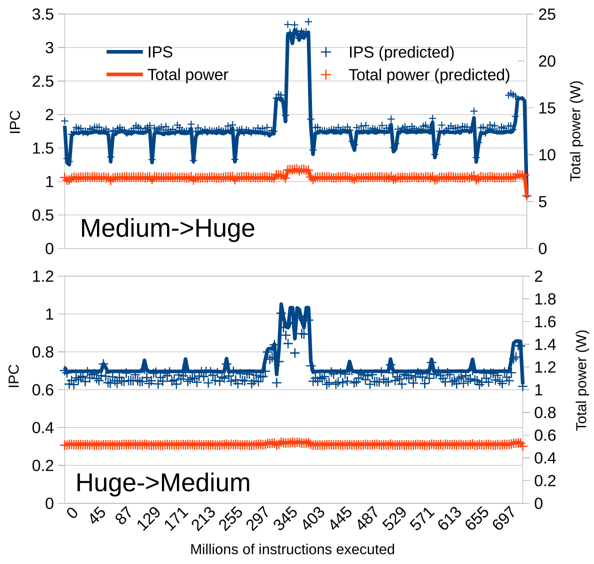


Figure 7.7: IPC and power prediction for the bodytrack PARSEC benchmark. The upper part of the figure shows predictions for the *Huge* core from measurements made on the *Medium* core, while the lower part shows the inverse.

we developed a mechanism of sparse virtual sensing guaranteeing a minimal number of counters and sensors for performance and power predictions, can be easily used to overcome this perceived limitation.

7.8 Summary

Single-ISA heterogeneous MPSoCs with architecturally differentiated cores are an attractive computing paradigm achieving higher performance and energy efficiency compared to the homogeneous counterparts. However, aggressively heterogeneous MPSoCs pose a serious challenge to the traditional Linux operating system scheduling as existing load balancers do not fully exploit widely heterogeneous architectural configurations (core types, core strength, and their combinations) and are not capable of adapting to new processor architectural changes without significant engineering efforts. In this chapter, we presented SmartBalance: a closed-loop approach to load balancing under a sense-predict-balance paradigm that can efficiently manage the chip resources while opportunistically exploiting the diverse workload and performance-power characteristics of different cores for energy efficiency. Our SmartBalance approach deployed an efficient runtime optimization executed at the beginning of each balancing epoch (covering multiple Linux scheduling periods) that effectively predicts the per-thread power and performance characteristics for each core, allowing efficient online load balancing. Our approach showed an improvement in energy efficiency of over 50% for a Heterogeneous MPSoC with 4-cores executing benchmarks from the PARSEC benchmark

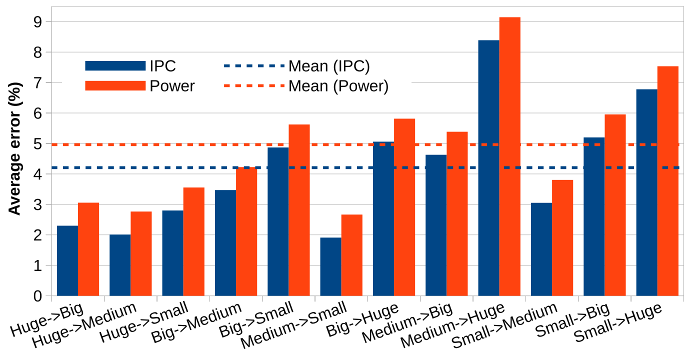
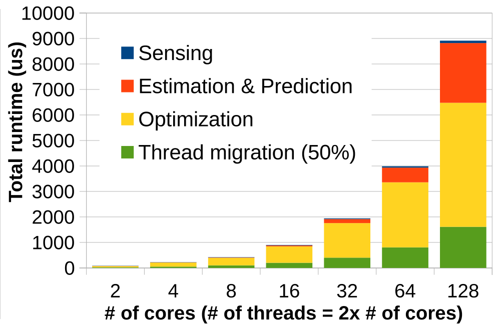


Figure 7.8: Average error in performance and power prediction across PARSEC.

SmartBalance phases runtime (µs)				
Number of threads	**Sensing**	**Estm. & Pred.**	**Optim.**	**Thread Migr.**
4	9.4	4.5	161.3	50.4
8	11.9	12.0	295.9	100.8
16	13.5	20.0	420.0	201.6
32	15.3	28.0	530.0	403.2
64	16.8	36.8	645.0	806.4
128	18.5	55.0	750.0	1612.8

(a)

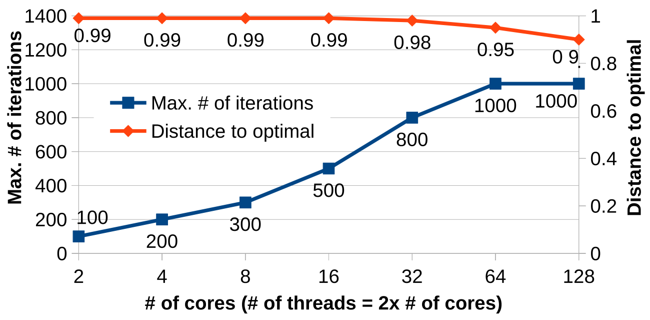


(b)

Figure 7.9: (a) Overhead with the quad-core HMP and (b) scalability analysis with increasing number of threads and cores for the SmartBalance approach.

suite, as compared to the standard vanilla Linux kernel load balancer and over 20% improvement with respect to the state-of-the-art ARM's global task scheduler (GTS). The approach can be generalized to support multiple mix of goals. By reusing most of the Linux kernel source with minimal changes along with a light-weight run-time scalable global optimization engine, the energy efficiency of both application and OS workloads can be improved for heterogeneous multicore processors (HMP).

(a)



(b)

***Opt* parameter**	perturb	Δperturb	accept	Δaccept
Value	3670	0.98	28	0.91

Figure 7.10: Maximum number of iterations (Opt_{max_iter} parameter) for each scalability scenario (a). The *distance to optimal* is obtained by running our optimization algorithm for synthetic cases whose optimal solution is known. (b) shows the values used for the remaining optimization parameters.

Chapter 8

FPGA Prototyping of CPSoC

8.1 Introduction

In the nanometer era, complex SoCs have higher risk of re-spins as large SoC designs present challenges in both design and verification on leading edge process nodes. FPGA prototyping is currently an established way for pre-silicon SoC validation, accelerated system software development and meeting time-to-market demands. FPGA prototypes provide early access to a fully functional hardware platform and the ability to develop and test several aspects of the system ahead of the actual silicon availability. The availability of an FPGA prototype can enable early application software development, operating system integration, as well as better understanding of the complex SoC system performance and quality of other design metrics.

Customized hardware emulation tools are capable and fast, but highly expensive, often out of reach for small design teams. FPGA-based prototyping tools are scalable, cost-effective, and are readily available. FPGA prototypes offer improved debug visibility and are well suited for software co-verification as well as rapid turnaround of design changes. In fact, a complete verification effort has multiple types of tests for all individual components, intellectual property (IP) blocks, as well as the completely integrated design running actual software (co-verification). As the verification and validation of such complex SoCs are far beyond what traditional simulation tools alone can do in reasonable time, FPGA prototyping is increasingly gaining popularity as a pre-silicon SoC validation approach in complex SoC design. In fact, FPGA Prototyping is no longer optional because of the re-spin cost of chips, the difficulty to simulate many-core system, and the narrowing window of time-to-market.

However, FPGA prototyping of emerging SoCs is not only complex and non-trivial task, but also involves a significant effort at different stages of development. In addition to requiring major development time in implementing the given specification in HDL, there are several time-consuming and complex steps involved during the deployment of the design including multi-FPGA design partitioning across many FPGAs, arduous and long bring-up time with complex debugging steps, as well as possibility of large number of errors and issues emanating out of the complexity of the design. Even with a small change in the requirement specification, the design and validation steps can incur a significant time and validation effort. Consequently, FPGA prototyping of emerging SoCs need to be built with the aim to reduce the development time by supporting modularity and reusability of design and test, support for configurability to accommodate range of specifications, as well as providing extensibility- the capability of extending a baseline design with ease.

In this chapter, we present an open architecture based CPSoC design library to build and evaluate the effectiveness of various architectural features, by leveraging industrial-grade design/verification/synthesis flow to significantly reduce design time and alleviate prototyping difficulties. We present CPSoC FPGA platform as an exemplar self-aware MPSoC paradigm along with open heterogeneous architecture frameworks for simulation, synthesis, and design exploration which support extensibility, scalability, and configurability, along-side an established base of supported software and verification tools. The CPSoC FPGA emulation and prototyping platform helps to investigate self-aware adaptive computing paradigms. We create an FPGA library using open source, general-purpose, processor cores to prototype multithreaded manycore architectures for CPSoC and provide a qualitative comparison against prior open source processors. In addition to emulating manycore architectures, the library can also be used for limitedly scalable multicore architecture using on-chip buses based on current standards such as AMBA or AXI. The chapter presents detailed description and FPGA implementation of the CPSoC architecture including cores, memory hierarchy that coherent memory system that integrates on-chip and off-chip communication networks, and on-chip sensor networks. Our implementation includes ring-oscillator (RO) based multi-purpose on-chip sensors integrated with multiple sensor-network-on-chip (sNoC) as discussed in Chapter 4 which in turn is interfaced either to a bus based shared memory architecture or to a communication and computation network-on-chip (cNoC) distributed fabric supporting several actuation mechanisms in the software and hardware stack. We describe the CPSoC platform features, present multiple use-cases of configurability, extensibility, and adaptation of the CPSoC architecture. In addition, we provide a comparison of FPGA implementation resources, backend synthesis runtimes, brief description of the test vector and coverage of the CPSoC test and verification efforts.

8.2 FPGA Prototyping Library for CPSoC

In order to build and illustrate the effectiveness of various architectural features in CPSoC paradigm in a fast, cost-effective, and modular way while supporting high degree of configurability and extensibility of CPSoC prototyping and emulation platform, a reusable and parameterizable HDL library components are absolutely necessary. We develop a CPSoC prototyping library called CPSoClib by leveraging industrial-grade CPU core design together with a robust software ecosystem. The components of the library broadly fall into four categories: processor and memories, networks-on-chip, on-chip sensors, and actuators as shown in Fig. 8.1. Since the CPSoC platform introduces several new hardware and architectural extensions for on-chip sensing, processing, and actuation, in this chapter we discuss the design and implementation of these components of the CPSoClib in the following subsections.

8.3 CPSoC Architecture and Design

CPSoC is a tiled-manycore architecture designed to support scalability, extensibility, and self-aware adaptation. The architecture supports both intra and inter-cluster communication. Intra-cluster tiles are connected via on-chip network interface chipset

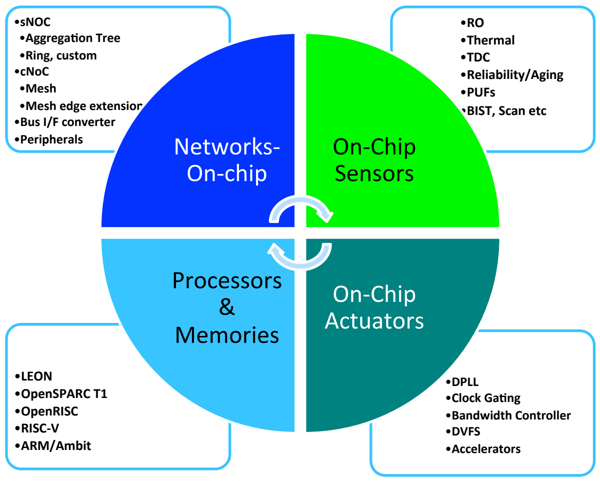


Figure 8.1: CPSoClib: an FPGA prototyping library for CPSoC. The library is organized in four groups of processor & memories, NoCs, sensors, and actuators.

(cNIC) using mesh topology by default, however, configuration ability allows for flexibility in the number of tiles in the CPSoC platform.

The cluster network interface chipset (cNIC) connects different clusters (inter-cluster) through a bridge which connects to the tile array through the upper left most tile. By multiplexing the three on-chip cNoCs over a single link, the cNIC also extends the cNoCs for off-chip access and seamless connection of multiple clusters to create large scale systems as illustrated in Fig. 8.2. The architecture uses existing state-of-the-art cache-coherence protocols and extends to multiple clusters enabling shared memory across multiple clusters. As the core in a tile of a cluster may be cache coherent with cores in another cluster, the capability to enable larger shared-memory manycore system is available.

8.3.1 Heterogeneous Tiled Cluster Architecture

CPSoC fabric is characterized by heterogeneity at multiple levels of abstraction and granularity. Fig. 8.2 depicts the various architectural configurations due to heterogeneity of the cluster and the tiles. In Fig. 8.2 (a) all the clusters and tiles are uniform resulting in a perfect homogeneous tiled cluster, while Fig. 8.2(b) depicts the architectural configuration with heterogeneous tiles but homogeneous cluster. Similar to the previous two configurations, we have the heterogeneous architectural configurations in Fig. 8.2 (c) and Fig. 8.2 (d) where heterogeneity is introduced in the cluster size as well as jointly in cluster size and tiles respectively.

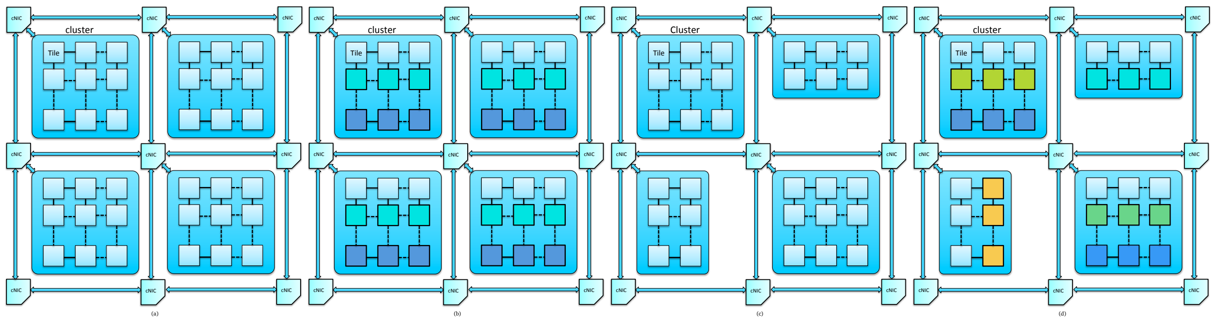


Figure 8.2: Homogeneous and Heterogeneous Tiled Architecture of CPSoC fabric (a) homogeneous cluster and tile (b) homogeneous cluster, heterogeneous tile (c) heterogeneous cluster, homogeneous tiles (d) heterogeneous cluster and tile.

8.3.2 Tile and cNIC

The architecture and configuration of the tile is shown in Fig. 8.3. The base line tile architecture consists of a SPARC cores [115, 151] with L1 cache, private L2 cache, on-chip cNoC routers, and distributed L3 cache, along with optional floating point unit (FPU) and accelerators. In order to support self-awareness and actuation mechanisms, the core is modified to include the on-chip sensing and actuation (OCSA) unit in the tile. The on-chip L2 and L3 caches connect directly with the cNoCs and uses the CPU-cache-crossbar (CCX) bus interface of OpenSPARC to connect the cores, the caches, FPU, I/O and other components. The CCX arbiter demultiplexes memory and floating-point requests from the core to the L2 and FPU and arbitrates responses back to the core.

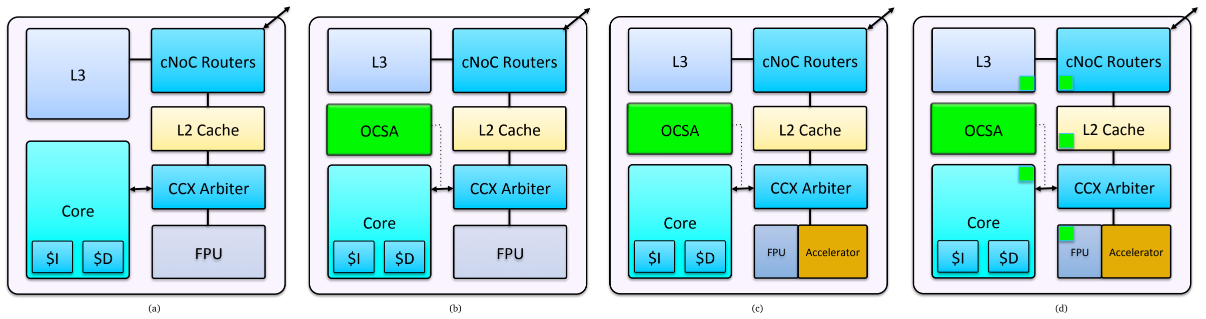


Figure 8.3: Architecture of the tile and its different configurations (a) baseline tile (b) tile with on-chip sensing and actuation (OCSA) unit (c) the tile with OCSA and accelerators (d) tile with distributed OCSA.

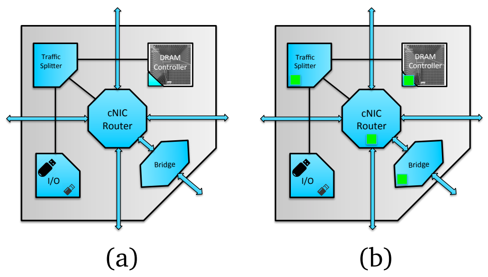


Figure 8.4: Cluster network interface chipset (cNIC)

8.3.3 Processor Cores

CPSoC platform can be built using processors with different instruction set architectures (ISA) that provide the ability to reconstitute or compose them (with varied component specifications) during the design time along with the tool chains. Open source processor cores (see Table 8.1) that provide the necessary tool chains are ideal candidates and we consider the following cores for the CPSoC FPGA prototyping and emulation. In this work, we use the SPARC cores (OpenSPARC T1 and Leon cores) because of the industry-hardened design, multi-threaded capability, simplicity, and modest silicon area requirements. In addition, these open frameworks have a stable code base, a matured ISA with compiler, OS, and validation test suite.

Table 8.1: Open Source Processors.

SL No	Processor Name	ISA	Bus	Type
1	LEON3[115]	SPARC V8	AMBA (32 bit)	In-Order
2	OpenRISC[261]	OpenRISC	Wishbone (32 bit)	In-Order
4	Amber ARM [302]	ARM	AMBA (32 bit)	In-Order
5	IVA [364]	PISA	64bit	O3
6	OpenSPARC T1[151]	SPARC V9	JBI	O3
7	RISC-V [358, 357]	RISC-V	RISC-V	O3

In order to build a heterogeneous CPSoC architecture and reduced programming complexity, cores of the same ISA are modified with different resources to get heterogeneous cores. Heterogeneous architectures with different ISA can also be built, but the added complexity and tool compatibility is limited in our present development. For a selected processor type and ISA, the IP cores and peripherals have to follow the bus standard of the processor. However, in order to facilitate the library components' use with different bus architectures, we develop bus converters and bridges such that components can be interfaced to a processor with different bus. Our design of the different components of the library are AMBA, AXI compatible and we provide bus converters (e.g., CCX to AXI, AMBA to Wishbone, AXI to APB and vice versa) to interface with other processor bus specifications.

8.3.4 Memory Hierarchy

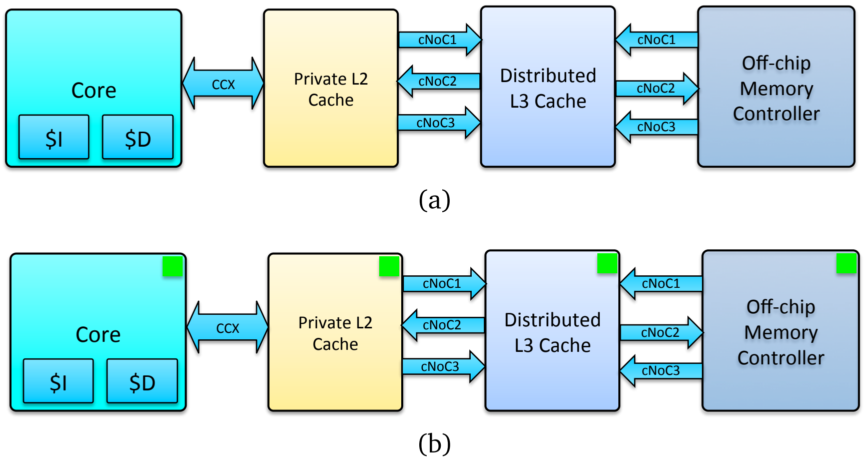


Figure 8.5: Memory hierarchy showing the data path and interfaces.

The L3 cache is distributed write-back cache shared by all the tiles in the cluster. The default cache configurations is 64KB per tile and 4-way set associativity with 64 bytes per cache line and 64 bit per entry but both the size and associativity is configurable.

8.3.5 Interconnect Architecture

8.3.5.1 Network-on-Chip (NoC)

The core-to-core NoC (cNoC) is an integral part of the tiled distributed architectures of CPSoC. There are three cNoCs that connect the tiles in a 2D mesh topology with purpose of providing communication between the tiles for cache coherence, I/O and memory traffic, inter-core interrupts. In addition to routing traffic destination for other cluster through the cNIC chipset bridge, the cNoCs maintain point-to-point ordering to maintain consistency. The cNoCs implementation uses physical networks, credit base flow control, and wormhole routing to ensure a deadlock-free operation.

8.3.5.2 cNIC Cluster Bridge

The cluster bridge connects the tile array to the cluster network interface chipset (cNIC) as shown in Fig. 8.3. All memory and I/O requests are directed through the bridge interface to be served by the cNIC. The bridge transparently multiplexes the cNoCs over a single link and provides a link between I/O and core clock domains. The bridge is connected to a cNIC router that implements virtual channels over a off-chip physical channel providing arbitration logic. In addition to traditional credit based flow control in the cNIC router, we introduce a software-defined configurable router with bandwidth, channel direction, and routing policy control to achieve improved adaptivity and control of the inter-cluster networks as shown in Fig. 3.15.

8.3.5.3 cNIC Chipset

The cNIC connects multiple clusters in the CPSoC prototype and it consists of the DRAM controller, the cluster bridge, the intra-chip network router (cNIC router) and the I/O interfaces.

8.3.6 On-chip Sensors and Sensor Network (sNoC)

Many of the on-chip sensors are low bandwidth sensors and thus the sensor data can be time division multiplexed over a router. However, as packet switching introduces considerable overhead, we consider a simplified custom circuit switched sNoC architecture as shown in Fig. 8.6. We consider two candidate topologies (ring and star) for the custom sNoC based on the sampling rate and bandwidth requirements. If the rate or bandwidth requirement is low, a ring topology may be sufficient. On the other hand, if the sampling rate desired is high other network topologies supporting higher bandwidth (e.g., the star topology) can be used.

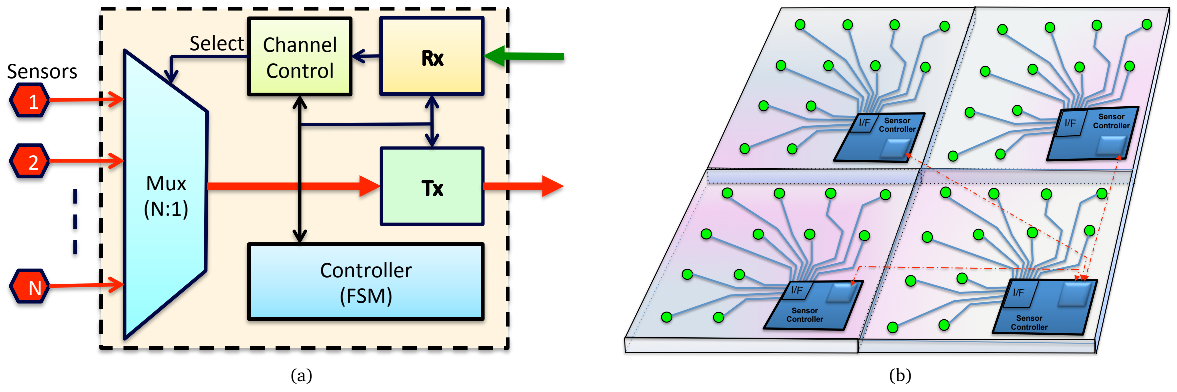
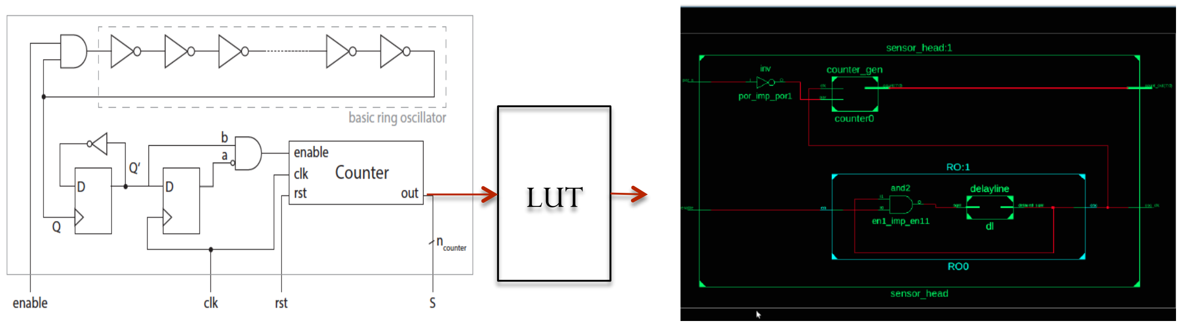


Figure 8.6: Resource efficient custom time division multiplexed (TDM) Router for sNoC.

We implemented a circuit switched ring topology sNoC as shown in Fig. 3.11 for 16 and 32 channels at each node/ router. The channels are time division multiplexed (TDM) for low bandwidth (slow varying phenomena) sensors such as aging and temperature. The comparison of the FPGA resources with respect to the packet switched aggregation tree based sNoC is substantially low. We found that our custom circuit switched sNoC can save on-chip resources required by two orders of magnitude [312].

8.3.7 Multi-purpose On-chip Sensors

Today's computing platforms are already equipped with few sensors such as temperature sensors and are expected to increase manyfold in the future. For example, the IBM Power 7 [215] processor has over 40 distributed thermal sensors, processor core and memory activity counters, off-chip current/voltage sensors, and critical path monitors. These sensors include physical sensors such as delay sensors, voltage and power sensors, temperature sensors [79], and reliability sensors. Sensors can also be implemented at higher levels of system stack such as architecture performance counters [337] and NoC traffic/congestion monitors [80]. Most existing sensors are designed solely to monitor specific phenomena. On the other hand, CPSoC expands the role of native sensors by multi-purposing these sensors and reducing sensing overheads through use of sensor fusion in absence of such new sensors. We show an example of RO based thermal sensor (as in Fig. 8.7) that is configurable at design time by specifying the generic parameters such as delay-line length and the counter range. The basic design can be configured and modified to build aging sensors as discussed in [11].



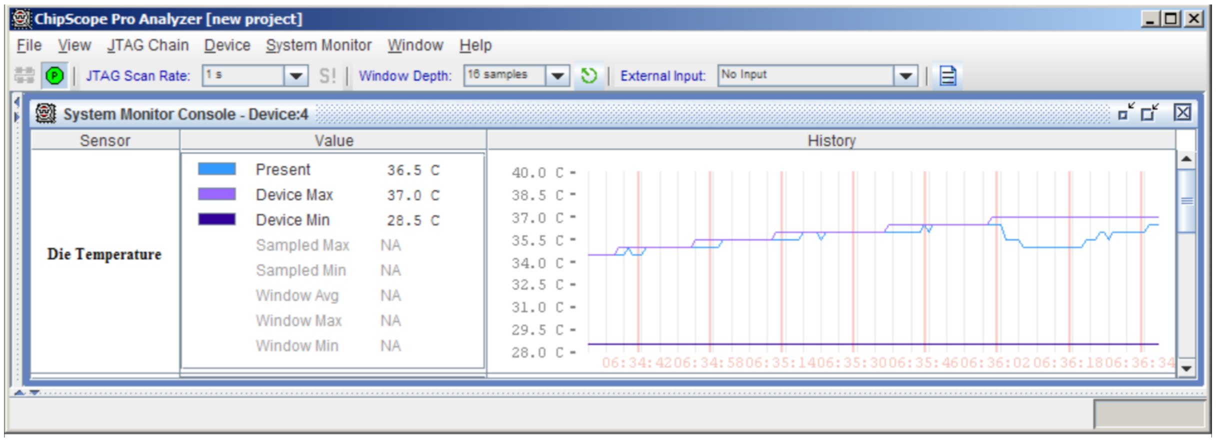
(a) (b)

Figure 8.7: Ring Oscillator (RO) based thermal sensor. The delay line length and the counter range (proportional to temperature range and precision) is configurable at design time.

Note that since most of the sensors considered in the CPSoC platform can be implemented using digital logic, there are no special technology/manufacturing requirements for prototyping in the FPGA. Although traditionally mixed-signal designs have been used to implement some specialized sensors, we believe that by virtualizing and fusing several digital sensors we can avoid such specialized sensor needs. For instance, although the leakage power sensors (and few types of thermal sensors) use mixed signal design requiring OpAmps, Signal conditioners, ADC etc., we overcome these limitations of process and custom design requirements by using virtual sensing as explained in [312]. We use simple ring oscillator (RO) based delay sensors as a proxy for different sensors (e.g., temperature and power) and accurately estimate their values while saving substantial sensor area, power, design complexity and cost. The virtual sensing approach described in [312] uses simple temperature sensor to estimate the power of the full chip at run-time. Therefore, the CPSoC paradigm enables intelligent choice of sensors along with virtual sensing and sensor fusion for different design concerns that can benefit the FPGA platform realization.

8.3.8 Built-in Sensors and System Monitors

In addition to on-chip sensors, the CPSoC platform can support external sensors to monitor the environment and other phenomena. The Xilinx platform provides a built-in feature as shown in Fig. 8.8 to read such sensor data through the system monitor interface. The system monitor interface reads the temperature and the supply voltage fluctuations at the centre of the FPGA in addition to several external analog channels interfaced through a high speed ADC. Note that the on-chip sensors that CPSoC emulates are directly synthesizable in the FPGA. Any custom sensor and actuator mechanism that cannot be directly synthesized limits the capability of the prototyping and emulation FPGA platform, however, custom ASIC design methodology or future FPGA platforms can provide these additional capabilities.



JTAG Scan Rate: 1 s

Window Depth: 16 samples

External Input: No Input

Sensor	Value	
Die Temperature	Present	36.5 C
	Device Max	37.0 C
	Device Min	28.5 C
	Sampled Max	NA
	Sampled Min	NA
	Window Avg	NA
	Window Max	NA
	Window Min	NA

Figure 8.8: Built-in FPGA sensors and system monitor support in the CPSoC FPGA platform.

8.3.9 On-Chip Actuation Mechanism

The CPSoC platform can support many actuation mechanisms across several layers; here we consider emulation and prototyping of these mechanisms in the FPGA architecture. The actuation mechanisms in the software layers can be supported with modifications to application, programming models, and the runtime system. However, the mechanism in the networking, and architecture layers require hardware design modifications. We illustrate the support of some of these mechanisms in the FPGA platform. We consider the dynamic frequency scaling (DFS) per core using a DPLL implementation and providing the DPLL control directly as shown in Fig. 8.9 to the runtime system and OS. Another actuation mechanism that can easily be implemented is clock gating which is an extension or special case of DFS. We introduce the heterogeneity of cores by different cache sizes and number of functional components (e.g., integer unit, FPU, timers etc.) that provide performance-power trade-offs for the runtime system to exploit.

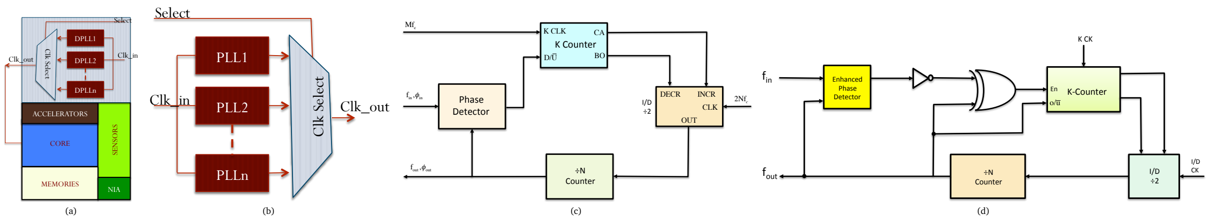


Figure 8.9: Clock frequency adaptation and control in CPSoC platform (a) clock scheme (b) clock selection (c) DPLL (d) DPLL with jitter reduction.

8.3.10 CPSoC Runtime and OS Support

The CPSoC runtime is developed with a custom message passing microkernel as well as the Linux 2.6.x kernel for different benchmark applications. The runtime provides abstraction to expose cyber and physical aspects of the application and platform to the runtime to achieve different system goals. In order to verify and test the hardware support for message passing among the cores and the on-chip sensing and actuation, the custom bare kernel supporting message passing over the cNoC is developed.

8.3.11 CPSoC FPGA Prototyping Platforms

We used different prototyping FPGA boards based on Xilinx Virtex Series of FPGA [152] [153] (with different technology nodes) as shown in Fig. 8.10 for proof-of-concept and emulation of CPSoC. The resource usage of different components of CPSoC in a typical prototyping scenario on the Virtex 6 board is shown in Fig. 8.4. The prototyping environment and tools used are tabulated in Table 8.3.

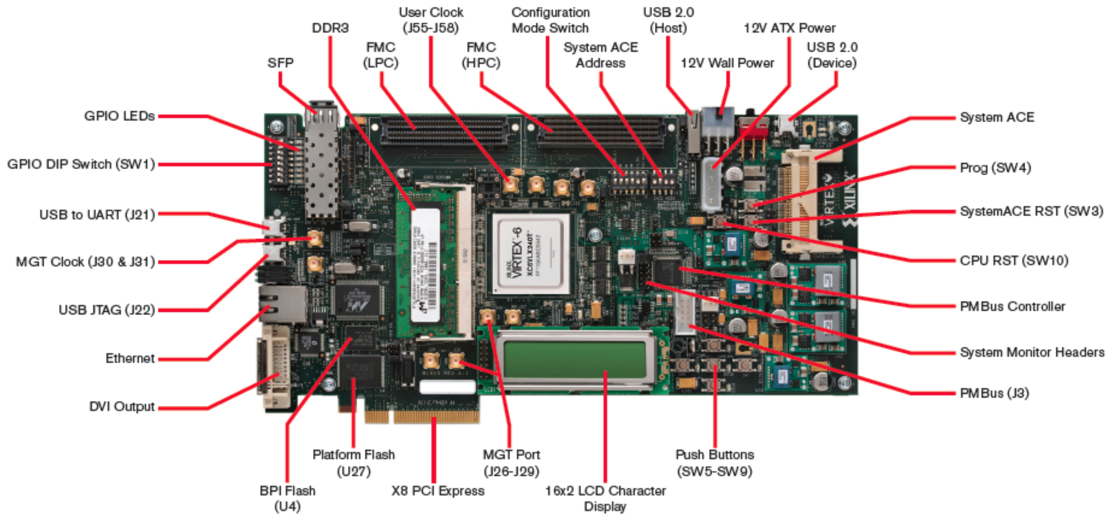


Figure 8.10: Proof-of-concept prototyping boards using Virtex-6 ML605 evaluation board [153].

8.4 Platform Features

The CPSoC platform is uniquely characterized by the four key features of configurability, extensibility, adaptivity, and heterogeneity in the FPGA platform.

- **Configurability**: The platform provides extensive configurability options for the cores by allowing changing the types, translation look-aside buffers (TLB) size, and enabling/disabling built-in accelerators such as FPU, crypto core (SPU) as described in Table 8.3. The caches at various levels (L1, L2, and L3) can be configured by their size, ways, associativity, etc. In addition, both the cNoC and sNoC can be configured by their topology, size, as well as bandwidth. The tiles are configurable by their address range for large 2-dimensional rectangular mesh interconnect of up to 256x256 tiles for manycore scalability.

- **Heterogeneity**: CPSoC platform supports different form and extent of heterogeneity at specific levels of granularity. Of specific interest is the support for heterogeneous multicore architectures and clusters. Different core types are implemented by diverse combination of the cache memories, availability / removal or accelerators such as FPU, SPU etc. as well as configuring the microarchitecture. The cores types can be further distinguished by varying the pipeline depth, ROB sizes, and issue width for the out-of-order architectures and in-order cores. However, the existing implementation is not supported with complete validation suites for the all the combinations.

- **Extensibility**: The extensibility of CPSoC platform is provided by enabling the

Table 8.2: FPGA Development and Prototyping Tools.

Tools	Description	Remarks/Specs
Host System	Linux/ Windows	Ubuntu 12.x/Win 7
OS Support	Linux (Debian / Ubuntu)	2.26.x/3.x
	Microkernel	real-time
Middleware	CyPhy middleware	version 1.0 provides predictive models of power, perf
Languages	C/C++	version 4.x
IDE	Xilinx ISE/EDK/Plan Ahead	version 14.x
	Vivado HLS	version 14.x
	Gaisler Eclipse IDE	
Compiler	GCC/LLVM	version 4.x/3.4
Debug Interface	JTAG	standard
	Chip Scope Pro	version 14.x
Debuggers	GDB/ Gaiseler GRMON	version 2.0.x

replacement of the core types, inclusion of custom accelerators as co-processors, as well as custom IPs using the bus converters and bridges. The platform can easily be extended with multiple ISA, for example, some of the clusters may be formed with different ISA such as ARM or open source RISC-V. The CPSoC platform includes bus converters (e.g., AXI4-Lite bridges) to connect wide range of I/O operations by interfacing memory mapped IO operations form the cNoC to the AXI-lite. By using the standard interface like AXI, many I/O devices can be interfaced while using existing drivers.

- **Adaptation**: One of the most compelling aim of creating the CPSoC platform is to easily study improved awareness and adaptation in CPSoC architecture and to enable rapid prototyping self-awareness and adaptation ideas backed by infrastructure for validation, characterization, and implementation. In order to corroborate this goal, the CPSoC paradigm includes several sensing and actuation mechanisms along with runtime software, APIs and SDK. The platform implements clock frequency adaptation and control, dynamic/adaptive voltage scaling and control, as well as adaptation support for the on-chip networks.

8.5 Tool-Chain and OS Support

The tool-chain and operating system support for the CPSoC platform is very critical for enabling goals for the CPSoC platform. Without proper tools and operating system support, the objective of creating a self-aware SoC prototype in FPGA can not be achieved. In the following subsection, we briefly describe the tools chain and simulation framework supporting the platform.

Table 8.3: CPSoC FPGA Platform and Configuration Options.

Component	Description	Configuration Options	Component	Description	Configuration Options
Processor Core	Leon3/SPARC T1	small/ big/ huge	Actuators	Application Approximation	
TLBs	translation look-aside buffers	8/16/32/64		Algorithmic choice	
cNoC	intra-chip topology	2D Mesh / Crossbar Connected		Memory Allocation	OS default
cNIC	inter-chip topology	2D/3D Mesh, crossbar		Task migration	OS default
ISU	Cortex-M3/Leon2/DLX-32	1/2 nos.		Allocation/ Load Balance	OS default/ Smartbalance
sNoC	Sensor network topology	Custom TDM/Aggregation Tree		Scheduling	OS default/modified CFS
L1 $I Cache	On-chip level 1 instruction cache,	16/32/64KB , block RAM		cNoC Bandwidth Control	present/ absent
L1 $D Cache	On-chip level 1 data cache,	8/16/32KB, block RAM		sNoC Bandwidth/Direction Control	present/ absent
L2 Cache	On-chip level 2 Cache,	no of sets, ways, block RAM		Accelerators	see below, include/ remove
L3 Cache	On-chip level 3 Cache	no of sets, ways, block RAM		Static Voltage Scaling	at board level
External Memory	DRAM	1-4 GB		Dynamic Frequency Scaling	per core
	Flesh Memory	1-2 GB		Clock gating	design time
	Hard Disk in board	256MB-1GB		Adaptive Voltage Scaling	per core
Sensors	Performance Counters	per core,		Power gating	not supported
	Program Phase Detector	per core, include/ remove	Accelerators	Crypto accelerator (SPU)	include /remove
	On-board Timers	per core, include/ remove		Floating Point Unit (FPU)	include/ remove
	NoC Traffic monitor	per NoC, include/ remove		O-chip Predictors	include / remove
	Ring Oscillators	two types, include/ remove	Chipset Bridge	interconnect clusters	
	Temperature	configurable, include/ remove	Arbiter	FPU and L2 arbitration	
	Aging /Reliability	RO based, include/ remove	Bus converter	AXI to CCX	
	Variability Monitors	Razors no, word packing size		AXI to Wishbone	
	CPM	include/ remove		AXI to APB	
	instruction power	include/ remove	BIST	On-chip stored tests	include / remove
	Voltage/Current/Power	core/ board level	Bootloading	OS booting	SD Card/ UART

8.5.1 API and Software Development Kit

The CPSoC platform is supported by several application programming interfaces (APIs) and a software development kit in order to ease development effort. The software development framework provides API for the accessing the on chip sensors. API functions are available to read sensor and monitor specific readings, convert the measurement data to meaningful scale, as well as process the sensed data (for example, pack several reading in a specified format), calculate average, rate of change, and the range of reading for a given time duration. The API functions are wrapped around drivers built for the Linux OS supporting the CPSoC platform. APIs are also built for accessing and using the actuation mechanisms using Linux driver. For example, the SDK includes API function to change the clock frequency of the cores and uncross by changing the on-chip DLL parameters. On similar lines, API functions are included in the SDK to support dynamic voltage scaling supported by the FPGA devices (Xilinx Zynq, Virtex series).

8.5.2 CPSoC Simulation Framework

In order to evaluate the functionalities of CPSoC, we have created an experimental CPSoC simulation platform shown in Fig. 8.11. At the core of our platform we use the *Gem5* performance simulator [49]. Gem5 includes cycle-accurate models for various CPU architectures, as well as peripheral models that enable full system simulations with Linux OS. We use *Gem5* to create both homogeneous and heterogeneous systems based on the model of supported processors (ARM, Alpha 21264, SPARC, MIPS). For obtaining power data, we have integrated the *McPAT* power model [218, 219] directly with the *Gem5* simulation framework. In addition, *Process Variability models* are augmented in the *McPAT* to reflect the manufacturing variabilities. *Gem5* is extended with the *Sensing Interface, Actuation Interface, Sensor Network,* and *Memories*. Hardware performance counters are emulated by exporting *McPAT* power information and other *Gem5*'s statistics at run-time. The CPSoC simulation platform also introduces a new component VAR (Vulnerability, Aging, and Reliability) by integrating Hotspot [147]. VAR provides a prediction model for system vulnerability (e.g., hotspot tracker, power bug tracker [306], malicious attack tracker etc.), aging and threshold voltage shift due to NBTI, PBTI, HCI, EM and other failure modes [162], and component and system level reliably, availability, and mean time to failure (MTTF) of each component and core.

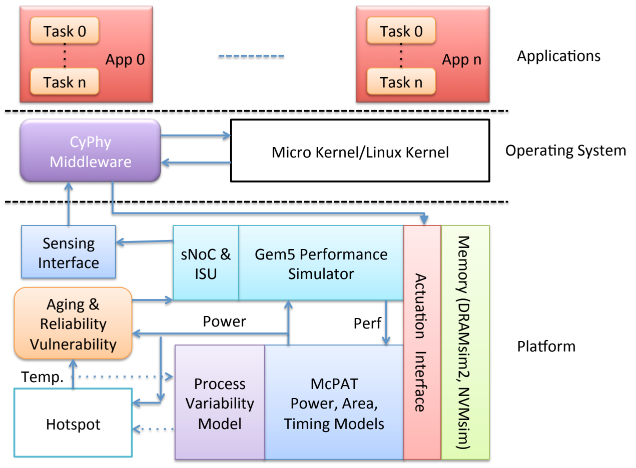

Task 0
App 0
Task n
Task 0
App n
Task n
Applications
CyPhy Middleware
Micro Kernel/Linux Kernel
Operating System
Sensing Interface
sNoC & ISU
Gem5 Performance Simulator
Actuation Interface
Memory (DRAMsim2, NVMsim)
Aging & Reliability Vulnerability
Power
Perf
Temp.
Process Variability Model
McPAT Power, Area, Timing Models
Hotspot
Platform

Figure 8.11: CPSoCSim: CPSoC Experimental Simulation Framework. Several new components are developed and integrated to existing architectural tool chains to build the CPSoC simulation framework.

8.6 Evaluation Results

In order to evaluate the resource utilization of each component of the CPSoClib, we perform independent syntheses and resource utilization of the components and tabulate the results in Table 8.4. The Table 8.4 shows the resources utilized by the ROs, the sNoC, cNoC, the actuation mechanism such as clock control DLL, and the heterogeneous processor cores. Fig. 8.12 shows the FPGA resources for different components of CPSoC using an OpenSPARC T1 core [151].

Table 8.4: Resource Usage for Different Hardware Components in Small CPSoC prototype.

FPGA Resources	Sensors	sNoC	cNoC	Actuation Mechanisms (DFS)		Processor Cores	
	RO	16 ch	Mesh2x2	DPLL (Standard)	DPLL (Jitter free)	Leon Big	Leon Small
Number of Slice Registers	0	670	1096	40	47	15053	12891
Number of Slice LUTs	31	42	3344	34	44	21777	16185
Number of occupied Slices	1	194	1557	14	19	10085	6424
Number of LUT Flip Flop pairs used	0	687	3854	41	51	26434	19670
Number of bonded IOBs	21	336	590	3	3	234	226
Average Fanout of Non-Clock Nets	0	1.70	4.20	2.30	2.11	3.79	4.10

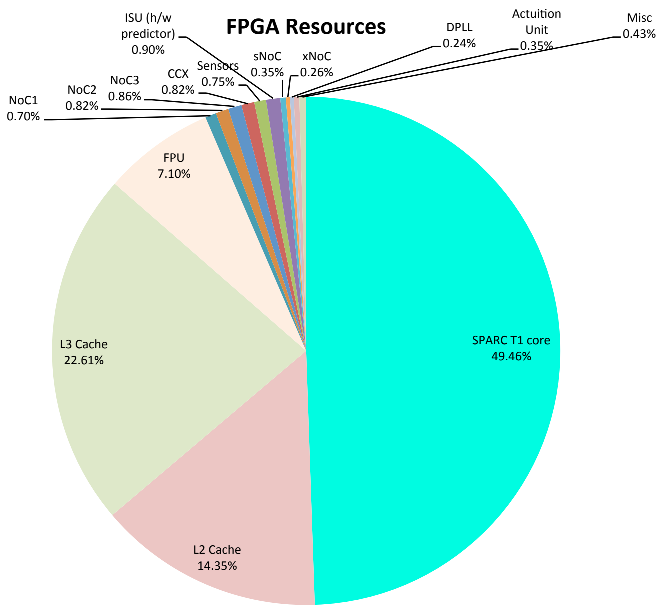


Figure 8.12: FPGA resources breakdown for different components of CPSoC.

8.6.1 Validation

The CPSoC platform is validated using different types of test cases as listed below:

- **Randomized Test**: This test involves generation of a number of randomized assembly test cases, where number of test parameters can be configured by the generating scripts written in python and Perl, to generate as many test instances. They cover test of the core, memory, branches, ALU, FPU, and interfaces.

- **Functional C Test**: In this validation methodology, test case written in C is used to perform directed test. The test cases involve small C function and microbenchmarks.
- **RTL and Gate Simulation**: In this approach, we perform simulation of the HDL code for the functional blocks of the CPSoC platform using industry standard tools such as Synopsys VCS, Mentor's Modelsim / Questasim, or built-in tools in Xilinx ISE / Vivado.
- **Full-System Simulation**: Full system simulation of the complete or portion of the platform is supported using the CPSoCSim simulator as described in Fig. 8.11
- **Test Coverage and Back-end support**: In order to improve the test coverage of the validation effort in creating the CPSoC platform, we generate coverage reports automatically and parse them to produce quick statistics for immediate feedback. Scripts have been developed to automate the FPGA and ASIC flow for industry standard tools. We have created scripts to support synthesis, timing analysis, place and route, as well as design space exploration.

8.6.2 FPGA Boards

The validation is performed using several off-the self FPGA boards from Xilinx and its vendors. We extend the capability of the boards by connecting multiple boards through the chip bridge, connected by external connector as shown in Fig. 8.13.

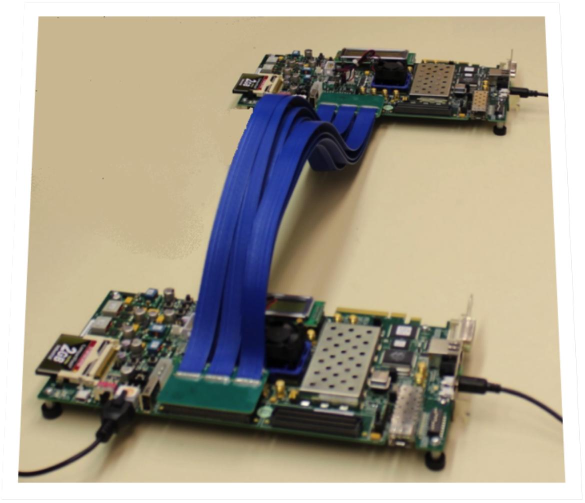


Figure 8.13: Xilinx Virtex-6 boards connected by the bridge and external connectors.

8.7 Enabled Use Cases

The CPSoC simulation and FPGA platform presented in this chapter can be used in several applications ranging from rapid prototyping of soft-realtime embedded systems to hard real-time cyber-physical systems. On-chip self-awareness with cross-layer virtual and physical sensing and actuations is a key for efficient use of heterogeneous architectures, and applications with guarantee runtime system QoS (performance, reliability, power, thermal behavior) in a highly dynamic environment. Our technical report [312] contains several sample applications where self-awareness is used to improve energy efficiency, increase system lifetime by reducing aging effects and improve system performance under thermal constraints. For instance, we show that cross-layer virtual sensing and actuation can improve the sensing accuracy and reduce the sensing overhead of thermal and power estimation by an order of magnitude [312].

In addition to the application described in [312], the CPSoC FPGA platform can be used for several enabled applications in research and development of self-aware architecture, performance analysis, power and energy awareness, thermal awareness, reliability analysis, cross-layer reliability and resilience, compiler and operating system research, as well as education. We are currently investigating more aggressive cross layer sensing and actuation mechanisms to improve system resilience and energy efficiency using the FPGA prototyping and library platform. For detailed descriptions of the FPGA library components and measurement results, please refer to our technical report [340].

8.8 Platform Limitations

We have attempted to create the first CPSoC FPGA platform with the goals of supporting key architectural features of CPSoC paradigm. While most of the architectural features of the CPSoC paradigm are implemented by the FPGA prototype, however, there are few that could not be realized in the current platform due to technology limitation, resource capability in the FPGA, as well as development cost. For example, the existing CPSoC FPGA platforms do not support seamless integration of Multiple-ISA for cores in the framework, rather insist on use of single-ISA cores in order to reduce the complexity of the development and tool chain. Several sensor types that are proposed in the CPSoC paradigm are not directly implementable in the FPGA fabric, which otherwise could easily be implemented in ASICs. For example, the implementation of oxide breakdown sensors for aging monitoring could not be included in the paradigm. However, we also foresee further extension of this work and the platform to address some of the shortcomings and limitations.

8.9 Related Work

Emerging many-core computing architectures [53] appear in two classes: homogeneous and heterogeneous configurations. homogeneous tiled many-core architectures with shared/distributed memories connected through a Network-on-Chip (NoC) fabric are emerging as general-purpose platforms (e.g., Tilera [38] and Intel Single Chip Computer [299]). On the other hand, high-performance embedded computing platforms typically deploy heterogeneous combinations of multi/many-core architectures, with tens to hundreds of small and big cores [51], to deliver unprecedented performance within an affordable power envelope. These advances are driven by Moore's law, demanding radical changes to both the architecture and the entire software stack [135]. Traditionally, MPSoC architectures [374] have been driven by features that improve performance, but constrained by power and thermal budgets [231]. Power-aware [59], thermal-aware [334], and reliability-aware [339] microarchitectures have been proposed over the last decade. However, a computing framework that addresses and assures the dependability of the information processing (i.e., the cyber aspects such as integrity, correctness, accuracy, timing, reliability and security) while simultaneously addressing the physical manifestations (in performance, power, thermal, aging, wear-out, material degradation, and reliability/dependability) of the information processing on the underlying computing platform, specifically SoC, has been missing. CyberPhysical-System-on-Chip (CPSoC) aims to coalesce these two traditionally disjoint aspects/abstractions of cyber/information world and the underlying physical computing worlds into a unified abstraction of computing. This helps to deal with the increased vulnerability due to semiconductor process variabilities as well as environmental and aging effects that induce errors in future computational platforms.

Even though MPSoCs are studied on various aspects such as power consumption [229], thermal [148], memory bandwidth [51] or network-on-chip [51], little attention has been given on their ability to be prototyped. A multi-core system on a multi-FPGA board is implemented in [60], but only on the perspective to study parallel programming. The influence of awareness mechanism on the architecture as well as system adaptation is not discussed. An efficient and rapid method to prototype and evaluate large MPSoC architectures on multi-FPGA boards is proposed in [200]. The authors only focused on proposing the architecture which allows large architecture prototypes to be easily distributed over to several chips in a device-independent and platform-independent way, but do not study the self-awareness capabilities in the architecture as well as the system adaptability. Nor do they explore the capability of heterogeneous architecture in the prototyping platform. On the contrary, in this work we focused on developing a reusable library for quickly building FPGA prototypes for CPSoC with the goal of supporting self-awareness and adaptation in scalable manycore architecture. Equipped with advances in machine learning, distributed on-chip sensor-networks, and multiple types of sensors, CPSoC brings many such attributes directly on-chip for a smart, intelligent, and autonomic computing fabric. The platform features extensive customization, extensibility, heterogeneity, and adaptation. Support for specialized features such as virtual sensing [311, 306], synergistic cross-layer cooperation, and actuator fusion [312] are some key attributes for CPSoC computing platforms.

8.10 Summary

In this chapter, we presented CPSoC: a sensor-actuator rich MPSoC paradigm and its prototyping using FPGAs. The proposed design paradigm enables self-awareness (i.e., the ability of the system to observe its own internal and external behaviors such that it is capable of making judicious decisions) and adaptation using the concept of cross-layer physical and virtual sensing and actuation abilities. We developed a FPGA emulation and prototyping platform using a combination of on-chip sensor network, adaptive communication network and cores, and several actuation mechanisms that enable the approach of computation-communication-control co-design. In our ongoing work, we plan to use this FPGA emulation platform to demonstrate many features of CPSoC, in particular, the ability to improve energy efficiency and adaptively respond to environmental and application load variations across multiple layers of abstraction that couple both software and hardware.

We presented an open architecture based CPSoC design library to build and evaluate the effectiveness of various architectural features, by leveraging industrial-grade development flow to significantly reduce design time and alleviate prototyping difficulties. We presented CPSoC FPGA platform as an exemplar self-aware MPSoC paradigm along with open heterogeneous architecture frameworks for simulation, synthesis, and design exploration which support extensibility, scalability, and configurability, along-side an established base of supported software and verification tools. The CPSoC FPGA emulation and prototyping platform helps to build and investigate self-aware adaptive computing paradigm. We created an FPGA library using open source, general-purpose, processor cores to prototype multithreaded manycore architectures for CPSoC and provide a qualitative comparison against prior open source processors. In addition to emulating manycore architecture, the library can also be used for limitedly scalable multicore architecture using on-chip buses based on standard such as AMBA or AXI. The chapter provided a detailed description and FPGA implementation of the CPSoC architecture including cores, memory hierarchy that coherent memory system that integrates on-chip and off-chip communication networks, and on-chip sensor networks.

Our implementation includes ring-oscillator (RO) based multi-purpose sensors integrated with multiple sensor-network-on-chip (sNoC) which in turn is interfaced either to a bus based shared memory architecture or to a communication and computation network-on-chip (cNoC) distributed fabric supporting several actuation mechanisms in the software and hardware stack. We described the CPSoC platform features, presented multiple use-cases of configurability, extensibility, and adaptation of the CPSoC architecture. In addition, we provided a comparison of FPGA implementations resources, backend synthesis runtimes, brief description of the test vector and coverage of the CPSoC test and verification efforts.

Chapter 9

Conclusions and Future Work

Embedded systems are increasingly seeing the need for self-awareness to operate autonomously in the face of uncertainty and unpredictability in the environment, the applications they execute, and in the manufactured hardware. As several application domains increasingly converge to a single multicore SoC platform, emerging multi/manycore heterogeneous SoC architectures [53] must address a multitude of potentially conflicting design and runtime constraints such as resiliency, energy, heat, cost, performance, security, etc., all in the face of highly dynamic operational behaviors and environmental conditions. Not only do they have to address the emerging challenges of manufacturing process variability [125] and dark silicon [107], but they also need to address the increasing complexity of programming and the burden of on-chip efficient resource management of these multicore heterogeneous systems. Programming and managing the resources in these systems with multiple conflicting goals have become hard as we are faced with a multidimensional optimization problem that requires improved awareness and runtime decision making capability. As a result, our fundamental computing model must change to address this multidimensional optimization and adaptation problem.

The thesis presented and built a new class of self-aware SoC called Cyber-Physical-Systems-on-Chip (CPSoC) [310, 311, 305, 312] and contributed toward a principled design and implementation of awareness and adaptation mechanisms for whole-stack co-design and optimization. The dissertation presented a new paradigm through the principled design and implementation of awareness and adaptation mechanisms in emerging MPSoCs (e.g., mobile SoCs and heterogeneous MPSoC) to deliver greater performance, energy efficiency, and resilience by adopting *self-aware control theoretic cross-layer mechanisms*. The dissertation made the following specific contributions:

- **CPSoC Paradigm.** The dissertation presented the foundation for a new class of *self-aware adaptive SoC* called Cyber-Physical-Systems-on-Chip (CPSoC) that contributes toward the whole-stack co-design of emerging MPSoCs by using the notion of self-aware adaptation through a tightly coupled optimization of control, communication, and computing in order to achieve competing design and runtime goals (e.g., boosting energy and power efficiency, improving thermal resilience, and delivering greater performance). The notion of self-awareness enables a system to monitor its own state and behavior such that it is capable of making judicious decisions and adapt intelligently. Unlike traditional MPSoCs, CPSoCs are distinguished by an intelligent co-design of the control, communication, and computing (C3) infrastructure while considering both the cyber and physical aspects together so as to adaptively achieve desired objectives and goals. CPSoC's sensor-actuator rich scalable architecture intrinsically couples on-chip and cross-layer sensing and actuation to enable self-awareness in a principled way. The thesis corroborated, through experiments and FPGA prototypes, the key idea that giving the SoC the freedom to opportunistically adapt the software and the hardware stack by infusing self-awareness mechanisms and steerable knobs across the stack can open up new and otherwise untapped opportunities in energy efficiency, performance, and thermal resilience.

- **Architectural Support**s: The dissertation provided the foundation for CPSoC through the development of closed loop modeling, predictive behavior models, architecture, algorithm, system software, and platform that makes it possible to adapt the system stack. The dissertation develops critical architectural support such as dedicated multi-sensor and actuator networks-on-chip, suitable abstractions to expose awareness properties to the system software and other layers, and architectural techniques that navigate these complex trade-offs to reduced design overheads. A predictive model based *cross-layer* design space exploration of these emerging heterogeneous architectural configuration and compositions with system level design goals is formulated to reduce exploration time by three orders of magnitude[307, 308].

- **Energy Efficiency**. The thesis presented a generalized technique that uses sensing-driven adaptation to improve energy efficiency of emerging heterogeneous MPSoCs [313, 244]. The experimental results showed that exposing and adapting the system software and OS with quantitative and predictive awareness of power and throughput at thread level can improve energy efficiency by over 50% with respect to existing Linux Kernel load balancer and over 20% with respect to ARM's GTS scheme [313].

- **Performance.** While SoC performance had been a well studied problem, the research showed that there still exists opportunities to improve SoC performance under peak thermal constraints by improving the peak thermal monitoring accuracy and timely DTM actions [306]. Precise and timely awareness of the dynamics of the physical phenomena (e.g., thermal hotspots) can avoid unwanted and inaccurate run-time decisions (e.g., false DTM triggers) resulting in improved performance by almost 28% [306] in a single core and holds the potential for substantial improvement in heterogeneous MPSoCs.

- **Thermal Resilience.** The thesis presented a run-time thermal and power prediction technique that estimates and predicts both the power consumption and temperature of each functional unit *ahead-of-time* using only temperature measurements that are readily available in the previous epoch [310, 306]. This ensures the awareness of the safe power a core can dissipate without violating a peak temperature safety limit while controlling the power using well known DVFS techniques. In addition, by being aware of the peak temperature which directly impacts the aging characteristics of functional units, an adaptive resting and recovery cycle can be scheduled to improve resilience and mitigate aging [312].

9.1 Future Work

The CPSoC paradigm presented in this thesis can be extended in several research directions. We list some of the short term and long term future work as follows. In the short term, the thesis can be extended with:

- **Cross-layer Reliability Models**: As emerging SoCs are prone to increasing reliability and resilience issues, the ability to model the reliability behavior and use such predictive models in run-time intelligent decision making to improve resilience of emerging SoC architecture would be a promising step ahead.
- **New Actuation Mechanisms**: The potential of new actuation mechanisms such as controlled algorithmic choice and approximation in combination with already available on-chip actuation mechanism can be explored to significantly improve performance and energy efficiency.
- **Multipurpose Sensors Design and Integration**: Sensors will play a crucial role in the self-awareness of emerging CPSoCs. The research can be extended to build multipurpose sensors and their integration methodology in the platform fabric.
- **Better Predictive Models**: The research can be extended to build more accurate and better predictive models using neural networks and emerging computational models.
- **Novel Application and Use cases**: Future research directions could develop novel uses cases of the paradigm by encompassing combination of energy efficiency, resilience, performance, and predictability targeting new application domains such as heterogeneous scale-out computing systems (e.g., data centers and cloud computing).

In the long term, the following research problems can be addressed:

- **Biologically Inspired Adaptation**: Future research directions could have design goals that encompass combination of energy efficiency, resilience, performance, and predictability targeting new application domains such as heterogeneous architectures and scale-out computing systems. As the need for intelligent management of computing system is increasing, use of on-chip machine learning and biologically inspired adaptation techniques would be promising.
- **Unification of Multiple Awareness Dimensions**: Self-awareness being a first class property to enable system to automatically adapt to their environment and goals, unifying and generalizing several awareness dimensions and goals across the system stack to achieve efficient system coordination is a promising research direction. An implementation of such a generalized scheme in traditional operating system such as Linux can result in meaningful enhancements toward a

self-aware operating system and its application supporting heterogeneous mobile SoCs and futuristic mobile platforms.

Bibliography

[1] *Dynamic System Identification: Experiment Design and Data Analysis*. Mathematics in Science and Engineering. Elsevier Science, 1977.

[2] *The Design and Analysis of Computer Experiments*. Springer-Verlag, 2003.

[3] *Microelectronic Circuits*. Oxford University Press, 5th london, u.k edition, 2004.

[4] Beri processor, arcina release 1. 2016. https://github.com/CTSRD-CHERI/beri.

[5] eXtensible Utah Multicore (xum). 2016. https://github.com/grantae/mips32r1_xum.

[6] Mips32 release 1. 2016. https://github.com/grantae/mips32r1_core.

[7] Zet processor. 2016. http://zet.aluzina.org/index.php/Zet_processor.

[8] Zylin cpu. 2016. https://github.com/zylin/zpu.

[9] Aeroflex Gaisler AB. Sparc v8 32-bit processor Leon3/Leon3-FT companion core data sheet, March 2010.

[10] Cobham Gaisler AB. Grlib IP core users manual, May 2015.

[11] M. Agarwal, V. Balakrishnan, A. Bhuyan, et al. Optimized Circuit Failure Prediction for Aging: Practicality and Promise. In *IEEE International Test Conference, ITC'08*, pages 1–10, Oct 2008.

[12] Mridul Agarwal, Bipul C. Paul, Ming Zhang, and Subhasish Mitra. Circuit Failure Prediction and Its Application to Transistor Aging. In *Proceedings of the 25th IEEE VLSI Test Symmposium*, VTS '07, pages 277–286, Washington, DC, USA, 2007. IEEE Computer Society.

[13] Vikas Agarwal, MS Hrishikesh, Stephen W Keckler, and Doug Burger. *Clock rate versus IPC: The end of the road for conventional microarchitectures*, volume 28. ACM, 2000.

[14] Hussam Amrouch, Thomas Ebi, Jurgen Schneider, Sri Parameswaran, and Jörg Henkel. Analyzing the thermal hotspots in fpga-based embedded systems. In *Field Programmable Logic and Applications (FPL), 23rd International Conference on*, pages 1–4. IEEE, 2013.

[15] A. Annamalai, R. Rodrigues, I. Koren, and S. Kundu. An opportunistic prediction-based thread scheduling to maximize throughput/watt in AMPs. In *Parallel Architectures and Compilation Techniques (PACT), 22nd International Conference on*, pages 63–72, Sept 2013.

[16] ARM Inc. big.LITTLE Technology: The Future of Mobile. 2013. http://www.arm.com/files/pdf/big_LITTLE_Technology_the_Futue_of _Mobile.pdf.

[17] Asen Asenov. Random dopant induced threshold voltage lowering and fluctuations in sub-0.1 μm MOSFET's: A 3-D atomistic simulation study. *Electron Devices, IEEE Transactions on*, 45(12):2505–2513, 1998.

[18] Asen Asenov, Andrew R Brown, John H Davies, Savas Kaya, and Gabriela Slavcheva. Simulation of intrinsic parameter fluctuations in decananometer and nanometer-scale MOSFETs. *Electron Devices, IEEE Transactions on*, 50(9):1837–1852, 2003.

[19] Asen Asenov, Savas Kaya, and John H Davies. Intrinsic threshold voltage fluctuations in decanano MOSFETs due to local oxide thickness variations. *Electron Devices, IEEE Transactions on*, 49(1):112–119, 2002.

[20] Todd Austin, Valeria Bertacco, David Blaauw, and Trevor Mudge. Opportunities and challenges for better than worst-case design. In *Proceedings of the Asia and South Pacific Design Automation Conference*, pages 2–7. ACM, 2005.

[21] Todd Austin, Valeria Bertacco, Scott Mahlke, and Yu Cao. Reliable systems on unreliable fabrics. *Design & Test of Computers, IEEE*, 25(4):322–332, 2008.

[22] Bernard J. Baars and Stan Franklin. Consciousness is computational: the LIDA model of global workspace theory. *International Journal of Machine Consciousness, World Scientific Publishing Company*, 2009.

[23] B.J. Baars. *A Cognitive Theory of Consciousness*. Cambridge University Press, 1989.

[24] B.J. Baars. The conscious access hypothesis: origins and recent evidence. *Trends in Cognitive Science*, 6(1):47–52, 2002.

[25] M. Bakhouya. A Bio-Inspired Architecture for Autonomic Network-on-Chip. In Phan Cong-Vinh, editor, *Autonomic Networking-on-Chip - Bio-Inspired Specification, Development, and Verification*, chapter 1, pages 1–20. CRC Press, December 2011.

[26] M. Bakhouya and J. Gaber. Bio-inspired Approaches for Engineering Adaptive Systems. *Procedia Computer Science*, 32:862 – 869, 2014. The 5th International Conference on Ambient Systems, Networks and Technologies (ANT'14), the 4th International Conference on Sustainable Energy Information Technology (SEIT'14).

[27] Saisanthosh Balakrishnan, Ravi Rajwar, Mike Upton, and Konrad Lai. The Impact of Performance Asymmetry in Emerging Multicore Architectures. *SIGARCH Comput. Archit. News*, 33(2):506–517, May 2005.

[28] Jonathan Balkind, Michael McKeown, Yaosheng Fu, et al. OpenPiton: An Open Source Manycore Research Framework. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '16, pages 217–232, New York, NY, USA, 2016. ACM.

[29] Luiz André Barroso, Jimmy Clidaras, and Urs Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis lectures on computer architecture*, 8(3):1–154, 2013.

[30] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *IEEE computer*, 40(12):33–37, 2007.

[31] Lyonel Barthe, Luis Vitorio Cargnini, Pascal Benoit, and Lionel Torres. The secretblaze: A configurable and cost-effective open-source soft-core processor. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), IEEE International Symposium on*, pages 310–313. IEEE, 2011.

[32] Andrea Bartolini, Matteo Cacciari, Andrea Tilli, and Luca Benini. A distributed and self-calibrating model-predictive controller for energy and thermal management of high-performance multicores. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2011.

[33] Andrea Bartolini, Matteo Cacciari, Andrea Tilli, Luca Benini, and Matthias Gries. A virtual platform environment for exploring power, thermal and reliability management control strategies in high-performance multicores. In *Proceedings of the 20th symposium on Great lakes symposium on VLSI*, GLSVLSI '10, pages 311–316, New York, NY, USA, 2010. ACM.

[34] A Bassi, A Veggetti, L Croce, and A Bogliolo. Measuring the effects of process variations on circuit performance by means of digitally-controllable ring oscillators.

In *Microelectronic Test Structures, International Conference on*, pages 214–217. IEEE, 2003.

[35] Robert C Baumann. Radiation-induced soft errors in advanced semiconductor technologies. *Device and Materials Reliability, IEEE Transactions on*, 5(3):305–316, 2005.

[36] Michela Becchi et al. Dynamic thread assignment on heterogeneous multiprocessor architectures. In *Proceedings of the 3rd conference on Computing frontiers*, CF '06, pages 29–40, New York, NY, USA, 2006. ACM.

[37] Nathan Beckmann, Po-An Tsai, and Daniel Sanchez. Scaling distributed cache hierarchies through computation and data co-scheduling. In *High Performance Computer Architecture (HPCA), IEEE 21st International Symposium on*, pages 538–550. IEEE, 2015.

[38] S. Bell, B. Edwards, J. Amann, et al. TILE64 - Processor: A 64-Core SoC with Mesh Interconnect. In *Solid-State Circuits Conference, ISSCC'08. Digest of Technical Papers. IEEE International*, pages 88–598, 2008.

[39] L. Benini, A. Bogliolo, and G. De Micheli. A survey of design techniques for system-level dynamic power management. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 8(3):299–316, June 2000.

[40] Luca Benini and Giovanni De Micheli. Networks-on-Chips: A New SoC Paradigm. *Computer*, 35(1):70–78, January 2002.

[41] C. Benito, P. Ituero, and M. Lopez-Vallejo. A Low-Area Reference-Free Power Supply Sensor. In *Digital System Design (DSD), Euromicro Conference on*, pages 728–733, Sept 2013.

[42] Keren Bergman, Shekhar Borkar, Dan Campbell, et al. Exascale computing study Technology challenges in achieving exascale systems. *DARPA Exascale Hardware Study*, 2008.

[43] Joseph B Bernstein, Moshe Gurfinkel, Xiaojun Li, et al. Electronic circuit reliability modeling. *Microelectronics Reliability*, 46(12):1957–1979, 2006.

[44] Srikar Bhagavatula and Byunghoo Jung. A low power real-time on-chip power sensor in 45-nm SOI. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 59(7):1577–1587, 2012.

[45] Srikar Bhagavatula and Byunghoo Jung. A power sensor with 80ns response time for power management in microprocessors. In *Proceedings of the IEEE Custom Integrated Circuits Conference, CICC'13*, pages 1–4, San Jose, CA, USA, September 22-25 2013.

[46] Manjul Bhushan, Anne Gattiker, Mark B Ketchen, and Koushik K Das. Ring oscillators for CMOS process tuning and variability control. *Semiconductor Manufacturing, IEEE Transactions on*, 19(1):10–18, 2006.

[47] Christian Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

[48] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The PARSEC benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 72–81. ACM, 2008.

[49] N. Binkert et al. The Gem5 Simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, August 2011.

[50] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog Computing and Its Role in the Internet of Things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC ’12, pages 13–16, New York, NY, USA, 2012. ACM.

[51] S. Borkar. Thousand Core Chips-A Technology Perspective. In *Design Automation Conference, 2007. DAC ’07. 44th ACM/IEEE*, pages 746 –749, june 2007.

[52] Shekhar Borkar. 3D integration for energy efficient system design. In *Proceedings of the 48th Design Automation Conference*, pages 214–219. ACM, 2011.

[53] Shekhar Borkar and Andrew A Chien. The future of microprocessors. *Communications of the ACM*, 54(5):67–77, 2011.

[54] Shekhar Borkar, Tanay Karnik, Siva Narendra, et al. Parameter variations and impact on circuits and microarchitecture. In *Proceedings of the 40th annual Design Automation Conference*, DAC ’03, pages 338–342, New York, NY, USA, 2003. ACM.

[55] K.A. Bowman, C. Tokunaga, T. Karnik, V.K. De, and J.W. Tschanz. A 22 nm All-Digital Dynamically Adaptive Clock Distribution for Supply Voltage Droop Tolerance. *Solid-State Circuits, IEEE Journal of*, 48(4):907–916, 2013.

[56] David Brooks, Robert P Dick, Russ Joseph, and Li Shang. Power, Thermal, and Reliability Modeling in Nanometer-Scale Microprocessors. *Micro, IEEE*, 27(3):49–62, 2007.

[57] David Brooks and Margaret Martonosi. Dynamic thermal management for high-performance microprocessors. In *High-Performance Computer Architecture, HPCA. The Seventh International Symposium on*, pages 171–182. IEEE, 2001.

[58] David Brooks, Vivek Tiwari, and Margaret Martonosi. *Wattch: a framework for architectural-level power analysis and optimizations*, volume 28. ACM, 2000.

[59] David M. Brooks, Pradip Bose, Stanley E. Schuster, et al. Power-Aware Microarchitecture: Design and Modeling Challenges for Next-Generation Microprocessors. *IEEE Micro*, 20(6):26–44, November 2000.

[60] Dan Burke, John Wawrzynek, Alex Krasnov, et al. RAMP blue: Implementation of a manycore 1008 processor system. 2008.

[61] Remi Busseuil, Lyonel Barthe, Gabriel Marchesan Almeida, et al. Open-Scale: A scalable, open-source NOC-based MPSoC for design space exploration. In *Reconfigurable Computing and FPGAs (ReConFig), International Conference on*, pages 357–362. IEEE, 2011.

[62] Adam C Cabe, Zhenyu Qi, Stuart N Wooters, Travis N Blalock, and Mircea R Stan. Small embeddable NBTI sensors (SENS) for tracking on-chip performance decay. In *Quality of Electronic Design,ISQED'09. Quality Electronic Design*, pages 1–6. IEEE, 2009.

[63] Stephen L Campbell, Nancy K Nichols, and William J Terrell. Duality, observability, and controllability for linear time-varying descriptor systems. *Circuits, Systems and Signal Processing*, 10(4):455–470, 1991.

[64] F. Cancare, S. Bhandari, D.B. Bartolini, M. Carminati, and M.D. Santambrogio. A bird's eye view of FPGA-based Evolvable Hardware. In *Adaptive Hardware and Systems (AHS), NASA/ESA Conference on*, pages 169–175, June 2011.

[65] Yu Cao. *Predictive technology model for robust nanoelectronic design*. Springer Science & Business Media, 2011.

[66] Christopher Celio, David A. Patterson, and Krste Asanovic. The Berkeley Out-of-Order Machine (BOOM): An Industry-Competitive, Synthesizable, Parameterized RISC-V Processor. Technical Report UCB/EECS-2015-167, EECS Department, University of California, Berkeley, Jun 2015.

[67] T.-B. Chan, P. Gupta, A.B. Kahng, and L. Lai. Synthesis and Analysis of Design-Dependent Ring Oscillator (DDRO) Performance Monitors, 2013.

[68] Anantha P Chandrakasan, William J Bowhill, and Frank Fox. *Design of high-performance microprocessor circuits*. Wiley-IEEE press, 2000.

[69] Chun-Chi Chen, Wen-Fu Lu, Chin-Chung Tsai, and Chun-Chi Chen. A time-to-digital-converter-based CMOS smart temperature sensor. In *Circuits and Systems, ISCAS'05. IEEE International Symposium on*, pages 560–563 Vol. 1, 2005.

[70] Jian Chen and Lizy K John. Efficient program scheduling for heterogeneous multi-core processors. In *DAC '09*, pages 927 –930, july 2009.

[71] Poki Chen, Chun-Chi Chen, Yu-Han Peng, Kai-Ming Wang, and Yu-Shin Wang. A time-domain SAR smart temperature sensor with curvature compensation and a 3σ inaccuracy of- 0.4 C + 0.6 C over a 0 C to 90 C range. *Solid-State Circuits, IEEE Journal of*, 45(3):600–609, 2010.

[72] Qikai Chen, Saibal Mukhopadhyay, Hamid Mahmoodi, and Kaushik Roy. Process variation tolerant online current monitor for robust systems. In *On-Line Testing Symposium, IOLTS'05. 11th IEEE International*, pages 171–176, July 2005.

[73] Tao Chen, Funmilade Faniyi, Rami Bahsoon, et al. The Handbook of Engineering Self-Aware and Self-Expressive Systems. *Computing Research Repository (CoRR)*, abs/1409.1793, 2014.

[74] Tianshi Chen, Yunji Chen, Qi Guo, et al. Effective and efficient microprocessor design space exploration using unlabeled design configurations. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(1):20, 2013.

[75] Tianshi Chen, Qi Guo, Ke Tang, et al. Archranker: A ranking approach to design space exploration. In *Computer Architecture (ISCA), ACM/IEEE 41st International Symposium on*, pages 85–96. IEEE, 2014.

[76] Tze Wee Chen, Kyunglok Kim, Young Moon Kim, and Subhasish Mitra. Gate-oxide early life failure prediction. In *VLSI Test Symposium, VTS'08. 26th IEEE*, pages 111–118. IEEE, 2008.

[77] Betty HC Cheng, Rogerio De Lemos, Holger Giese, et al. Software engineering for self-adaptive systems: A research roadmap. In *Software engineering for self-adaptive systems*, pages 1–26. Springer, 2009.

[78] Nagabhushan Chitlur, Ganapati Srinivasa, Scott Hahn, et al. QuickIA: Exploring heterogeneous architectures on real prototypes. In *High Performance Computer Architecture (HPCA), IEEE 18th International Symposium on*, pages 1–8. IEEE, 2012.

[79] Ching-Che Chung and Cheng-Ruei Yang. An Autocalibrated All-Digital Temperature Sensor for On-Chip Thermal Monitoring. *Circuits and Systems II: Express Briefs, IEEE Transactions on*, 58(2):105–109, 2011.

[80] C. Ciordas, T. Basten, A. Radulescu, K. Goossens, and J. Meerbergen. An event-based network-on-chip monitoring service. In *High-Level Design Validation and Test Workshop, Ninth IEEE International*, pages 149–154, 2004.

[81] Ali Çivril and Malik Magdon-Ismail. On selecting a maximum volume sub-matrix of a matrix and related problems. *Theoretical Computer Science*, 410(47):4801–4811, 2009.

[82] A. Clark. *Mindware An Introduction to the Philosophy of Cognitive Science*. Oxford University Press, New York, 2001.

[83] R. Cochran and S. Reda. Spectral techniques for high-resolution thermal characterization with limited sensor data. In *Design Automation Conference, DAC '09. 46th ACM/IEEE*, pages 478–483, 2009.

[84] W. E. Cohen. Tuning programs with OProfile. 2004. http://oprofile.sourceforge.net/news/.

[85] Phan Cong-Vinh, editor. *Autonomic Networking-on-Chip: Bio-Inspired Specification, Development, and Verification*. CRC Press, December 2011.

[86] Henry Cook and Kevin Skadron. Predictive design space exploration using genetically programmed response surfaces. In *Proceedings of the 45th annual Design Automation Conference*, pages 960–965. ACM, 2008.

[87] Ayse Kivilcim Coskun, Tajana Simunic Rosing, and Kenny C Gross. Temperature management in multiprocessor SoCs using online learning. In *Design Automation Conference,DAC'08. 45th ACM/IEEE*, pages 890–893. IEEE, 2008.

[88] Matthew Curtis-Maury, Filip Blagojevic, Christos D Antonopoulos, and Dimitrios S Nikolopoulos. Prediction-based power-performance adaptation of multithreaded scientific codes. *Parallel and Distributed Systems, IEEE Transactions on*, 19(10):1396–1410, 2008.

[89] William J Dally, James Balfour, David Black-Shaffer, et al. Efficient embedded computing. *Computer*, (7):27–32, 2008.

[90] Bishnu Prasad Das, Bharadwaj Amrutur, HS Jamadagni, NV Arvind, and V Visvanathan. Within-die gate delay variability measurement using reconfigurable ring oscillator. *Semiconductor Manufacturing, IEEE Transactions on*, 22(2):256–267, 2009.

[91] Shidhartha Das, Carlos Tokunaga, Sanjay Pant, et al. RazorII: In situ error detection and correction for PVT and SER tolerance. *Solid-State Circuits, IEEE Journal of*, 44(1):32–48, 2009.

[92] Basab Datta. On-chip thermal sensing in deep sub-micron CMOS. 2007.

[93] Kalyanmoy Deb. Multi-Objective Optimization using Evolutionary Algorithms. *John Wiley & Sons, Ltd, Chichester, England*, may 2001.

[94] Robert H Dennard, VL Rideout, E Bassous, and AR LeBlanc. Design of ion-implanted MOSFET's with very small physical dimensions. *Solid-State Circuits, IEEE Journal of*, 9(5):256–268, 1974.

[95] Gaurav Dhiman, Raid Ayoub, and Tajana Rosing. PDRAM: a hybrid PRAM and DRAM main memory system. In *Design Automation Conference, 2009. DAC'09. 46th ACM/IEEE*, pages 664–669. IEEE, 2009.

[96] Saurabh Dighe, Sriram R Vangal, Paolo Aseron, et al. Within-die variation-aware dynamic-voltage-frequency-scaling with optimal core allocation and thread hopping for the 80-core teraflops processor. *Solid-State Circuits, IEEE Journal of*, 46(1):184–193, 2011.

[97] E.W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974.

[98] David L Donoho. Compressed sensing. *Information Theory, IEEE Transactions on*, 52(4):1289–1306, 2006.

[99] J. Dorsey, S. Searles, M. Ciraula, et al. An Integrated Quad-Core Opteron Processor. In *2007 IEEE International Solid-State Circuits Conference. Digest of Technical Papers*, pages 102–103, Feb 2007.

[100] A.J. Drake, M.S. Floyd, R.L. Willaman, et al. Single-cycle, pulse-shaped critical path monitor in the $POWER7$ microprocessor. In *Low Power Electronics and Design (ISLPED), 2013 IEEE International Symposium on*, pages 193–198, 2013.

[101] Alan Drake, R Senger, H Deogun, et al. A distributed critical-path timing monitor for a 65nm high-performance microprocessor. In *Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International*, pages 398–399. IEEE, 2007.

[102] Alan J Drake, Robert M Senger, Harmander Singh, Gary D Carpenter, and Norman K James. Dynamic measurement of critical-path timing. In *Integrated Circuit Design and Technology and Tutorial, 2008. ICICDT 2008. IEEE International Conference on*, pages 249–252. IEEE, 2008.

[103] Ronald G Dreslinski, Michael Wieckowski, David Blaauw, Dennis Sylvester, and Trevor Mudge. Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits. *Proceedings of the IEEE*, 98(2):253–266, 2010.

[104] Nikil Dutt, Axel Jantsch, and Santanu Sarma. Toward Smart Embedded Systems: A Self-aware System-on-Chip (SoC) Perspective. *ACM Transactions on Embedded Computing Systems (TECS)*, 15(2):22, 2016.

[105] Thomas Ebi, M Faruque, and Jörg Henkel. Tape: Thermal-aware agent-based power econom multi/many-core architectures. In *Computer-Aided Design-Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on*, pages 302–309. IEEE, 2009.

[106] Mica R. Endsley. Design and Evaluation for Situation Awareness Enhancement. In *Proceedings of the Human Factors and Ergonomics Society 32th Annual Meeting*, pages 97–101, 1988.

[107] Hadi Esmaeilzadeh, Emily Blem, Renee St Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*, pages 365–376. IEEE, 2011.

[108] Lei Fang, Panos J Antsaklis, Luis A Montestruque, et al. Design of a wireless assisted pedestrian dead reckoning system - the NavMote experience. *Instrumentation and Measurement, IEEE Transactions on*, 54(6):2342–2358, Nov. 2005 2005.

[109] F. Faniyi, P. R. Lewis, R. Bahsoon, and X. Yao. Architecting Self-Aware Software Systems. In *Proc. of the 2014 IEEE/IFIP Conference on Software Architecture (WICSA)*, pages 91–94, Sydney, Australia, April 2014.

[110] Laurene V Fausett. *Fundamentals of neural networks*. Prentice-Hall, 1994.

[111] Michael Ferdman, Almutaz Adileh, Onur Kocberber, et al. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. *ACM SIGARCH Computer Architecture News*, 40(1):37–48, 2012.

[112] Michael S Floyd, Soraya Ghiasi, Tom W Keller, et al. System power management support in the IBM POWER6 microprocessor. *IBM Journal of Research and Development*, 51(6):733–746, 2007.

[113] James Fung and Steve Mann. Computer vision signal processing on graphics processing units. In *Acoustics, Speech, and Signal Processing, 2004. Proceedings.(ICASSP'04). IEEE International Conference on*, volume 5, pages V–93. IEEE, 2004.

[114] James Fung, Felix Tang, and Steve Mann. Mediated reality using computer graphics hardware for computer vision. In *Wearable Computers, 2002.(ISWC 2002). Proceedings. Sixth International Symposium on*, pages 83–89. IEEE, 2002.

[115] Aeroflex Gaisler. LEON3 Processor. *www.gaisler.com*, 2013.

[116] Rong Ge, Xizhou Feng, Shuaiwen Song, et al. PowerPack: Energy Profiling and Analysis of High-Performance Systems and Applications. *IEEE Trans. Parallel Distrib. Syst.*, 21(5):658–671, May 2010.

[117] Debanjan Ghosh, Raj Sharman, H. Raghav Rao, and Shambhu Upadhyaya. Self-healing Systems - Survey and Synthesis. *Decis. Support Syst.*, 42(4):2164–2185, January 2007.

[118] Keith Godfrey. Identification of parametric models from experimental data [Book Review]. *Automatic Control, IEEE Transactions on*, 44(12):2321–2322, 1999.

[119] Bhavishya Goel, Sally A McKee, Roberto Gioiosa, et al. Portable, scalable, per-core power estimation for intelligent resource management. In *Green Computing Conference, 2010 International*, pages 135–146. IEEE, 2010.

[120] P. Greenhalgh. Big.LITTLE processing with ARM Cortex-A15 & Cortex-A7: Improving energy efficiency in high-performance mobile platforms. Technical report, ARM Ltd., 2011.

[121] G. Grey. big.LITTLE Software Update. http://www.linaro.org/blog/hardware-update/big-little-software-update/, 2013.

[122] L. Guang, G. Plosila, J. Isoaho, and H. Tenhunen. HAMSoC: A Monitoring-Centric Design Approach for Adaptive Parallel Computing. In Phan Cong-Vinh, editor, *Autonomic Networking-on-Chip: Bio-Inspired Specification, Development, and Verification*, chapter 6, pages 135–164. CRC Press, December 2011.

[123] Liang Guang, Ethiopia Nigussie, Pekka Rantala, Jouni Isoaho, and Hannu Tenhunen. Hierarchical agent monitoring design approach towards self-aware parallel systems-on-chip. *ACM Trans. Embed. Comput. Syst.*, 9(3):1–24, 2010.

[124] Liang Guang, Juha Plosila, Jouni Isoaho, , and Hannu Tenhunen. Hierarchical Agent Monitored Parallel On-Chip System: A Novel Design Paradigm and its Formal Specification. *International Journal of Embedded and Real-Time Communication Systems (IJERTCS)*, 1(2), 2010.

[125] Puneet Gupta, Yuvraj Agarwal, Lara Dolecek, et al. Underdesigned and Opportunistic Computing in Presence of Hardware Variability. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 32(1):8–23, 2013.

[126] Yongkui Han, Israel Koren, and C Mani Krishna. TILTS: A fast architectural-level transient thermal simulation method. *Journal of Low Power Electronics*, 3(1):13–21, 2007.

[127] Katalin M Hangos, József Bokor, and Gábor Szederkényi. *Analysis and control of nonlinear process systems*. Springer Science & Business Media, 2006.

[128] Vinay Hanumaiah, Sarma Vrudhula, and Karam S Chatha. Performance optimal online dvfs and task migration techniques for thermally constrained multi-core processors. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 30(11):1677–1690, 2011.

[129] Monson H Hayes. Statistical Digital Signal Processing and Modeling. 1996.

[130] Simon S Haykin. *Neural networks and learning machines*, volume 3. Pearson Education Upper Saddle River, 2009.

[131] Stephan Hengstler, Daniel Prashanth, Sufen Fong, and Hamid Aghajan. Mesh-Eye: A Hybrid-Resolution Smart Camera Mote for Applications in Distributed Intelligent Surveillance. In *Information Processing in Sensor Networks, 2007. IPSN 2007. 6th International Symposium on*, pages 360–369, Stanford Univ., Stanford, Nov. 2007 2007.

[132] J. Henkel, L. Bauer, J. Becker, et al. Design and architectures for dependable embedded systems. In *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2011 Proceedings of the 9th International Conference on*, pages 69–78, Oct 2011.

[133] J. Henkel, A. Herkersdorf, L. Bauer, et al. Invasive manycore architectures. In *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, pages 193–200, Jan 2012.

[134] T. Higuchi, Y. Liu, and X. Yao, editors. *Evolvable Hardware*. Springer Science+Media LLC, New York, 2006.

[135] Mark D Hill and Michael R Marty. Amdahl's law in the multicore era. *Computer*, (7):33–38, 2008.

[136] Eric Hoffman, Peter Martin, Thomas Pütz, Aymeric Trzmiel, and Karim Zeghal. Airborne spacing: flight deck view of compatibility with continuous descent approach (CDA). *Interface,(September)*, pages 1–12, 2007.

[137] H. Hoffmann. CoAdapt: Predictable Behavior for Accuracy-Aware Applications Running on Power-Aware Systems. In *Real-Time Systems (ECRTS), 2014 26th Euromicro Conference on*, pages 223–232, July 2014.

[138] H. Hoffmann, M. Maggio, M.D. Santambrogio, A. Leva, and A. Agarwal. A generalized software framework for accurate and efficient management of performance goals. In *Embedded Software (EMSOFT), 2013 Proceedings of the International Conference on*, pages 1–10, Sept 2013.

[139] Henry Hoffmann, Jonathan Eastep, Marco D Santambrogio, Jason E Miller, and Anant Agarwal. Application heartbeats: a generic interface for specifying program performance and goals in autonomous computing environments. In *Proceedings of the 7th international conference on Autonomic computing*, pages 79–88. ACM, 2010.

[140] Henry Hoffmann, Martina Maggio, Marco D Santambrogio, Alberto Leva, and Anant Agarwal. Seec: A framework for self-aware computing. 2010.

[141] Henry Hoffmann, Martina Maggio, Marco D Santambrogio, Alberto Leva, and Anant Agarwal. SEEC: A framework for self-aware management of multicore resources. (MIT-CSAIL-TR-2011-016), 2011.

[142] Henry Hoffmann, Stelios Sidiroglou, Michael Carbin, et al. Dynamic knobs for responsive power-aware computing. In *ACM SIGPLAN Notices*, volume 46, pages 199–212. ACM, 2011.

[143] Kirak Hong, David Lillethun, Umakishore Ramachandran, Beate Ottenwälder, and Boris Koldehofe. Mobile Fog: A Programming Model for Large-scale Applications on the Internet of Things. In *Proceedings of the Second ACM SIGCOMM Workshop on Mobile Cloud Computing*, MCC '13, pages 15–20, New York, NY, USA, 2013. ACM.

[144] John E. Hopcroft and Richard M. Karp. A n5/2 algorithm for maximum matchings in bipartite. In *Switching and Automata Theory, 1971., 12th Annual Symposium on*, pages 122–125, 1971.

[145] Chien-Shu Hsieh. Robust two-stage Kalman filters for systems with unknown inputs. *Automatic Control, IEEE Trans. on*, 45(12):2374–2378, 2000.

[146] HT-Lab. Cpu86: 8088 fpga ip core, 2016.

[147] Wei Huang, S. Ghosh, S. Velusamy, et al. HotSpot: a compact thermal modeling methodology for early-stage VLSI design. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 14(5):501 –513, may 2006.

[148] Wei Huang, Mircea R Stant, Karthik Sankaranarayanan, Robert J Ribando, and Kevin Skadron. Many-core design from a thermal perspective. In *Proceedings of the 45th annual Design Automation Conference*, pages 746–749. ACM, 2008.

[149] IBM. *PowerPC Microprocessor Family: Vector/SIMD Multimedia Extension Technology Pro- gramming Environments Manual*. IBM, 2005.

[150] Toshiyuki INAGAKI. Design of human interactions with smart machines: Lessons learned from aircraft accidents. *The 4th IARP/IEEE RAS/EURON, keynote lecture, June 17, 2005 Nagoya*, 2005.

[151] Oracle Inc. OpenSPARC Processor. *Oracle Inc.*, 2013.

[152] Xilinx Inc. Xilinx University Program XUPV5-LX110T Development System. 2011.

[153] Xilinx Inc. Virtex-6 FPGA ML605 Evaluation Kit. 2013.

[154] Texas Instruments. Msp430x1xx family users guide. 2006.

[155] Engin Ïpek, Sally A. McKee, Rich Caruana, Bronis R. de Supinski, and Martin Schulz. Efficiently Exploring Architectural Design Spaces via Predictive Modeling. *SIGPLAN Not.*, 41(11):195–206, October 2006.

[156] Canturk Isci, Gilberto Contreras, and Margaret Martonosi. Live, runtime phase monitoring and prediction on real systems with application to dynamic power management. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 359–370. IEEE Computer Society, 2006.

[157] Canturk Isci and Margaret Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, page 93. IEEE Computer Society, 2003.

[158] ITRS. The International Technology Roadmap for Semiconductors [Online]. Available: http://public.itrs.net/. July 2011.

[159] ITRS. Process Integration, Devices, and Structures (PIDS). July 2011.

[160] Syed M. A. H. Jafri, Liang Guang, Axel Jantsch, et al. Self-Adaptive NoC Power Management with Dual-Level Agents: Architecture and Implementation. In *Proceedings of the Conference on Self-adaptive Netwotrked Embedded Systems*, Rome, Italy, February 2012.

[161] Axel Jantsch and Kalle Tammemäe. A Framework of Awareness for Artificial Subjects. In *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis*, CODES '14, pages 20:1–20:3, New York, NY, USA, 2014. ACM.

[162] JEDEC. *Failure Mechanisms and Models for Semiconductor Devices*. In JEDEC Publication JEP122-A, 2002.

[163] Jeff, B. Advances in big.LITTLE Technology for Power and Energy Savings. Technical report, ARM Ltd., 2012.

[164] Brendan Jennings and Rolf Stadler. Resource Management in Clouds: Survey and Research Challenges. *Journal of Network and Systems Management*, pages 1–53, 2014.

[165] Kwangok Jeong, Andrew B Kahng, and Kambiz Samadi. Impact of guardband reduction on design outcomes: A quantitative approach. *Semiconductor Manufacturing, IEEE Transactions on*, 22(4):552–565, 2009.

[166] William M Johnson. *Super-scalar processor design*. 1989.

[167] PJ Joseph, Kapil Vaswani, and Matthew J Thazhuthaveetil. Construction and use of linear regression models for processor performance analysis. In *High-Performance Computer Architecture, 2006. The Twelfth International Symposium on*, pages 99–108. IEEE, 2006.

[168] Russ Joseph, David Brooks, and Margaret Martonosi. Control techniques to eliminate voltage emergencies in high performance processors. In *High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings. The Ninth International Symposium on*, pages 79–90. IEEE, 2003.

[169] Hwisung Jung et al. A stochastic local hot spot alerting technique. In *Design Auto. Conf., 2008. ASPDAC 2008. Asia and South Pacific*, pages 468–473, 2008.

[170] Hermann Kaindl, Mathieu Vallée, and Edin Arnautovic. Self-Representation for Self-Configuration and Monitoring in Agent-Based Flexible Automation Systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 43(1):164–175, January 2013.

[171] Rudolf Emil Kalman. Mathematical description of linear dynamical systems. *Journal of the Society for Industrial & Applied Mathematics, Series A: Control*, 1(2):152–192, 1963.

[172] Eric Karl, David Blaauw, Dennis Sylvester, and Trevor Mudge. Reliability modeling and management in dynamic microprocessor-based systems. In *Proceedings of the 43rd annual Design Automation Conference*, pages 1057–1060. ACM, 2006.

[173] Eric Karl, Prashant Singh, D Blaauw, and D Sylvester. Compact in-situ sensors for monitoring negative-bias-temperature-instability effect and oxide degradation. In *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International*, pages 410–623. IEEE, 2008.

[174] Stefanos Kaxiras and Margaret Martonosi. Computer architecture techniques for power-efficiency. *Synthesis Lectures on Computer Architecture*, 3(1):1–207, 2008.

[175] J. Keane, Xiaofei Wang, D. Persaud, and C.H. Kim. An All-In-One Silicon Odometer for Separately Monitoring HCI, BTI, and TDDB. *Solid-State Circuits, IEEE Journal of*, 45(4):817–829, April 2010.

[176] John Keane, Tae-Hyoung Kim, and Chris H Kim. An on-chip NBTI sensor for measuring PMOS threshold voltage degradation. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 18(6):947–956, 2010.

[177] John Keane, Shrinivas Venkatraman, Paulo Butzen, and Chris H Kim. An array-based test circuit for fully automated gate dielectric breakdown characterization. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 19(5):787–795, 2011.

[178] John H Kelm, Daniel R Johnson, Matthew R Johnson, et al. Rigel: an architecture and scalable programming interface for a 1000-core accelerator. In *ACM SIGARCH Computer Architecture News*, volume 37, pages 140–151. ACM, 2009.

[179] John H Kelm, Daniel R Johnson, Steven S Lumetta, Matthew I Frank, and Sanjay J Patel. A task-centric memory model for scalable accelerator architectures. In *Parallel Architectures and Compilation Techniques, 2009. PACT'09. 18th International Conference on*, pages 77–87. IEEE, 2009.

[180] Jeffrey O Kephart and David M Chess. The vision of autonomic computing. *Computer*, 36(1):41 – 50, jan 2003.

[181] R.E. Kessler. The Alpha 21264 microprocessor. *Micro, IEEE*, 19(2):24 –36, mar/apr 1999.

[182] Kurt Keutzer, Jan M Rabaey, A Sangiovanni-Vincentelli, et al. System-level design: orthogonalization of concerns and platform-based design. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 19(12):1523 – 1543, dec 2000.

[183] Brucek Khailany, William J Dally, Ujval J Kapasi, et al. Imagine: Media processing with streams. *IEEE micro*, 21(2):35–46, 2001.

[184] Brucek K Khailany, Ted Williams, Jim Lin, et al. A programmable 512 GOPS stream processor for signal, image, and video processing. *Solid-State Circuits, IEEE Journal of*, 43(1):202–213, 2008.

[185] Usman A Khan and José MF Moura. Distributing the Kalman filter for large-scale systems. *Signal Processing, IEEE Transactions on*, 56(10):4919–4935, 2008.

[186] Chris H Kim, Kaushik Roy, Steven Hsu, Ram Krishnamurthy, and Shekhar Borkar. A process variation compensating technique with an on-die leakage current sensor for nanometer scale dynamic circuits. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 14(6):646–649, 2006.

[187] J. M. Kim, S. K. Seo, and S. W. Chung. Looking into heterogeneity: when simple is faster. In *The 2nd International Workshop on Parallelism in Mobile Platforms.*, 2014.

[188] Kyung Ki Kim, Wei Wang, and Ken Choi. On-chip aging sensor circuits for reliable nanometer MOSFET digital circuits. *Circuits and Systems II: Express Briefs, IEEE Transactions on*, 57(10):798–802, 2010.

[189] Myungsun Kim, Kibeom Kim, James R Geraci, and Seongsoo Hong. Utilization-aware load balancing for the energy efficient operation of the big. LITTLE processor. In *Proceedings of the conference on Design, Automation & Test in Europe*, page 223. European Design and Automation Association, 2014.

[190] Tae-Hyoung Kim, Randy Persaud, and Chris H Kim. Silicon odometer: An on-chip reliability monitor for measuring frequency degradation of digital circuits. *Solid-State Circuits, IEEE Journal of*, 43(4):874–880, 2008.

[191] Youngtaek Kim, Lizy Kurian John, Sanjay Pant, et al. AUDIT: Stress testing the automatic way. In *Microarchitecture (MICRO), 2012 45th Annual IEEE/ACM International Symposium on*, pages 212–223. IEEE, 2012.

[192] V.B. Kleeberger, C. Gimmler-Dumont, C. Weis, et al. A Cross-Layer Technology-Based Study of How Memory Errors Impact System Resilience. *Micro, IEEE*, 33(4):46–55, July 2013.

[193] Joonho Kong, Sung Woo Chung, and Kevin Skadron. Recent Thermal Management Techniques for Microprocessors. *ACM Comput. Surv.*, 44(3):13:1–13:42, June 2012.

[194] Georgios Kornaros and Dionisios Pnevmatikatos. A Survey and Taxonomy of On-chip Monitoring of Multicore Systems-on-chip. *ACM Trans. Des. Autom. Electron. Syst.*, 18(2):17:1–17:38, April 2013.

[195] David Koufaty, Dheeraj Reddy, and Scott Hahn. Bias scheduling in heterogeneous multi-core architectures. In *Proceedings of the 5th European conference on Computer systems*, EuroSys '10, pages 125–138, New York, NY, USA, 2010. ACM.

[196] Christoforos Kozyrakis and David Patterson. Vector vs. superscalar and VLIW architectures for embedded multimedia benchmarks. In *Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*, pages 283–293. IEEE Computer Society Press, 2002.

[197] Ronny Krashinsky, Christopher Batten, and Krste Asanović. Implementing the scale vector-thread processor. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 13(3):41, 2008.

[198] Ronny Krashinsky, Christopher Batten, Mark Hampton, et al. The vector-thread architecture. In *ACM SIGARCH Computer Architecture News*, volume 32, page 52. IEEE Computer Society, 2004.

[199] M. Kuhn and K. Johnson. *Applied Predictive Modeling*. SpringerLink : Bücher. Springer New York, 2013.

[200] Ari Kulmala, Erno Salminen, and Timo D Hämäläinen. Evaluating large system-on-chip on multi-FPGA platform. In *Embedded Computer Systems: Architectures, Modeling, and Simulation*, pages 179–189. Springer, 2007.

[201] R. Kumar, D.M. Tullsen, P. Ranganathan, N.P. Jouppi, and K.I. Farkas. Single-ISA heterogeneous multi-core architectures for multithreaded workload performance. In *Computer Architecture, 2004. Proceedings. 31st Annual International Symposium on*, pages 64 – 75, june 2004.

[202] Rakesh Kumar, Keith I Farkas, Norman P Jouppi, Parthasarathy Ranganathan, and Dean M Tullsen. Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction. In *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, pages 81–92. IEEE, 2003.

[203] Rakesh Kumar, Dean M. Tullsen, Norman P. Jouppi, and Parthasarathy Ranganathan. Heterogeneous Chip Multiprocessors. *Computer*, 38(11):32–38, November 2005.

[204] Ravi Kuppuswamy, Shankar R Sawant, Srikanth Balasubramanian, et al. Over one million TPCC with a 45nm 6-core Xeon® CPU. In *Solid-State Circuits Conference-Digest of Technical Papers, 2009. ISSCC 2009. IEEE International*, pages 70–71. IEEE, 2009.

[205] Robert Laddaga. Active Software. In *Self-Adaptive Software*, volume 1936 of *Lecture Notes in Computer Science*, pages 11–26. Springer, July 2001.

[206] Liangzhen Lai, Vikas Chandra, Robert Aitken, and Puneet Gupta. SlackProbe: A low overhead in situ on-line timing slack monitoring methodology. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, pages 282–287, 2013.

[207] Clemens J.M. Lasance. Thermally driven reliability issues in microelectronic systems: status-quo and challenges. *Microelectronics Reliability*, 43(12):1969 – 1974, 2003.

[208] Benjamin C Lee and David M Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *ACM SIGPLAN Notices*, volume 41, pages 185–194. ACM, 2006.

[209] Benjamin C Lee, Jamison Collins, Hong Wang, and David Brooks. CPR: Composable performance regression for scalable multiprocessor models. In *Microarchitecture, 2008. MICRO-41. 2008 41st IEEE/ACM International Symposium on*, pages 270–281. IEEE, 2008.

[210] E.A. Lee. Cyber Physical Systems: Design Challenges. In *ISORC, 2008*, pages 363 –369, may 2008.

[211] Edward A Lee. The past, present and future of cyber-physical systems: A focus on models. *Sensors*, 15(3):4837–4869, 2015.

[212] Walter Lee, Rajeev Barua, Matthew Frank, et al. Space-time scheduling of instruction-level parallelism on a raw machine. In *ACM SIGPLAN Notices*, volume 33, pages 46–57. ACM, 1998.

[213] Yunsup Lee, Andrew Waterman, Rimas Avizienis, et al. A 45nm 1.3 GHz 16.7 double-precision GFLOPS/W RISC-V processor with vector accelerators. In *European Solid State Circuits Conference (ESSCIRC), ESSCIRC 2014-40th*, pages 199–202. IEEE, 2014.

[214] Larkhoon Leem, Hyungmin Cho, Jason Bau, Quinn A Jacobson, and Subhasish Mitra. ERSA: Error resilient system architecture for probabilistic applications. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010*, pages 1560–1565. IEEE, 2010.

[215] C.R. Lefurgy, A.J. Drake, M.S. Floyd, et al. Active Guardband Management in Power7+ to Save Energy and Maintain Reliability. *Micro, IEEE*, 33(4):35–45, 2013.

[216] Peter R. Lewis, Arjun Chandra, Funmilade Faniyi, et al. Architectural Aspects of Self-aware and Self-expressive Computing Systems. *IEEE Computer*, August 2015.

[217] P.R. Lewis, A. Chandra, S. Parsons, et al. A Survey of Self-Awareness and Its Application in Computing Systems. In *Self-Adaptive and Self-Organizing Systems Workshops (SASOW), 2011 Fifth IEEE Conference on*, pages 102–107, October 2011.

[218] Sheng Li, Jung Ho Ahn, Richard D Strong, et al. McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, pages 469–480. IEEE, 2009.

[219] Sheng Li, Jung Ho Ahn, Richard D Strong, et al. The McPAT framework for multicore and manycore architectures: Simultaneously modeling power, area,

and timing. *ACM Transactions on Architecture and Code Optimization (TACO)*, 10(1):5, 2013.

[220] Tao Li and Lizy Kurian John. Run-time modeling and estimation of operating system power consumption. *SIGMETRICS Perform. Eval. Rev.*, 31(1):160–171, June 2003.

[221] Tuo Li, Muhammad Shafique, Jude Angelo Ambrose, et al. RASTER: runtime adaptive spatial/temporal error resiliency for embedded processors. In *Proceedings of the 50th Annual Design Automation Conference*, page 62. ACM, 2013.

[222] Y Li, Young Moon Kim, E Mintarno, D.S. Gardner, and S Mitra. Overcoming Early-Life Failure and Aging for Robust Systems. *Design Test of Computers, IEEE*, 26(6):28–39, 2009.

[223] Guangshuo Liu et al. Dynamic thread mapping for high-performance, power-efficient heterogeneous many-core systems. In *Computer Design (ICCD), 2013 IEEE 31st International Conference on*, pages 54–61, Oct 2013.

[224] Yang-Yu Liu, Jean-Jacques Slotine, and Albert-László Barabási. Controllability of complex networks. *Nature*, 473(7346):167–173, 2011.

[225] Yang-Yu Liu, Jean-Jacques Slotine, and Albert-László Barabási. Observability of complex systems. *Proceedings of the National Academy of Sciences*, 110(7):2460–2465, 2013.

[226] Lennart Ljung. *System identification*. Springer, 1998.

[227] Jieyi Long, Seda Ogrenci Memik, Gokhan Memik, and Rajarshi Mukherjee. Thermal Monitoring Mechanisms for Chip Multiprocessors. *ACM Trans. Archit. Code Optim.*, 5(2):9:1–9:33, September 2008.

[228] D. G. Luenberger. *Introduction to Dynamic Systems: Theory, Models, and Applications*. Wiley, 1979.

[229] Kai Ma, Xue Li, Ming Chen, and Xiaorui Wang. Scalable power control for many-core architectures running multi-threaded applications. In *ACM SIGARCH Computer Architecture News*, volume 39, pages 449–460. ACM, 2011.

[230] Martina Maggio, Henry Hoffmann, Marco D. Santambrogio, Anant Agarwal, and Alberto Leva. Decision Making in Autonomic Computing Systems: Comparison of Approaches and Techniques. In *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ICAC '11, pages 201–204, New York, NY, USA, 2011. ACM.

[231] R. Marculescu, U.Y. Ogras, Li-Shiuan Peh, N.E. Jerger, and Y. Hoskote. Outstanding Research Problems in NoC Design: System, Microarchitecture, and Circuit Perspectives. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 28(1):3–21, 2009.

[232] Benjamin LaSalle Meakin. *Multicore system design with xum: The extensible utah multicore project*. PhD thesis, The University of Utah, 2010.

[233] S.O. Memik, R. Mukherjee, Min Ni, and Jieyi Long. Optimizing Thermal Sensor Allocation for Microprocessors. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 27(3):516–527, 2008.

[234] Pietro Mercati, Andrea Bartolini, Francesco Paterna, Tajana Simunic Rosing, and Luca Benini. Workload and user experience-aware dynamic reliability management in multicore processors. In *Proceedings of the 50th Annual Design Automation Conference*, page 2. ACM, 2013.

[235] Pietro Mercati, Andrea Bartolini, Francesco Paterna, Tajana Simunic Rosing, and Luca Benini. A Linux-governor based Dynamic Reliability Manager for android mobile devices. In *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 1–4. IEEE, 2014.

[236] Pietro Mercati, Francesco Paterna, Andrea Bartolini, Luca Benini, and Tajana Simunic Rosing. Dynamic variability management in mobile multicore processors under lifetime constraints. In *Computer Design (ICCD), 32nd IEEE International Conference on*, pages 448–455, Oct 2014.

[237] Campbell Millar, David Reid, Gareth Roy, Scott Roy, and Asen Asenov. Accurate statistical description of random dopant-induced threshold voltage variability. *Electron Device Letters, IEEE*, 29(8):946–948, 2008.

[238] Sparsh Mittal. A survey of techniques for improving energy efficiency in embedded computing systems. *International Journal of Computer Aided Engineering and Technology*, 2014.

[239] Sparsh Mittal and Jeffrey S Vetter. A survey of CPU-GPU heterogeneous computing techniques. *ACM Computing Surveys (CSUR)*, 47(4):69, 2015.

[240] J.C. Mogul et al. Using Asymmetric Single-ISA CMPs to Save Energy on Operating Systems. *Micro, IEEE*, 28(3):26 –41, May-June 2008.

[241] R. Morales-Ramos, J.A. Montiel-Nelson, R. Berenguer, and A. Garcia-Alonso. Voltage Sensors for Supply Capacitor in Passive UHF RFID Transponders. In *Digital System Design: Architectures, Methods and Tools, DSD 2006. 9th EUROMICRO Conference on*, pages 625–629, 2006.

[242] Alain Morin. Levels of consciousness and self-awareness: A comparison and integration of various neurocognitive views. *Consciousness and Cognition*, 15(2):358 – 371, 2006.

[243] Lo. Motus, M. Meriste, and J. Preden. Towards Middleware Based Situation Awareness. In *Military Communications Conference (MILCOM)*, Boston, October 2009.

[244] Tiago Mück, Santanu Sarma, and Nikil Dutt. Run-DMC: runtime dynamic heterogeneous multicore performance and power estimation for energy efficiency. In *Proceedings of the 10th International Conference on Hardware/Software Codesign and System Synthesis*, pages 173–182. IEEE Press, 2015.

[245] Rajarshi Mukherjee and Seda Ogrenci Memik. Systematic temperature sensor allocation and placement for microprocessors. In *Proceedings of the 43rd annual Design Automation Conference*, pages 542–547. ACM, 2006.

[246] Sani R Nassif, Nikil Mehta, and Yu Cao. A resilience roadmap. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1011–1016. European Design and Automation Association, 2010.

[247] Ripal Nathuji and Karsten Schwan. Virtualpower: coordinated power management in virtualized enterprise systems. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 265–278. ACM, 2007.

[248] Ulric Neisser. The Roots of Self-Knowledge: Perceiving Self, It, and Thou. *Annals of the New York Academy of Sciences*, 818:19–33, June 1997.

[249] Mario Nemirovsky and Dean M Tullsen. Multithreading architecture. *Synthesis Lectures on Computer Architecture*, 8(1):1–109, 2013.

[250] S. T. S. Ngiap. Aemb 32-bit microprocessor core datasheet, 2007.

[251] Abdullah Nazma Nowroz, Ryan Cochran, and Sherief Reda. Thermal monitoring of real processors: techniques for sensor allocation and full characterization. In *Proceedings of the 47th Design Automation Conference*, DAC '10, pages 56–61, New York, NY, USA, 2010. ACM.

[252] NVidia. Variable SMP - a multi-core CPU architecture for low power and high performance. 2011.

[253] Katsuhiko Ogata. *Modern control engineering*. Prentice Hall PTR, 2001.

[254] Reza Olfati-Saber. Distributed Kalman filtering for sensor networks. In *Decision and Control, 2007 46th IEEE Conference on*, pages 5492–5498. IEEE, 2007.

[255] A. Olshevsky. The Minimal Controllability Problem. *ArXiv e-prints*, April 2013.

[256] Kunle Olukotun, Basem A Nayfeh, Lance Hammond, Ken Wilson, and Kunyung Chang. The case for a single-chip multiprocessor. *ACM Sigplan Notices*, 31(9):2–11, 1996.

[257] M. Omana, D. Rossi, N. Bosio, and C. Metra. Self-checking monitor for NBTI due degradation. In *Mixed-Signals, Sensors and Systems Test Workshop (IMS3TW), 2010 IEEE 16th International*, pages 1–6, June 2010.

[258] OpenCores. Altor32 - alternative lightweight openrisc cpu. 2016.

[259] OpenCores. Amber arm-compatible core. 2016. http://opencores.org/project,amber.

[260] OpenCores. pAVR. 2016.

[261] OpenCores.org. OpenRISC Processor. *OpenCores.org*, 2013.

[262] Oracle Inc. OpenSPARC T1 Microarchitecture Specification. 2006. Santa Clara, CA.

[263] Oracle Inc. OpenSPARC T2 Core Microarchitecture Specification. 2007. Santa Clara, CA.

[264] P. Oreizy, M.M. Gorlick, R.N. Taylor, et al. An architecture-based approach to self-adaptive software. *Intelligent Systems and their Applications, IEEE*, 14(3):54–62, May 1999.

[265] Organic Computing. Organic Computing Initiative. http://www.organic-computing.de/.

[266] Oroid. ODROID-XU3 Single Board Computer. 2014. http://www.hardkernel.com/main/products/prdt_info.php?g_code=G140448267127&tab_idx=2.

[267] Michael Orshansky, Linda Milor, and Chenming Hu. Characterization of spatial intrafield gate CD variability, its impact on circuit performance, and spatial mask-level correction. *Semiconductor Manufacturing, IEEE Transactions on*, 17(1):2–11, 2004.

[268] G. Palermo, C. Silvano, and V. Zaccaria. ReSPIR: A Response Surface-Based Pareto Iterative Refinement for Application-Specific Design Space Exploration. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 28(12):1816 –1829, dec. 2009.

[269] M. Pedram and S. Nazarian. Thermal Modeling, Analysis, and Management in VLSI Circuits: Principles and Methods. *Proceedings of the IEEE*, 94(8):1487 – 1501, aug. 2006.

[270] M. A P Pertijs, A. Niederkorn, Xu Ma, et al. A CMOS smart temperature sensor with a 3 sigma; inaccuracy of plusmn;0.5 deg;C from -50 deg;C to 120 deg;C. *Solid-State Circuits, IEEE Journal of*, 40(2):454–461, 2005.

[271] A.D. Pimentel, C. Erbas, and S. Polstra. A systematic approach to exploring embedded system architectures at multiple abstraction levels. *Computers, IEEE Transactions on*, 55(2):99 – 112, feb. 2006.

[272] M. Poirier. In Kernel Switcher: A solution to support ARM's new big.LITTLE technology. https://events.linuxfoundation.org/images/stories/slides/elc2013_poirier.pdf, 2013.

[273] R.; Culler D. Polastre, J.; Szewczyk. Telos: enabling ultra-low power wireless research. In *Information Processing in Sensor Networks, Fourth International Symposium on*, pages 364–369, Dept. of Comput. Sci., California Univ., Berkeley, CA, USA, June 2005.

[274] J. Preden, J. Llinas, G. Rogava, R. Pathma, and L. Motus. On-line data validation in distributed data fusion. In T. Pham, M. A. Kolodny, and K. L. Priddy, editors, *Ground/Air Multisensor Interoperability, Integration, and Networking for Persistent ISR IV: SPIE Defense, Security and Sensing*. SPIE - International Society for Optics and Photonics, 2013.

[275] Jurgo Preden. Generating Situation Awareness in Cyber-Physical Systems: Creation and Excahnge of Situational Information. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, New York, NY, USA, October 2014. ACM.

[276] Jürgo-Sören Preden. *Enhancing Situation-Awareness, Cognition and Reasoning of Ad-Hoc Network Agents*. PhD thesis, Tallinn University of Technology, June 2012. Thesis on informatics and system engineering C56.

[277] Jürgo-Sören Preden and J. Helander. Auto-adaptation Driven by Observed Context Histories. In *Proceedings of ECHISE (Exploiting Context Histories in Smart Environments) workshop at UbiComp*, Irvine, CA, 2006.

[278] Harald Psaier and Schahram Dustdar. A survey on self-healing systems: approaches and systems. *Computing*, 91(1):43–73, 2011.

[279] Z. W. Pylyshyn. *Computation and Cognition*. MIT Press, 2nd edition, 1984.

[280] Arun Raghavan, Yixin Luo, Anuj Chandawalla, et al. Computational sprinting. In *High Performance Computer Architecture (HPCA), 2012 IEEE 18th International Symposium on*, pages 1–12. IEEE, 2012.

[281] Krishna K Rangan, Gu-Yeon Wei, and David Brooks. Thread motion: fine-grained power management for multi-core systems. In *ACM SIGARCH Computer Architecture News*, volume 37, pages 302–313. ACM, 2009.

[282] Juri Ranieri, Alessandro Vincenzi, Amina Chebira, David Atienza, and Martin Vetterli. EigenMaps: algorithms for optimal thermal maps extraction and sensor placement on multicore processors. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, pages 636–641, New York, NY, USA, 2012. ACM.

[283] C. R Rao et al. *Linear Models and Generalizations: Least Squares and Alternatives*. Springer, 2008.

[284] Sherief Reda. Thermal and power characterization of real computing devices. *Emerging and Selected Topics in Circuits and Systems, IEEE Journal on*, 1(2):76–87, 2011.

[285] Sherief Reda, Ryan J Cochran, and Abdullah Nazma Nowroz. Improved thermal tracking for processors using hard and soft sensor allocation techniques. *Computers, IEEE Transactions on*, 60(6):841–851, 2011.

[286] Vijay Janapa Reddi and Meeta Sharma Gupta. *Resilient Architecture Design for Voltage Variation*. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2013.

[287] Vijay Janapa Reddi, Meeta Sharma Gupta, Glenn Holloway, et al. Voltage emergency prediction: Using signatures to reduce operating margins. In *High Performance Computer Architecture, IEEE 15th International Symposium on*, pages 18–29. IEEE, 2009.

[288] Vijay Janapa Reddi, Svilen Kanev, Wonyoung Kim, et al. Voltage noise in production processors. *IEEE micro*, (1):20–28, 2010.

[289] Vijay Janapa Reddi, Svilen Kanev, Wonyoung Kim, et al. Voltage Smoothing: Characterizing and Mitigating Voltage Noise in Production Processors via Software-Guided Thread Scheduling. In *MICRO*, pages 77–88, 2010.

[290] Vijay Janapa Reddi, David Z. Pan, Sani R. Nassif, and Keith A. Bowman. Robust and resilient designs from the bottom-up: Technology, CAD, circuit, and system issues. In *ASP-DAC*, pages 7–16, 2012.

[291] Semeen Rehman, Florian Kriebel, Duo Sun, Muhammad Shafique, and Jörg Henkel. dTune: Leveraging reliable code generation for adaptive dependability tuning under process variation and aging-induced effects. In *Proceedings of the The 51st Annual Design Automation Conference on Design Automation Conference*, pages 1–6. ACM, 2014.

[292] Simple RISC. Simply risc s1 core, 2016.

[293] E. Rotem et al. Temperature Measurement in the Intel Core Duo Processor. In *Proc. IntâĂŹl Workshop Thermal Investigations of ICs*, pages 23–27, 2006.

[294] Efraim Rotem, Alon Naveh, Doron Rajwan, Avinash Ananthakrishnan, and Eliezer Weissmann. Power-management architecture of the intel microarchitecture code-named sandy bridge. *IEEE Micro*, 32(2):0020–27, 2012.

[295] Michael Rubenstein, Christian Ahler, and Radhika Nagpal. Kilobot: A low cost scalable robot system for collective behaviors. In *Robotics and Automation (ICRA), IEEE International Conference on*, pages 3293–3298. IEEE, 2012.

[296] Michael Rubenstein, Alejandro Cornejo, and Radhika Nagpal. Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799, 2014.

[297] T.A. Runkler. *Data Analytics: Models and Algorithms for Intelligent Data Analysis*. SpringerLink : Bücher. Vieweg+Teubner Verlag, 2012.

[298] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 4(2):14, 2009.

[299] P. Salihundam, S. Jain, T. Jacob, et al. A 2 Tb/s 6 $\times$ 4 Mesh Network for a Single-Chip Cloud Computer With DVFS in 45 nm CMOS. *Solid-State Circuits, IEEE Journal of*, 46(4):757–766, April 2011.

[300] M.G. Sánchez-Escribano and Ricardo Sanz. Emotions and the engineering of adaptiveness. In *Procedia Computer Science: Conference on Systems Engineering Research*, volume 28, pages 473–480, Madrid, Spain, 2014. Elsevier.

[301] Marco D Santambrogio, Henry Hoffmann, Jonathan Eastep, and Anant Agarwal. Enabling technologies for self-aware adaptive systems. In *Adaptive Hardware and Systems (AHS), NASA/ESA Conference on*, pages 149–156. IEEE, 2010.

[302] Conor Santifort. Amber ARM-compatible core. *OpenCores.org*, 2010.

[303] Ricardo Sanz, Ignacio López, Manuel Rdoríguez, and Carlos Hernández. Principles for Consciousness in Integrated Cognitive Control. *Neural Networks*, 20(9), 11 2007.

[304] S. Sarma, T. Muck, M. Shoushtari, A. BanaiyanMofrad, and N. Dutt. Cross-layer virtual/physical sensing and actuation for resilient heterogeneous many-core SoCs. In *21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 395–402, Jan 2016.

[305] Santanu Sarma and Nikil Dutt. FPGA Emulation and Prototyping of a CyberPhysical-System-On-Chip (CPSoC). In *Proceedings of the International Symposium on Rapid System Prototyping (RSP)*, pages 121–127, New Delhi, India, October 2014.

[306] Santanu Sarma and Nikil Dutt. Minimal sparse observability of complex networks: Application to MPSoC sensor placement and run-time thermal estimation and tracking. In *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 1–6, March 2014.

[307] Santanu Sarma and Nikil Dutt. Cross-Layer Exploration of Heterogeneous Multicore Processor Configurations. In *VLSI Design (VLSID), 2015 28th International Conference on*, pages 147–152, Jan 2015.

[308] Santanu Sarma and Nikil Dutt. *Handbook of Hardware/Software Codesign*, chapter Architecture and Cross-layer Design Space Exploration of heterogeneous Muti-core Processors. Springer, to appread in 2016.

[309] Santanu Sarma, Nikil Dutt, P. Gupta, A. Nicolau, and N. Venkatasubramanian. On-Chip Self-Awareness Using Cyberphysical-Systems-On-Chip (CPSoC). In *Proceedings of the 12th International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, New Delhi, India, October 2014.

[310] Santanu Sarma, Nikil Dutt, P. Gupta, A. Nicolau, and N. Venkatasubramanian. CyberPhysical-System-On-Chip (CPSoC): A Self-Aware MPSoC Paradigm with Cross-Layer Virtual Sensing and Actuation. In *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 625–628. EDA Consortium, March 2015.

[311] Santanu Sarma, Nikil Dutt, and Nalini Venkatasubramanian. Cross-layer Virtual Observers for Embedded Multiprocessor System-on-chip (MPSoC). In *Proceedings of the 11th International Workshop on Adaptive and Reflective Middleware*, ARM '12, pages 4:1–4:7, New York, NY, USA, 2012. ACM.

[312] Santanu Sarma, Nikil Dutt, N. Venkatasubramaniana, A. Nicolau, and P. Gupta. CyberPhysical-System-On-Chip (CPSoC): Sensor-Actuator Rich Self-Aware Computational Platform. Technical report, Center for Embedded Computer Systems University of California, Irvine, CA 92697-2620, USA, May 2013. CECS Technical Report No: CECS-TR-13-06.

[313] Santanu Sarma, T. Muck, L. A.D. Bathen, N. Dutt, and A. Nicolau. SmartBalance: A Sensing-Driven Linux Load Balancer for Energy Efficiency of Heterogeneous MPSoCs. In *Proceedings of the 52nd Annual Design Automation Conference (DAC'15)*, Jun 2015.

[314] Ichiro Satoh. A Framework for Data Processing at the Edges of Networks. In *Database and Expert Systems Applications*, pages 304–318. 2013.

[315] Henry Scheffé et al. Review: H. Cramér, Mathematical methods of statistics. *Bulletin of the American Mathematical Society*, 53(7):733–735, 1947.

[316] F. Sebastiano, L.J. Breems, K. A A Makinwa, et al. A 1.2V 10 μ W NPN-Based Temperature Sensor in 65-nm CMOS With an Inaccuracy of 0.2° C (3 σ) From $-70^\circ C$ to $125^\circ C$. *Solid-State Circuits, IEEE Journal of*, 45(12):2591–2601, 2010.

[317] Lattice Semiconductor. Latticemico32 open, free 32-bit soft pro- cessor, 2016.

[318] Muhammad Shafique, Siddharth Garg, Tulika Mitra, Sri Parameswaran, and Jörg Henkel. Dark silicon as a challenge for hardware/software co-design: invited special session paper. In *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis*, page 13. ACM, 2014.

[319] Muhammad Shafique and Jörg Henkel. Agent-based distributed power management for kilo-core processors. In *Proceedings of the International Conference on Computer-Aided Design*, pages 153–160. IEEE Press, 2013.

[320] Muhammad Shafique, Benjamin Vogel, and Jörg Henkel. Self-adaptive hybrid dynamic power management for many-core systems. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 51–56. EDA Consortium, 2013.

[321] Michael W. Shapiro. Self-Healing in Modern Operating Systems. *Queue*, 2(9):66–75, Dec 2004.

[322] Daniel Shelepov et al. HASS: a scheduler for heterogeneous multicore systems. *SIGOPS Oper. Syst. Rev.*, 43(2):66–75, April 2009.

[323] Sang Shengfeng, Zhang Dexue, and Yu Guoping. SoC Verification Platform Based on AEMB Softcore Processor [J]. *Microcontrollers & Embedded Systems*, 4:016, 2010.

[324] T. Sherwood, E. Perelman, G. Hamerly, S. Sair, and B. Calder. Discovering and exploiting program phases. *Micro, IEEE*, 23(6):84 – 93, nov.-dec. 2003.

[325] Timothy Sherwood, Suleyman Sair, and Brad Calder. Phase tracking and prediction. In *Proceedings of the 30th annual international symposium on Computer architecture*, ISCA '03, pages 336–349, New York, NY, USA, 2003. ACM.

[326] David J Sheskin. *Handbook of parametric and nonparametric statistical procedures*. crc Press, 2003.

[327] Jun Yong Shin, Fadi Kurdahi, and Nikil Dutt. Cooperative On-Chip Temperature EstimationUsing Multiple Virtual Sensors. *Embedded Systems Letters, IEEE*, 7(2):37–40, 2015.

[328] Victor Shnayder, Bor-rong Chen, Konrad Lorincz, Thaddeus RF Fulford Jones, and Matt Welsh. Sensor networks for medical care. In *SenSys*, volume 5, pages 314–314, 2005.

[329] R.H. Shumway. *Applied statistical time series analysis*. Number v. 1 in Prentice-Hall series in statistics. Prentice-Hall, 1988.

[330] Dan Simon. *Optimal state estimation: Kalman, H infinity, and nonlinear approaches*. Wiley. com, 2006.

[331] Karan Singh, Major Bhadauria, and Sally A. McKee. Real Time Power Estimation and Thread Scheduling via Performance Counters. *SIGARCH Comput. Archit. News*, 37(2):46–55, July 2009.

[332] Prashant Singh, Eric Karl, David Blaauw, and Dennis Sylvester. Compact degradation sensors for monitoring nbti and oxide degradation. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 20(9):1645–1655, 2012.

[333] A. Singhee and R. Rutenbar. *Extreme Statistics in Nanoscale Memory Design*. Springer, 2010.

[334] Kevin Skadron et al. Temperature-aware microarchitecture. *SIGARCH Comput. Archit. News*, 31(2):2–13, May 2003.

[335] Kevin Skadron, Mircea R. Stan, Karthik Sankaranarayanan, et al. Temperature-aware microarchitecture: Modeling and implementation. *ACM Trans. Archit. Code Optim.*, 1(1):94–125, March 2004.

[336] Brian Cantwell Smith. *Procedural reflection in programming languages*. PhD thesis, Massachusetts Institute of Technology, 1982.

[337] Brinkley Sprunt. The basics of performance-monitoring hardware. *Micro, IEEE*, 22(4):64–71, 2002.

[338] Ranjani Sridharan, Nikhil Gupta, and Rabi Mahapatra. Feedback-controlled reliability-aware power management for real-time embedded systems. In *45th ACM/IEEE Design Automation Conference*, pages 185–190. IEEE, 2008.

[339] Jayanth Srinivasan, Sarita V Adve, Pradip Bose, and Jude A Rivers. The case for lifetime reliability-aware microprocessors. In *ACM SIGARCH Computer Architecture News*, volume 32, page 276. IEEE Computer Society, 2004.

[340] S.Sarma and N.Dutt. CPSoClib: An FPGA-based design library for Cyberphysical-System-on-Chip (CPSoC) Prototyping and Emulation. Technical Report CECS-TR, Univeristy of California Irvine, 2014.

[341] S.Sarma, N.Dutt, and P.Gupta. Strength of Diversity: Exploiting Cheap Heterogeneous Noisy Sensors for Accurate Full-Chip Thermal Estimation. Technical Report CECS-TR-14-011, Univeristy of California Irvine, Jan 2014.

[342] Ivan Stoianov, Lama Nachman, Steve Madden, Timur Tokmouline, and M Csail. PIPENET: A Wireless Sensor Network for Pipeline Monitoring. In *Information Processing in Sensor Networks, 6th International Symposium on*, pages 264–273, Imperial Coll. London, London, Nov. 2007 2007.

[343] V. Stopjaková, H. Manhaeve, and M. Sidiropulos. On-chip Transient Current Monitor for Testing of Low-voltage CMOS IC. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '99, New York, NY, USA, 1999. ACM.

[344] Sujesha Sudevalayam and Purushottam Kulkarni. Energy harvesting sensor nodes: Survey and implications. *Communications Surveys & Tutorials, IEEE*, 13(3):443–461, 2011.

[345] Jin Sun, Avinash Kodi, Ahmed Louri, and Janet Meiling Wang. NBTI aware workload balancing in multi-core systems. In *Quality of Electronic Design, ISQED'09*, pages 833–838. IEEE, 2009.

[346] Jin Sun, Rui Zheng, Jyothi Velamala, et al. A Self-tuning Design Methodology for Power-efficient Multi-core Systems. *ACM Trans. Des. Autom. Electron. Syst.*, 18(1):4:1–4:24, January 2013.

[347] Dennis Sylvester, David Blaauw, and Eric Karl. Elastic: An adaptive self-healing architecture for unpredictable silicon. *Design & Test of Computers, IEEE*, 23(6):484–490, 2006.

[348] S. Sze. Using thermal diodes in the powerpc970MP processor. *IBM White Paper*, 2006.

[349] James Tandon. The openrisc processor: open hardware and linux. *Linux Journal*, (212):6, 2011.

[350] Michael Bedford Taylor. A landscape of the new dark silicon design regime. *Micro, IEEE*, 33(5):8–19, 2013.

[351] Michael Bedford Taylor, Jason Kim, Jason Miller, et al. The Raw microprocessor: A computational fabric for software circuits and general-purpose programs. *Micro, IEEE*, 22(2):25–35, 2002.

[352] Michael Bedford Taylor, Walter Lee, Jason Miller, et al. Evaluation of the Raw microprocessor: An exposed-wire-delay architecture for ILP and streams. *ACM SIGARCH Computer Architecture News*, 32(2):2, 2004.

[353] Jürgen Teich et al. Invasive Computing: An Overview. pages 241–268. Springer, Berlin, Heidelberg, 2011.

[354] E. Thelen and L. B. Smith. *A Dynamic Systems Approach to the Development of Cognition and Action*. Bradford Books Series in Cognitive Psychology. MIT Press, Cambridge, Massachusetts, 1994.

[355] William Thies, Michal Karczmarek, and Saman Amarasinghe. StreamIt: A language for streaming applications. In *Compiler Construction*, pages 179–196. Springer, 2002.

[356] J.A. Tropp et al. Signal Recovery From Random Measurements Via Orthogonal Matching Pursuit. *Information Theory, IEEE Trans. on*, 53(12):4655–4666, 2007.

[357] UC Berkeley Architecture Research. The berkeley out-of- order risc-v processor, 2016.

[358] UC Berkeley Architecture Research. Rocket core, 2016.

[359] Inna Vaisband and Eby G Friedman. Energy efficient adaptive clustering of on-chip power delivery systems. *INTEGRATION, the VLSI journal*, 48:1–9, 2015.

[360] Kenzo Van Craeynest, Aamer Jaleel, Lieven Eeckhout, Paolo Narvaez, and Joel Emer. Scheduling Heterogeneous Multi-Cores through Performance Impact Estimation (PIE). ISCA'12, 2012.

[361] Akshat Verma, Puneet Ahuja, and Anindya Neogi. pMapper: power and migration cost aware application placement in virtualized systems. In *Middleware*, pages 243–264. Springer, 2008.

[362] D. Vernon, G. Metta, and G. Sandini. A Survey of Artificial Cognitive Systems: Implications for the Autonomous Development of Mental Capabilities in Computational Agents. *Evolutionary Computation, IEEE Transactions on*, 11(2):151–180, April 2007.

[363] Ram Viswanath et al. Thermal Performance Challenges from Silicon to Systems, 2000.

[364] Nicholas J Wang and Sanjay J Patel. ReStore: Symptom-based soft error detection in microprocessors. *Dependable and Secure Computing, IEEE Transactions on*, 3(3):188–201, 2006.

[365] Tianhan Wang, Degang Chen, and R. Geiger. Multi-site on-chip current sensor for electromigration monitoring. In *Circuits and Systems (MWSCAS), 2011 IEEE 54th International Midwest Symposium on*, pages 1–4, Aug 2011.

[366] Wenping Wang, Shengqi Yang, Sarvesh Bhardwaj, et al. The impact of NBTI on the performance of combinational and sequential circuits. In *Proceedings of the 44th annual Design Automation Conference*, pages 364–369. ACM, 2007.

[367] Wenping Wang, Shengqi Yang, Sarvesh Bhardwaj, et al. The impact of NBTI effect on combinational circuit: modeling, simulation, and analysis. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 18(2):173–183, 2010.

[368] Xiaorui Wang and Yefu Wang. Coordinating Power Control and Performance Management for Virtualized Server Clusters. *Parallel and Distributed Systems, IEEE Transactions on*, 22(2):245–259, Feb 2011.

[369] Yefu Wang, Kai Ma, and Xiaorui Wang. Temperature-constrained power control for chip multiprocessors with online model estimation. In *ACM SIGARCH computer architecture news*, volume 37, pages 314–324. ACM, 2009.

[370] L. Wanner, C. Apte, R. Balani, P. Gupta, and M. Srivastava. Hardware Variability-Aware Duty Cycling for Embedded Sensors. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 21(6):1000–1012, 2013.

[371] Malcolm Ware, Karthick Rajamani, Michael Floyd, et al. Architecting for power management: the IBM® Power7™ approach. In *High Performance Computer Architecture (HPCA), IEEE 16th International Symposium on*, pages 1–11. IEEE, 2010.

[372] Robert NM Watson, Jonathan Woodruff, David Chisnall, et al. Bluespec Extensible RISC Implementation: BERI Hardware reference. 2014.

[373] David Wentzlaff, Patrick Griffin, Henry Hoffmann, et al. On-chip interconnection architecture of the tile processor. *IEEE micro*, (5):15–31, 2007.

[374] Wayne Wolf, Ahmed Amine Jerraya, and Grant Martin. Multiprocessor System-on-Chip (MPSoC) technology. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 27(10):1701–1713, 2008.

[375] Aeste Works. Aemb multi-threaded 32-bit embedded core family, 2016.

[376] Qiang Wu, Philo Juang, Margaret Martonosi, and Douglas W Clark. Formal online methods for voltage/frequency control in multiple clock domain microprocessors. *ACM SIGARCH Computer Architecture News*, 32(5):248–259, 2004.

[377] Qiang Wu, Margaret Martonosi, Douglas W Clark, et al. A dynamic compilation framework for controlling microprocessor energy and performance. In *Proceedings of the 38th annual IEEE/ACM International Symposium on Microarchitecture*, pages 271–282. IEEE Computer Society, 2005.

[378] Weidan Wu and Benjamin C Lee. Inferred models for dynamic and sparse hardware-software spaces. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 413–424. IEEE Computer Society, 2012.

[379] Xiaoxia Wu, Jian Li, Lixin Zhang, et al. Hybrid cache architecture with disparate memory technologies. In *ACM SIGARCH computer architecture news*, volume 37, pages 34–45. ACM, 2009.

[380] Qing Xie, Mohammad Javad Dousti, and Massoud Pedram. Therminator: A thermal simulator for smartphones producing accurate chip and skin temperature maps. In *Low Power Electronics and Design (ISLPED), IEEE/ACM International Symposium on*, pages 117–122. IEEE, 2014.

[381] Teng Yang, Seongjong Kim, Peter R Kinget, and Mingoo Seok. Compact and Supply-Voltage-Scalable Temperature Sensors for Dense On-Chip Thermal Monitoring. *Solid-State Circuits, IEEE Journal of*, 50(11):2773–2785, 2015.

[382] Chunhua Yao, Kewal K Saluja, and Parmesh Ramanathan. Calibrating On-chip Thermal Sensors in Integrated Circuits: A Design-for-Calibration Approach. *Journal of Electronic Testing*, 27(6):711–721, 2011.

[383] X. Yao and T. Higuchi. Promises and Challenges of Evolvable Hardware. *IEEE Transactions on Systems*, 29(1):87–97, February 1999.

[384] Juan Ye, Simon Dobson, and Susan McKeever. Situation Identification Techniques in Pervasive Computing: A Review. *Pervasive and Mobile Computing*, 8(1):36–66, February 2012.

[385] H. Zakaria, E. Yahya, and L. Fesquet. Self-Adaption in SoCs. In Phan Cong-Vinh, editor, *Autonomic Networking-on-Chip - Bio-Inspired Specification, Development, and Verification*, chapter 8. CRC Press, December 2011.

[386] Lenka Zdeborová and Marc Mézard. The number of matchings in random graphs. *Journal of Statistical Mechanics: Theory and Experiment*, 2006(05):P05003, 2006.

[387] Yufu Zhang et al. Accurate temperature estimation using noisy thermal sensors. In *Proc. of the 46th Annual Design Auto. Conf.*, DAC '09, pages 472–477, New York, NY, USA, 2009. ACM.

[388] Yufu Zhang and Ankur Srivastava. Accurate temperature estimation using noisy thermal sensors for Gaussian and non-Gaussian cases. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 19(9):1617–1626, 2011.

[389] T. Zidenberg, I Keslassy, and U. Weiser. Optimal Resource Allocation with MultiAmdahl. *Computer*, 46(7):70–77, July 2013.

[390] Tsahee Zidenberg, Isaac Keslassy, and Uri Weiser. MultiAmdahl: How Should I Divide My Heterogenous Chip? *Computer Architecture Letters*, 11(2):65–68, 2012.