Dynamic Power Aware Packet Processing with CMP

Zhen Ma Weifeng Zhang
Department of Computer Science and Engineering
University of California, San Diego
{zhma,weifeng}@cs.ucsd.edu

Abstract

Network processors implemented as systems-on-chip with multiple processors and peripherals offer a reliable means of scaling network with high link capacities. As more and more co-processors and peripherals are integrated, the power requirement also dramatically increases. Therefore it is essential to efficiently parallelize the subsystems to maximize the packet processing capacities while maintaining low power consumption.

In this paper, we propose a power aware packet processing architecture with chip-multiprocessor (CMP), which consists of a number of processor clusters (or arrays). Each array includes a number of identical processor cores, and processor cores between different arrays have different performance and power consumption. Only one array of processors is active at any time. We devise a simple policy to select a proper array of processors to lower the power consumption while still meeting the QoS requirements. Our simulation results show that the proposed CMP model has an approximately 40% power reduction compared to the CMP without power management, and an 11% power improvement compared to the symmetric CMP approach.

1 Introduction

The various emerging network applications, such as VoIP, IPv4/IPv6 gateways, software routers, VPN, intrusion detection, stimulate designs of new packet processing systems, which require both high packet processing capacities and programmable flexibility. Traditional ASICs lack programmable flexibility, and general-purpose processors cannot deal with Gigabit link capacities. Network processors emerge as the solution by integrating multiprocessors on chip and exploiting aggressive parallelism inherent in the network workload. However, the trend to integrate more and more functionality on the same silicon die, as well as the continuous exponential growth of link capacities, makes the power consumption of the network processors a challenging issue.

[1]This work was done in Spring 2004.

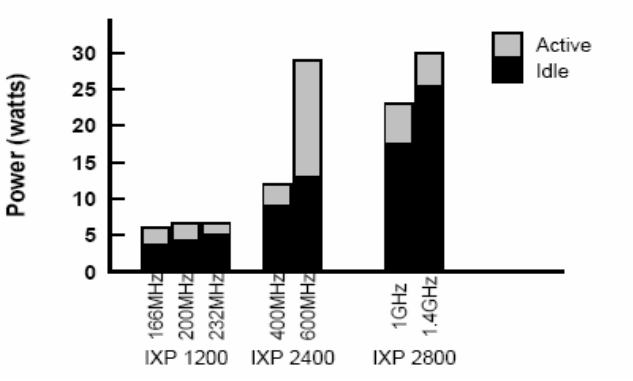

Figure 1. Power requirements for IXP series(from [8])

Figure 1 shows the trends of the power requirements as the processor functionality increases in Intel IXP series [6]. With dense integration and high performance requirement, it is certainly a design challenge to meet the tight system power budget.

Network processors are usually designed to target the maximum traffic loads. However, internet traffic analysis reveals that the packet traffic fluctuates significantly over time. The common traffic load is often substantially lower than the expected peak load, and the traffic shows bursty behaviors on all time scales [3, 13, 19]. In addition to internet traffic load, there are various network traffic patterns. For example, a LAN or a home network router may have low traffic most of time, with only sparse spikes of bursty behaviors.

With the emergence of a rich set of network applications, such as voice over IP and security applications, network processor performance is also limited by the memory latencies. Processors may spend a large portion of time waiting for data from lower memory hierarchy.

The above opportunities have been explored in the circuit level to achieve power efficient packet processing. There is also a large space to explore in architectural and OS levels.

In this project, we propose a novel CMP model for power aware packet processing. The CMP architecture consists of

a number of processor clusters (arrays), each of which has a number of identical processor cores. The processor cores in different arrays are geared for different level of performance and power requirement. At any time, only one array of cores is active, and the processor array is activated / deactivated based on the traffic load and other constraints (such as throughput, latency, etc). We devise a simple switching policy to use the low performance/power array of cores when the traffic is low and the high performance/power when the traffic is heavy. The high performance array is capable of handling the peak/bursty traffic load, and the low performance array for the non-peak load to reduce the power consumption.

This paper is organized as follows: section 2 briefly describes the existing power reduction techniques. Section 3 elaborates the proposed CMP model for power aware packet processing, followed by the simulation implementation details in section 4. Section 5 gives the simulation results of power/performance, and section 6 concludes the paper.

2 Related Work

Existing power reduction techniques fall in three categories, ranging from the microarchitectural level power management (fine-grained) to multi-core power management (coarse-grained).

2.1 Microarchitectural level power management

There are lots of research on lower power interconnection networks [16, 5] and low power processor design at the microarchitecture level. The idle functional units (such as instruction window) in the processor core are put into the sleeping state [15], or the cache component is set to drowsy state [4] to reduce static power leakage. Another approach is to improve memory hierarchy to save power, such as TCAM, which is content addressable memories performing exactly the fully-parallel search for IP-lookup; IPStash [7] is proposed as a memory architecture similar to set-associative caches but enhanced with mechanisms to facilitate IP-lookup and in particular longest prefix match.

2.2 Dynamic voltage scaling

Dynamic Voltage Scaling (DVS) [12] is widely applied in embedded systems with strict power constraints and real time constraints. The main idea of DVS is to scale down the core voltage and frequency with adaptation to runtime resource consumption. The GRACE-OS project [17] proposes cross layer energy aware DVS for mobile multimedia systems. In [18], Zhai et al show the optimal operating voltage should be scaled to approximately 30% of the maximum for typical workloads for energy efficiency.

2.3 Dynamic multi-core power management

Dynamic shutting down processors based on their workload is a natural approach to achieve energy efficiency for multiprocessor architectures. Kokku et al [8] propose an energy aware architecture for packet processing. In their architecture, there are multiple identical processor cores. According to traffic load, they dynamically shutdown or activate the processor engines. By reducing the number of working processor engines when dealing with light traffic load, they show significant power savings. Similarly, Nikitovic et al [10] also provide adaptive shutdown scheduling strategies in CMP for mobile devices. But their focus is on general mobile applications, instead of packet processing. Kumar et al [9] propose a single-ISA heterogeneous multi-core architecture, which switches between processors of different performance to save power consumption when executing a single-threaded application. Their experiment shows power savings of a factor of three with little performance loss.

Our CMP architecture extends Kumar's approach. Instead of a sequence of processor cores with different power/performance, we have variable power/performance clusters, and each cluster includes identical cores. This extended architecture aims to provide high processing capacity to handle peak network traffic loads, while allowing power saving when the traffic is low.

3 Power Aware CMP Model

There are two existing CMP models for packet processing. The first model has a control core with a number of high performance microengines. A typical example is the Intel IXP2800 network processor [6]. The processor contains a 700 MHz XScale core and sixteen 1.4GHz high performance microengines which support both DVS and DPM. However, to the best of our knowledge, there is no existing implementation for DVS or DPM on IXP series yet. The second model is a theoretical model provided in [8], which contains a number of identical processors. This symmetric CMP dynamically changes the number of running processors according to the packet load to reduce power consumption.

In this paper, we present a third type of power aware CMP model to process network packets, as illustrated in Figure 2. For simplicity, we assume it contains two arrays of processor cores. Each array includes two homogeneous processor cores, but two arrays have different performance / power consumption. For example, the high performance array can be two Alpha 21464 (250Watt, die size 350mm2), and the low performance array can be two Alpha 21064 (5 Watt, 1 die size 2.87mm2). Only one array of cores is active at any time. The processors in the active array monitor their own processing latency, response time, throughput, and the packet buffer queue utilization. When the workload becomes light (or heavy), the active array transfers the control to the low (high) performance array, which in turn shuts down the active array. Be-

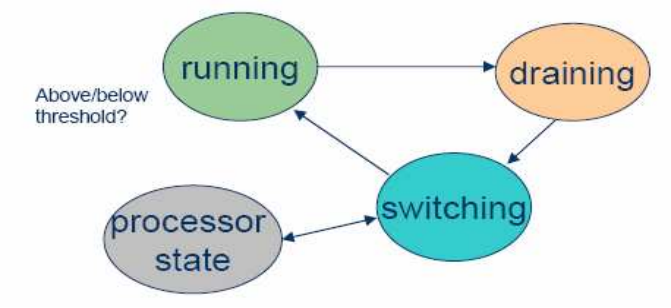

Figure 3. The state machine for the proposed pwoer aware CMP model

cause of the different power consumption on the CMP cores, the dynamic peer-to-peer processor migration should reduce the power consumption while still sustaining the QoS requirements.

The incoming packet stream is buffered in the packet queue (FIFO). When the event of low packet queue utilization occurs, it triggers the system to switch the low performance array. Likewise, it switches to the high performance array with the high queue utilization event. To ensure the packet queue overflow (packets dropped), the proper utilization threshold to trigger the switching event is essential.

We devise a state machine to control the transition between different arrays. The processor cores in an array are in the same state. Figure 3 shows the state machine for our proposed CMP model. There are three states in the simulator.

- Running: the packet processing application reads the packets from the queue and processes them. It does transition to the draining state when the queue threshold event occurs.
- Draining: stop fetching instructions and continue execution of instruction in the instruction queue. If the processor pipeline is in the error recovery (such as branch misprediction), the error recovery takes the priority. The length of draining state largely depends on the execution latency from the existing instructions. Meanwhile, the new array can be wakeup to reduce the startup latency. When the pipeline is empty, it does transition to the switching state.
- Switching: activate the new array, shut down the old array, and start fetching and execution from where it left.

Initially, the system starts up with the high performance array in the running state, and it is expected to stay in the low performance array most of time. If the packet queue underflows (empty), the packet processing application receives an empty packet. The application could pause to reduce execution power, or the low performance array could be clock-gated to reduce power even further. We don't simulate the clock gating in this project; instead, we let the application drop the empty packet and keep reading for new packets.

4 Simulation Implementation

We modify SimpleScalar 3.0 [2] to model our multi-array CMP architecture. And the power consumption is modeled using Wattch [1]. The different cores are modeled with different cache sizes, fetching/execution bandwidths, and die sizes.

4.1 Packet queue manager

In order to simulate packet processing, we also simulate the functions of the hardware queue manager. The queue manager checks the packet time stamp to emulate the true packet arrival rate. Upon packet arrival, the queue manager buffers the packet if the queue is no full; otherwise, the packet is dropped.

In this project, we have the queue with the fixed length of 512 entries.

4.2 Primitives

We implement three primitives for the communication between our simulator and the IP lookup application: read queue, notify queue ready, notify client done. The primitives are implemented by overloading special NOP instructions.

- Read queue: fetch packets from the queue. It passes a buffer pointer from the user address space to the simulator using the designated register (R2), and the simulator writes packets into the buffer.
- Notify queue ready: It tells the queue manager the application is ready to fetch packets. This is only needed for simulation purpose, serving as the mechanism to synchronize the application and the queue manager because the application needs to load a huge IP lookup table before it can starts processing packets.
- Notify client done: Only needed for simulation in case we only want to process a finite number of packets.

4.3 IP lookup application program

We develop an IP lookup program running on the proposed CMP model. Basically we apply the PATRICIA tree data structure to implement longest prefix matching for packet routing (IP lookup). PATRICIA tree is a compact representation of a trie where all nodes with one child are merged with their parent, which is the basic structure for IP lookup application.

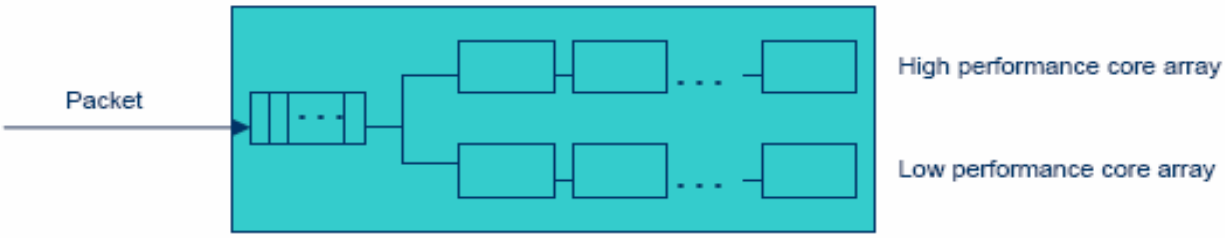

Figure 2. The power aware CMP model

4.4 Traffic traces

We extract the source IP, destination IP, timestamp information from a realistic internet routing trace (1.03M entries) [11] from the San Diego Supercomputing Center. We also retrieve the corresponding routing table (source IP, next hop) (157K entries) from the RIPE routing information database [14] and implement routing table using PATRICIA tree.

Based on the realistic traffic trace, we devise two traffic patterns: realistic traffic pattern (Figure 4) and synthetic traffic pattern (Figure 5). The realistic one is strictly the same as the packet trace we have obtained from the internet trace, while the synthetic trace is based on the realistic trace, but have more periods of low traffic load and more bursty behaviors.

The goal for the synthetic trace is to simulate traffic patterns for other network applications in addition to Internet traffic. For example, for a home network router, it does have more low traffic periods during workdays, while the users are out for work, but much higher traffic load during the evening and weekends.

5 Power/Performance Evaluation

In this section, we evaluate the costs, effectiveness, and performance of our dynamic prefetching technique. Performance improvement is relative to the baseline architecture, whose performance is shown in Figure .

5.1 Overhead of the Dynamic Prefetch Optimizer

In the section, we describe the detailed core configuration and the simulation results for both the realistic and synthetic traces. We define performance metrics for performance comparison and analyze the obtained power savings.

5.2 Core configuration and power/performance difference

The core configuration is shown in Table 1. The high performance array has two identical fast cores, while the low performance array has two identical slow cores. They have different L1 cache size, bandwidth, frequency (frequency could also be different for more power savings), die size (the low performance array will only add a very small portion of area and cost), and different power scaling factors (from Wattch). We then use Wattch to evaluate the power consumption of these two types of cores. We find that the power consumption of the fast cores is about 4 times of that of the slow cores.

Configuration	Fast cores	Slow cores
L1 Cache	16KB	4KB
Fetch bandwidth Issue	8	4
Core frequency	733MHz	733MHz
Die size	18mm	1mm
Power consumption	217 Watt	54 Watt

Table 1. CMP core configurations

Program	Instructions per second (IPC) for fast cores	Instructions per second (IPC) for slow cores
bzip2	0.8139	0.6981
crafty	0.9734	0.7815
eon	1.0987	0.8673
gap	1.1379	0.893
gzip	1.9602	1.4198
mcf	0.04	0.0399
parser	0.5976	0.5133
perlbmk	2.2856	1.4451
twolf	0.5821	0.5182
vortex	1.058	0.8364
vpr	0.594	0.5356
average	1.0129	0.7771

Table 2. Performance of SPEC2000 benchmarks

We also run the SPEC2000 benchmark on SimpleScalar 3.0 for the two types of cores respectively, and find that in terms of performance, the fast cores is approximately 30% faster than the slow ones (Table 2).

5.3 Performance

We run the CMP simulator to process 100,000 packets from both the realistic traffic trace and synthetic traffic trace in three modes (slow cores only, fast cores only, and switching-enabled only) respectively. We evaluate the performance

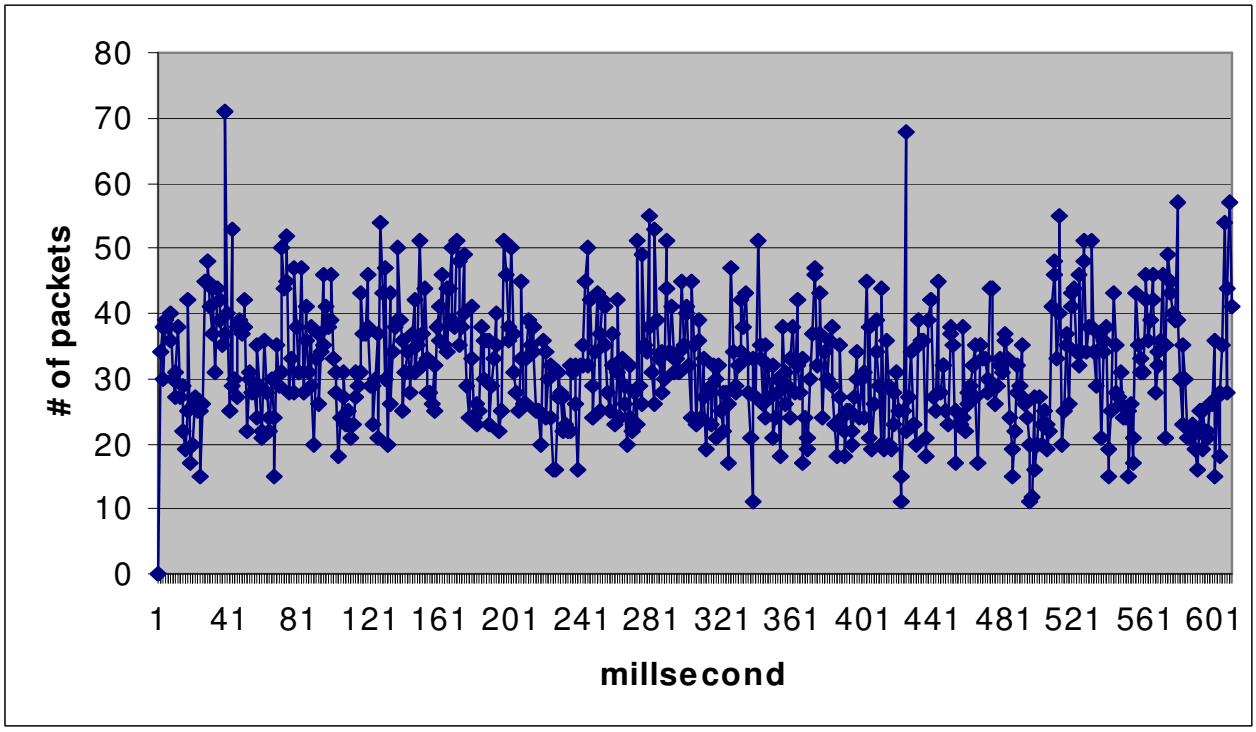

Figure 4. Realistic network traffic pattern

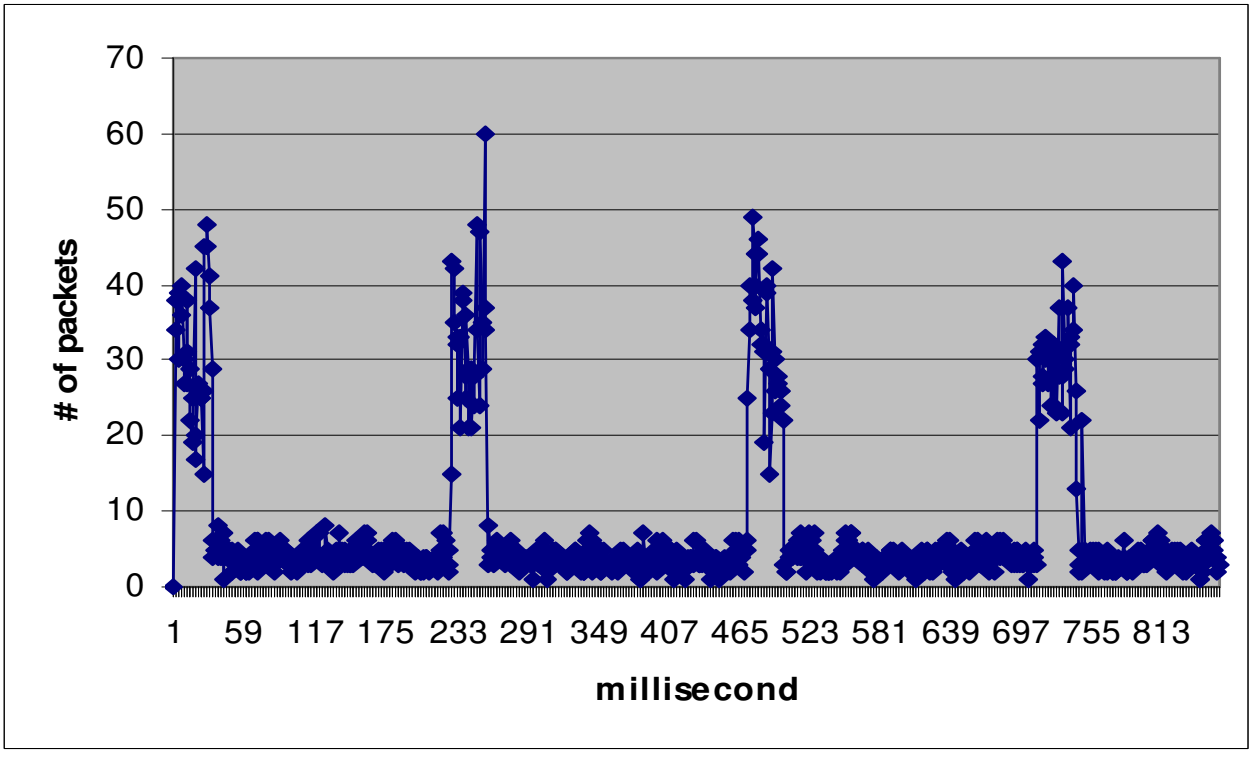

Figure 5. The synthetic network traffic pattern

	Normalized total energy consumption	Total cycles
Fast cores	100.0	3.48×10^8
Slow cores	49.2	6.97×10^8
Switching enabled	59.7	1.25×10^8 (fast cores) (26.9%) 3.39×10^8 (slow cores) (73.1%) 352 switching

Table 5. Power consumption and processing time for the realistic traffic

	Normalized total energy consumption	Total cycles
Fast cores	100.0	3.58×10^8
Slow cores	55.0	8.01×10^8
Switching enabled	59.7	0.79×10^8 (fast cores) (13.8%) 4.96×10^8 (slow cores) (86.2%) 251 switching

Table 6. Power consumption and processing time for the synthetic traffic

by the following performance metrics: queue_utilization (the queue size is 512 in our simulation); Latency (the time between a packet is enqueued and dequeued); Processing cycles (the time to process a packet).

Table 3 and Table 4 show the performance results of the three modes. For both traces, the fast-cores only mode has much lower queue utilization and lower latency and processing time; the slow-cores only mode will result in significant queue overflows for both traces. The switching-enabled mode although has a much higher queue utilization and latency, it still finishes all the packet processing tasks without any packet drops. For the synthetic traffic, it has lower queue utilization than the realistic traffic, due to more bursty behaviors.

5.4 Power consumption

Table 5 and Table 6 show the power consumption and total processing cycles (portions of cycles running on the fast cores and slow cores in the switching-enabled version as well) for the three modes respectively.

We find that we have 40.3% energy savings for the realistic traffic and 43.9% energy savings for the synthetic traffic relative to the fast-cores only approach. The power consumption is very close to the slow-cores only approach (especially for the synthetic traffic) without suffering any packet drops. For the realistic traffic, the system spends about 73% of execution time in the slow array, and 86% for the synthetic traffic. We believe our CMP mode has strong potential to handle the sparse spiky traffic loads efficiently.

We also did a preliminary evaluation to compare our CMP model with the symmetric CMP approach, and we find an 11% power reduction relative to the symmetric CMP approach.

6 Conclusion

In this project, we propose a power aware packet processing architecture with CMP. This CMP architecture with single ISA multi-cores provides an alternative network processor model for power conservation. By simulation, we show for both realistic and synthetic traffic traces, our proposed CMP model can achieve approximately 40architectures without power management. We believe we may have even more potential savings for traffic patterns with more bursty behaviors.

In order to make our approach more energy efficient, we need to add dynamic online prediction of traffic load and thus design more efficient switching policy and precise adaptive queue utilization threshold. Also we may combine the inter-array core switching with intra-array core shutdown for more energy savings.

References

[1] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *International Symposium on Computer Architecture (ISCA)*, June 2000.

[2] D. C. Burger and T. M. Austin. The simplescalar tool set, version 2.0. Technical Report CS-TR-97-1342, University of Wisconsin, Madison, June 1997.

[3] A. Erramilli, O. Narayan, and W. Willinger. Experimental queueing analysis with long-range dependent packet traffic. In *IEEE/ACM Transactions on Networking, 4(2):209-223*, 1996.

[4] K. Flautner, N. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: Simple techniques for reducing leakage power. In *Annual International Symposium on Computer Architecture*, 2002.

[5] M. Franklin and T. Wolf. Power coniderations in network processor design. In *Second Network Processor Workshop in conjunction with Ninth International Symposium on High Performance Computer Architecture (HPCA-9)*, pages 10-22 2003.

[6] Intel IXP2400 Network Processor. *http://www.intel.com/ design/network/products/npfamily/ixp2400.htm*.

[7] S. Kaxiras and G. Keramidas. Ipstash: A power-efficient memory architecture for ip lookup. In *36th International Symposium on Microarchitecture*, Dec. 2003.

[8] R. Kokku, U. Shevade, N. Shah, M. Dahlin, and H. Vin. Energy aware packet processing. In *University of Texas at Austin Technical Report TR04-04*, 2004.

[9] R. Kumar, K. Farkas, N. Jouppi, P. Ranganathan, and D. Tullsen. Single-isa heterogeneous multi-core architectures: The potential for processor power reduction. In *36th International Symposium on Microarchitecture*, Dec. 2003.

[10] M. Nikitovic and M. Brorsson. An adaptive chip-multiprocessor architecture for future mobile terminals. In

	Queue_Util	Max_Latency	Min_Latency	Avg_latency	Max_processing_cyc	Avg_processing_cyc
Fast cores	114	331681	0	5645.54	28190	2902.00
Slow cores	512 (43923 dropped)	3702252	65291	3521120	29118	6958.61
Switching enabled	505	2356878	26818	1672189	28310	4627

Table 3. Realstic network traffic

	Queue_Util	Max_Latency	Min_Latency	Avg_latency	Max_processing_cyc	Avg_processing_cyc
Fast cores	28	94597	0	3821	28190	2910
Slow cores	512 (23302 dropped)	3713762	0	2014250	29118	6996
Switching enabled	428	2419388	0	1409657	28310	4770

Table 4. Synthetic network traffic

the 2002 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES'02), Oct. 2002.

[11] NLANR Network Traffic Packet Header Traces. *http://pma.nlanr.net/Traces/*.

[12] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *the eighteenth ACM Symposium on Operating Systems Principles*, pages 89-102. ACM Press 2001.

[13] Y. Qiao, J. Skicewicz, and P. Dinda. An empirical study of the multiscale predictability of network traffic. In *HPDC'04: 66-76*, 2004.

[14] RIPE Network Coordination Center. *http://www.ripe.net*.

[15] J. Seng, E. Tunea, and D. Tullsen. Reducing power with dynamic critical path information. In *34th International Symposium on Microarchitecture*, 2001.

[16] H. Wang, L. Peh, and S. Malik. Power-driven design of router microarchitectures in on-chip networks. In *36th International Symposium on Microarchitecture*, 2003.

[17] W. Yuan, K. Nahrstedta, S. V. Advea, D. L. Jonesb, and R. H. Kravetsa. Design and evaluation of a cross-layer adaptation framework for mobile multimedia systems. In *SPIE/ACM Multimedia Computing and Networking (MMCN)*, 2003.

[18] B. Zhai, D. Blaauw, D. Sylvester, and K. Flautner. Theoretical and practical limits of dynamic voltage scaling. In *ACM/IEEE Design Automation Conference (DAC)*, June 2004.

[19] Z. Zhang, V. Ribeiro, S. Moon, and C. Diot. Small-time scaling behaviors of internet backbone traffic: An empirical study. In *the IEEE INFOCOM*, 2003.

Appendices

A Data Structures:

```
struct CMP_state {
    enum CMP_status status;
    counter_t timer;
    int transition_timer;
    int processor;
    struct cache_t *cache_dl1[MAX_PROCS];

    // I-cache configuration;
};

typedef struct CMP_Queue {
    char dest[20];
    char stamp[20];
};
```

B The CMP APIs

```
void CMP_queue_read(struct regs_t *regs,
                    mem_access_fn mem_fn,
                    struct mem_t *mem,
                    md_inst_t inst,
                    int traceable );
void CMP_queue_write();
void CMP_queue_init(struct CMP_state *state);
void CMP_queue_finish(md_inst_t inst);
int CMP_queue_fulling();
int CMP_queue_draining();
void CMP_switch_processor(int processor);
void CMP_power_scale(int processor);
```

C The Primitive Implementation

```
void volatile read_primitive(CMP_queue *buffer) {
        int tmp;

        asm volatile("addq $31, %0, $1"::"r"(buffer) );
        asm volatile("bis $31, 1, $31 );
}

void volatile finish_primitive() {
        asm volatile("bis $31, 2, $31 );
}
```