Title

Architectural opportunities for novel dynamic EMI shifting (DEMIS)

ISBN

978-1-4503-4952-9

Authors

Gorman, Daphne I
Guthaus, Matthew R
Renau, Jose

Publication Date

2017-10-14

DOI

10.1145/3123939.3123973

Peer reviewed

Architectural Opportunities for Novel Dynamic EMI Shifting (DEMIS)

Daphne I. Gorman
dgorman@ucsc.edu
Dept. of Computer Engineering
University of California Santa Cruz

Matthew R. Guthaus
mrg@ucsc.edu
Dept. of Computer Engineering
University of California Santa Cruz

Jose Renau
renau@ucsc.edu
Dept. of Computer Engineering
University of California Santa Cruz

ABSTRACT

Processors emit non-trivial amounts of electromagnetic radiation, creating interference in frequency bands used by wireless communication technologies such as cellular, WiFi and Bluetooth. We introduce the problem of in-band radio frequency noise as a form of electromagnetic interference (EMI) to the computer architecture community as a technical challenge to be addressed.

This paper proposes the new idea of Dynamic EMI Shifting (DEMIS) where architectural and/or compiler changes allow the EMI to be shifted at runtime. DEMIS processors dynamically move the interference from bands used during communication to other unused frequencies. Unlike previous works that leverage static techniques, DEMIS dynamically targets specific frequency bands; the type of techniques used here are only possible from an architectural perspective. This paper is also the first to provide insights in the new area of dynamic EMI shifting by evaluating several platforms and showing the EMI is sensitive to many architectural and compilation parameters.

Our evaluation over real systems shows a decrease of in-band EMI ranging from 3 to 15 dB with less than a 10% average performance impact. A 15dB EMI reduction for LTE can represent over 3x bandwidth improvement for EMI bound communication.

CCS CONCEPTS

• **Hardware → Noise reduction**; **Wireless devices**; *Signal integrity and noise analysis*;

ACM Reference format:
Daphne I. Gorman, Matthew R. Guthaus, and Jose Renau. 2017. Architectural Opportunities for Novel Dynamic EMI Shifting (DEMIS). In *Proceedings of MICRO-50, Cambridge, MA, USA, October 14–18, 2017,* 12 pages.
https://doi.org/10.1145/3123939.3123973

1 INTRODUCTION

As mobile devices become more ubiquitous and wireless communication technologies become more diverse, we want our mobile devices to be able to interact with a wide variety of communication technologies. We present to the computer architecture community the problem of in-band electromagnetic interference (EMI) caused by the processor. That is, when a processor is running, it may be emitting Radio Frequency (RF) interference, or EMI, at the same frequencies the device is using for wireless communication, causing interference. Unfortunately, the Shannon-Hartley theorem states that this interference constrains the speed that data can be sent wirelessly.

Computer processors emit non-trivial amounts of electromagnetic radiation; enough that the EMI can be exploited as a security risk [10, 11]. In addition to being a security concern, EMI can also interfere with wireless communication, which is the focus of the work presented in this paper. Even when running just the operating system, the processor will emit radiation at some frequencies. The EMI produced by a processor can create significant desensitization of the antenna, and is a well-known obstacle in the field, which many different publications and patents are dedicated to addressing [5, 15, 24, 27]. Currently, one way to mitigate this problem is to place the computer processor as far away as possible from the antennas [15, 25] to minimize the effect of the processor's EMI. However, modems are placed as close as possible to the antennas in order to mitigate losses before the signals are processed. With the introduction of more integrated chips such as the Snapdragon series, some of which include wireless communication processors, the CPUs are now no longer to be physically isolated this way from the antennas. This problem also tends to get worse with the emergence of smaller, more integrated devices such as wearables.

Although the EMI is deterministic, it is difficult to accurately predict what a processor will emit for a given application. A potential solution is to use Full Wave Simulation software [2, 17, 30] but this is prohibitively slow, in the same manner that requiring full SPICE simulations for a whole chip is not really feasible. Additionally this solution requires a layout, and would not be usable until the chip is ready for tapeout, at which point the designers would need to redesign many completed components. Currently, a feasible solution for finding and addressing EMI produced by a processor does not exist. Designers lack usable models, and these models would be too complicated to actually run. Typically, designers do not realize they have an EMI problem until after a system has been prototyped and evaluated, which puts them in the unfavorable position of having to apply costly and usually ineffective patches, or going back and redesigning the system [15].

This work proposes a novel Dynamic EMI Shifting technique, or DEMIS for short. DEMIS is based on the observation that wireless communication systems use many frequency bands, but not all the bands are used simultaneously and can alleviate the proposed problem by moving RF interference out of the bands being used for communication in a given time.

To illustrate this potential, Figure 1 shows the noise level captured from a spectrum analyzer for a Exynos 5433 processor running SPEC2006 hmmer application. This type of plot has frequency in the x-axis and radiated power in dBm in the y-axis. The higher the radiated power the higher the EMI. The plot shows a Super-WiFi XR7 frequency band. If we focus on this WiFi band, there is a higher EMI. From a communication point of view, we want

MICRO-50, October 14–18, 2017, Cambridge, MA, USA

ACM ISBN 978-1-4503-4952-9/17/10…$15.00
https://doi.org/10.1145/3123939.3123973

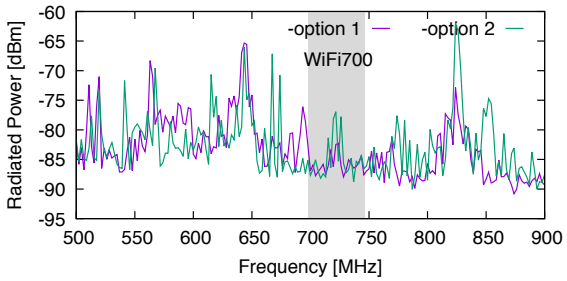

Figure 1: Changing compile options for the SPEC2006 hmmer benchmark on an ODROID-XU4 (Exynos 5422 chip) causes a significant difference in noise in the SuperWiFi XR7 frequencies (grey) with only a small change in performance.

the lowest possible amount of noise. Each line shows a different compilation option for hmmer from the SPEC2006 benchmark suite. The second option has a small 3% performance impact, but it has close to 6dB (or 4× power) reduction of in-band noise.

Since workload, compiler, architecture, and layout are necessary for understanding and addressing this problem, this belongs to the classical architectural domain where impact on execution performance and the interface between compiler/architecture/VLSI layers have different trade-offs. Computer architects are uniquely suited to provide solutions for minimizing in-band EMI from the processor.

In this work, we provide measurements of several processors running different applications to provide insights about EMI. As expected (and shown in previous work [10, 11]), running different processes on the same core will produce different EMI. This work shows and quantifies that small changes in the application may cause significant EMI shifts. We show that running the same application on two different chips causes very different EMI, to the point that an application can have very good noise levels in a core and very bad in another core. Although some cores have the same RTL, if they are manufactured in different fabs, they may not produce similar EMI. Running the same application on different cores may yield vastly different EMI. We particularly show that the interaction between core, process, and application has a deterministic, but very unpredictable result in EMI interference for a given frequency band. We experimentally show that small architectural changes with small performance impact have a big impact on EMI.

Additionally, this work proposes a Dynamic EMI Shifting enabled platform, or DEMIS for short, which leverages the insights about techniques to dynamically shift the interference to out of a band of interest. Notice that DEMIS does not reduce noise in all the bands like static EMI techniques. Instead it reduces the interference in the band currently used for communication. Figure 2 shows our proposal for the DEMIS system, which will monitor its own EMI and adjust its execution accordingly. The layout of this system is based on the Snapdragon 821 layout, which contains the Qualcomm Snapdragon X12 LTE modem and the Qualcomm Kyro CPU on die, but has an additional small DEMIS unit (in grey) that provides directives to the CPU based on the interference.

The main contributions of this paper are:

- Proposing DEMIS, a dynamic methodology based on EMI measurements to reduce in-band EMI during runtime via manipulation of architectural and compiler parameters

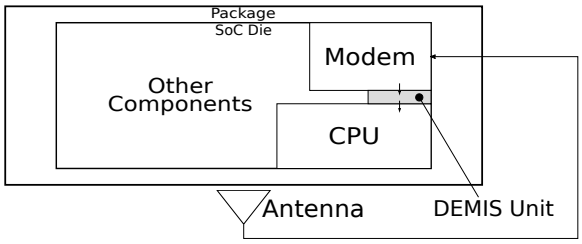

Figure 2: Proposed DEMIS architecture monitors the EMI and provides directives to the processor to change emitted interference.

- Showing that EMI produced by the core is dependent on architectural parameters
- Measuring several real devices to quantify EMI produced
- Presenting the problem of in-band radio frequency noise as a form of EMI to the computer architecture community

After providing some background on RF and EMI in Section 2 and related work in Section 3, we describe our experimental setup in Section 4. Then, we present the techniques and insights we utilized to identify the impact architectural parameters have on EMI in Section 5. We then provide an evaluation of DEMIS, leveraging these insights in Section 6 and show the effects on SPEC2006 benchmarks. Our conclusions and future work are provided in Section 7.

2 EMI BACKGROUND

This paper deals with the electromagnetic radiation from the processor that causes interference for the device's wireless communication. Since EMI may be an unfamiliar topic to parts of this community, this section aims to provide an overview of the main concepts and technologies involved. This is not meant to be a complete introduction to the subject. We start with a brief revision of wireless communication, with a focus on channel capacity and noise, and then explain the most common technologies used in mobile devices that could be affected by electromagnetic radiation from the CPU.

2.1 EMI Overview

Wireless communication is an important feature of most modern computational system, particularly for mobile devices. Wireless communication usually relies on allocating a specific frequency band in which data is transmitted according to a technology-specific protocol. Regardless of which specific protocol is being used, the theoretical amount of data that can be transferred through a channel depends on a certain number of factors, like the bandwidth and the Signal to Noise Ratio (SNR). This relation is governed by the Shannon-Hartley Theorem (Equation 1), which states that the channel capacity (C), or the theoretical upper bound on the net bit rate, is affected by the bandwidth (B) and the SNR ($\frac{S}{N}$). Since SNR is the quotient between noise and signal strength, the lower the noise the higher the bandwidth. Figure 3 shows a visualization of the Shannon-Hartley Theorem for typical values for LTE, with a bandwidth of nine megahertz.

$$C = B * log_2(1 + \frac{S}{N}) \quad (1)$$

Therefore, for a fixed bandwidth and signal strength, increasing the amount of noise will reduce the total channel capacity available, reducing the total amount of data that could be transmitted through

that channel. In this work, we explore the noise emitted by the processor that could interfere with the wireless communication technologies in a device.

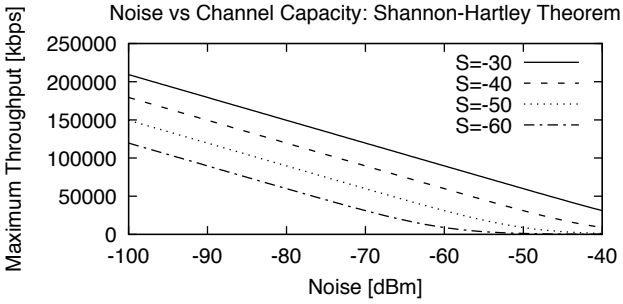

Figure 3: The amount of noise drastically changes the maximum theoretical bandwidth at a particular frequency at different signal strengths (S = signal strength in dBm).

To do so, we assume that the signal strength is out of our control as in practice (as signal strength depends primarily on environmental factors), and thus we focus only on minimizing the noise in order to improve the SNR and thus the bandwidth.

Unfortunately, EMI is produced any time current is present in a system, and thus computer processors constantly emit non-trivial amounts of noise. Fortunately, wireless communication is dependent on specific frequencies, and thus a device only needs certain frequency bands to be less noisy at a given time. Common sources of noise are the clock, its harmonics, memory accesses, and even power converters; these will be discussed later in this paper.

2.2 LTE

LTE is the current standard for cellular communication and was developed with the main objective of improving the communication rates to and from a cell phone. However, with the advent of smartphones, the noise emitted from the phone's CPU can interfere with communication. This section provides a brief introduction to cellular communication technology LTE with a focus on components related to DEMIS. LTE (and wireless communication in general) is commonly divided into frequency bands allocated to different users.

In order to connect to a network tower, a device utilizes two different radio links, one for uplink (where the data goes from the device to the tower) and one for downlink (tower to device) [16]. Most LTE bands are Frequency-Division Duplex (FDD) meaning that the uplink (UL) and downlink (DL) connections operate at different frequency bands between 10 MHz and 400 MHz apart, thus allowing them to happen at the same time [1], meaning communication happens at multiple frequencies simultaneously. However, some LTE bands use Time-Division Duplexing (TDD), which means that UL and DL occur at the same frequency, but at different times. The band selection is determined by many factors such as signal strength and band congestion and is usually out of the control of the user, since it is mostly defined by the tower, but higher-frequency bands are usually preferred since they can send more data.

A noisy band may lead the network to trigger a band switch. The noise may be due to congestion, interference with other networks or simply because the CPU on the cell phone is emitting noise. In this paper we show that architectural parameters can be manipulated to reduce the amount of in-band noise while communication is happening to reduce interference from the core to the antenna.

The implication for DEMIS is that to improve LTE bandwidth, only one or two bands of communication need to be cleared. The band to be cleared is not known a priori, and thus a dynamic band EMI management has high potential benefits.

2.3 Bluetooth

Another technology that is commonly implemented in mobile devices and could suffer from processor EM radiation is Bluetooth. Bluetooth operates at 2.4 GHz, and the entire Bluetooth spectrum spans 83 MHz. Each of the 79 Bluetooth channels has a bandwidth of 1 MHz [7]. The reason for so many channels is that Bluetooth utilizes a Frequency Hopping Spread Spectrum (FHSS) technique, which utilizes each channel in a preset sequence negotiated by both the master and slave when they are first connected. The channel hopping occurs regularly according to that predetermined sequence known to both the transmitter and the receiver.

Bluetooth's high frequency limits the communication distance and also makes it more susceptible to interference. Therefore, it is more crucial to prevent noise from a nearby source, such as that device's own CPU. Like in LTE, DEMIS can further reduce interference focusing on the bands being used at the moment.

2.4 WiFi and WLAN

Finally, another ubiquitous technology that may suffer from EMI from the CPU is WiFi, present in virtually every mobile device on the planet. WiFi operates at both the 2.4GHz and 5GHz (802.11a, n, and ac) frequencies. Older WiFi protocols use frequency hopping (similar to Bluetooth) or spread spectrum transmissions, while newer versions use Orthogonal Frequency Division Multiplexing.

Additionally, the 802.11af standard (also known as White-Fi or Super WiFi) operates in the 54 to 790 MHz range (in bands licensed for TV, VHF, and UHF) and has been in use since 2014. White-Fi uses frequency channels with bandwidths ranging from 6 to 8 MHz, and can use up to four channels at once in one or two contiguous blocks. Although operating at different bands and having different bandwidths, WiFi also divides the spectrum in different bands, dynamically assigned to each device. The communication speed of WiFi and WLAN can also be reduced due to the presence of in-band noise created by the CPU, and thus could also be improved by DEMIS.

3 RELATED WORK

This section provides a description of some of the current findings relevant to electromagnetic emanations on processors.

3.1 On-Chip Interferers

Although this is not a security paper, one of the most prevalent concerns with EM emissions in the computer architecture community is that they can be exploited as a side-channel by hackers. SAVAT [10] determines the impact a single instruction has on the RF signal produced by running a program. The authors were able to distinguish single instructions via studying differences in EM radiation. Thus, showing that radiation patterns of some hardware is dependent on what software is running. We can conclude that manipulating the architectural components of a processor will modify the EM emissions, as the same program will be executed differently.

Table 1: System specifications for each device measured.

Device	Commercial Name	Processor	ARM Core	Operating System	CPU Clock (GHz)	DDR Clock (MHz)
A8_A10	Allwinner A10	Allwinner A10	Cortex A8	Ubuntu	1.2	800
A53_K620	HiKey (LeMaker version)	Kirin 620	Cortex A53	Linaro	1.2	800
A11_PI2	Raspberry Pi 2	Broadcom BCM2835	ARM11	Arch Linux, Raspian (Dual Boot)	0.9	400
A53_C2	ODROID-C2	Amlogic S905	Cortex A53	Arch Linux	1.5	912
A15_XU4	ODROID-XU4	Exynos 5422	Cortex-A15	Ubuntu MATE	2.0	933

FASE [11] is a methodology for finding periodic signals (such as a clock signal) whose amplitude is dependent on processor or memory activity. The authors found that the signals generated by a processor fall into three main categories: strong signals from voltage regulators and power filtering components at the switching frequencies of the regulators, signals from memory-refreshes, and high frequency clock signals and clock harmonics, especially DRAM clocks. They note that all three of these types of signals are affected by what the processor is doing. For example, the signals from the voltage regulators are affected by how much activity is occurring in the processor —the more activity, the higher power consumption, the stronger the signal. The signal caused by memory refreshes are caused by activity by the memory controller, and the EMI from the DRAM clocks are dependant on DRAM activity.

Recently, Sehatbakhsh et al [29] have been using a CPU's electromagnetic radiation to profile code as well. They have shown a correlation between the amount of time a loop takes (T) and a frequency "spike" in the EMI ($f = 1/T$). Some of our findings may be influenced by this phenomenon. However, the measurements presented in this paper show that two processors can exhibit very different EMI, even when the two processors have the same RTL and are running the same benchmark compiled with the same options. This clearly shows that [29] has potential, but it is not obviously applicable, and we consider it future work for potential improvements.

RF ICs are vulnerable to on-chip in-band interferers [4], and may contain circuit-level RF noise couplings that would have a significant impact on system-level performance of wireless communication performance. Unfortunately, finding the cause of interferers is extremely complicated and have so many technical aspects, that achieving a reliable estimation using computer simulations is impossible. Although not strictly EMI, EmerGPU [32] detects and mitigates resonance voltage noise (which causes EM noise) in GPUs. In order to reduce voltage noise, some techniques include reducing the slope of current changes via hardware or software mechanisms.

3.2 Interference from the clock

There has been a lot of work done on minimizing the interference from the clock [19, 20, 22, 23]. As clock speeds tend to be lower than the frequencies used by wireless communications, it is the clock harmonics that have an adverse affect on signal. Therefore, it is common for chips to have modulated clock signals, slightly changing the cycle time of the clock every cycle. By modulating the clock signal, the attenuation for the harmonics is significantly increased, and therefore the harmonics at the communication frequencies are much lower than for an unmodulated clock signal.

3.3 Static Techniques to Reduce EMI

Currently, for LTE, the interference reduction techniques being utilized include only static techniques[1]. These techniques address two separate cases: noise from on-die interferers, and noise from an external source (such as the antenna itself).

The most consistent on-die interferer is the clock. Even low-frequency clocks tend to interfere with communication frequencies, as even the twenty fifth harmonic can create frequency spikes that add non-negligible noise in-band. In fact, some devices fully power down their cores in order to avoid noise during the transmit (TX) phases of communication.

Off-chip interference can be caused by power coupling issues. Even with separate power converters, some parasitics still propagate from core to core. Additionally, during simultaneous TX and receive (RX) actions, noise from the transmit antenna can overpower the received signal, and thus multiple high- and low-pass filters are utilized, and in some cases, the TX signal is fully subtracted out of the received communication. Furthermore, DRAM is often the cause of interference, especially when placed off-chip. However, moving the DRAM on-chip creates less interference.

For Bluetooth, successful operation in the presence of external interferers such as microwave ovens or WiFi networks is to provide some shielding or distance from the cause of the noise [8]. As Bluetooth operates at a relatively high frequency, any in-band interference degrades quickly as distance increases, and even more rapidly through physical objects.

However, if the same device is utilizing both WiFi and Bluetooth, reducing the interference becomes more complex. Currently, collaborative techniques (such as alternating transmissions between Bluetooth and WiFi, or managing packet transmissions based on signal strength) and non-collaborative techniques (such as classifying the Bluetooth channels and altering the channel hopping algorithm to avoid noisy channels) are being investigated [8].

3.4 Dynamic Processing and Execution

As shown throughout this paper, the EMI signature of a process is dependent on several factors and is different when run on processors with the same RTL but manufactured in different fabs. This makes it virtually impossible to know beforehand the EMI signature of a process. Furthermore, a processor will not know which bands will be used for communication during the runtime of that process. Hence the decision of which techniques to use to reduce in-band noise needs to be done at runtime. This section highlights some current techniques that a DEMIS processor could utilize, but evaluation of these techniques is out of the scope of this paper and is reserved for future work.

With DEMIS, we leverage the fact that the EMI signature of a process can be changed by changing architectural parameters such as introducing delays, changing compiler options and so on. This is not possible with regular compilation/execution flows, as those changes need to be made at compile time and cannot be changed during runtime. However, there are tools a DEMIS processor can utilize to change how a process is executed at runtime such as Just In Time compilers (JITs) and interpreted languages. For non-JIT systems, it is possible to switch binaries at runtime, assuming multiple binaries compiled with different compilation options are

[1]Information described in this and the following two paragraphs was acquired from personal contacts in industry.

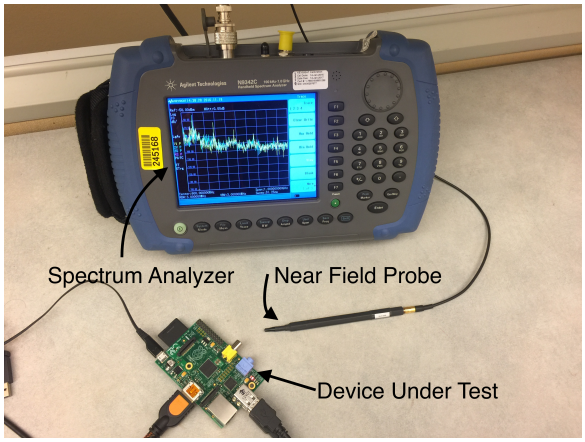

Figure 4: We utilized a spectrum analyzer and near-field probes to accurately measure the EMI from multiple hardwares running a set of benchmarks.

available. Though this is not standard operation, it is not hard to do so form an OS perspective.

Most mobile devices already use JITs, *e.g.*, the discontinued Dalvik [9] and the new JIT introduced in Android 7.0 Nougat [13]. Furthermore, many JITs have been developed for Java [3, 31, 34]. JITs are also widely used during internet browsing on all platforms, with a strong emphasis on JavaScript [12, 14, 18, 26, 28, 33]. Although these tools are almost exclusively geared towards efficiency, they can be repurposed to minimize in-band EMI.

In addition to JITs, there are other tools that dynamically switch how a process is being executed. For example, Dynamo [6] switches between binaries during execution in order to capitalize on runtime optimization opportunities. There are also many other tools that switch binaries at runtime.

4 SETUP

In this section, we describe the experimental setup in order to determine the effect architectural parameters have on EMI, including a description of the hardware used and the benchmarks that were executed.

4.1 Test Equipment

The measurements were taken using a Near-Field Probe set made by Keysight Technologies, which was attached to an N9342C Handheld spectrum analyzer from Agilent Technologies. We show the deployment in Figure 4. Additionally, we fixed the probe onto the device for consistency, as there can be a substantial differences in measured power when measuring different locations.

The specifications of the five devices measured are provided in Table 1. In this paper, we will refer to the devices as in the first column of Table 1.

4.2 Benchmarks

We measured each device while it was idling as well as while it was running a series of benchmarks. We utilized a set of in-house benchmarks, described in Section 5 and SPEC2006 applications, to determine if using architectural techniques would be effective for manipulating EMI in specific bands. From there, we utilized the SPEC2006 [21] benchmarks to determine the effects on larger processes.

Each device ran all the benchmarks from the SPEC2006 benchmark suite[2] natively on each device during measurements with the default settings for each device (without modifying the clock speed, etc.) as a baseline. Then, the benchmarks were run when changing different architectural parameters, including compiler optimizations.

Due to space constraints, the insights section focuses more the mcf, sjeng, and libquantum benchmarks. These benchmarks were chosen because they emphasize specific architectural parameters: mcf is memory intensive, with many RAM accesses; sjeng causes many branch mispredictions and calculations; and libquantum triggers many cache misses and prefetching.

4.3 Bands Analyzed

The noise was measured in the scope of wireless communication technologies and is reported as the difference between emitted power when running the benchmark and idling for five different RF communication frequency bands:

- **LTE 800 Lower** LTE band 18, between 815 and 875 MHz (UL and DL).
- **LTE 700** Split between LTE bands 12, 13, 14, and 17, between 699 and 798 MHz (UL and DL).
- **LTE 450** LTE band 31, between 452.5 and 467.2 MHz (UL and DL).
- **SuperWiFi XR7** Ubiquity XtremeRange7 frequency for WiFi, between 698 and 746 MHz.
- **SuperWiFi 600** SuperWiFi in TV spectrum, 600 and 630 MHz (pending confirmation).

5 INSIGHTS AND MEASUREMENTS

This section uses physical real system measurements to identify the effects of architectural and compiler parameters on RF noise. Through measuring in-house benchmarks on the hardware described in Table 1, we were able to determine that different parameters affect the RF noise differently. This section will describe our findings when measuring the differences in noise when applying different architectural manipulations.

5.1 Compiler Impact on EMI

To evaluate small binary changes impact on EMI, we use different compilation options (different schedulers or optimizations) that should have small performance impact to see the EMI effect. For example, using the O2 or the O3 compilation options, or changing the scheduling using the -mtune option.

Figure 5 shows the noise level and the IPC for the first 100 seconds of execution of mcf when executed on the A11_PI2. The IPC plot shows that the performance does not change when switching from O2 (option 1) to O3 (option 2) for mcf. Nevertheless, we see over 3dB noise reduction in the most noisy band (LTE 700) using O2, while the LTE 450 band has an increase in noise with O3. This means that for the mcf application, if we want to avoid noise in the LTE 450 band we should use O3 whereas O2 should be used if the communication is in the LTE 700 band.

We repeated the same experiment on the A8_A10 platform to understand the impact of hardware changes. The noise levels across

[2]Due to compilation issues, gobmk, omnetpp, and xalancbmk were not included in this experiment.

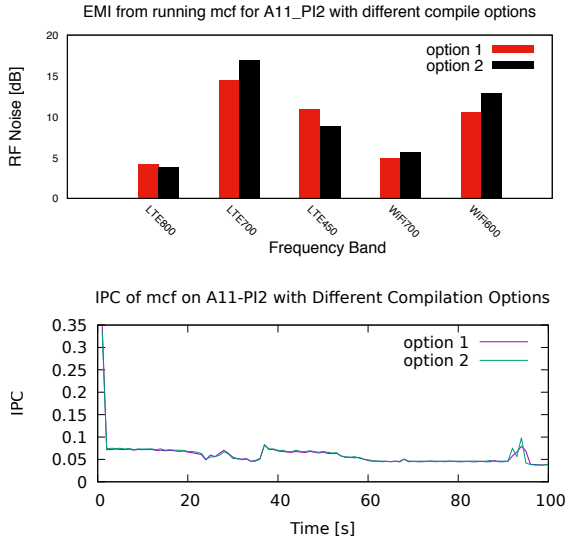

Figure 5: Running mcf with different compile options on the A11_PI2 yielded minimal difference in IPC, while still yielding a 3 dB swing in EMI at certain bands.

the spectrum are shown in Figure 6, which also shows the baseline noise level when the core is powered on, but idle. Unlike in the A11_PI2 case, the O2 improves noise compared to the O3 option for both LTE 700 (by 2dB) and LTE 450 (by 3dB). Again, for mcf the O2 and O3 does not significantly change the performance in this platform.

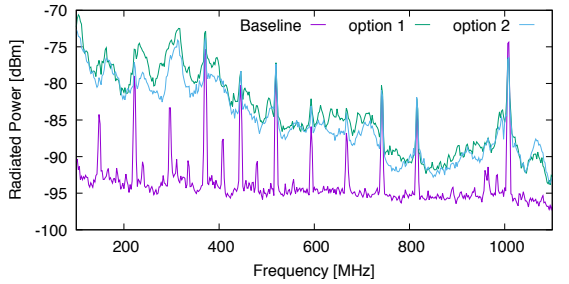

Figure 6: Different compilation options change the EMI in the mcf benchmark on the A8_A10 processor without performance impact.

To analyze its impact on another application, Figure 7 depicts the EMI of the A53_K620 processor running the sjeng benchmark after being compiled O3 and O2. In this case, the O3 compilation option yields a 1% speedup. The noise shifts in frequency slightly, and in most cases the O3 has a higher EMI. For this board mcf showed a similar behavior as sjeng with O2 having less noise.

Figure 8 shows the case that only mtune option is changed for hmmer application. We keep the default -march=native which implies a -mtune=native, and change the -mtune to cortex-a57. In this case, it has no performance difference when executing in the A11_PI2. The plots shows another case of clear shift in the frequency around 600MHz with little performance impact.

The maximum noise reduction for the 5 analyzed bands playing with O2, O3 and mtune options was 8dB in the bzip2 benchmark

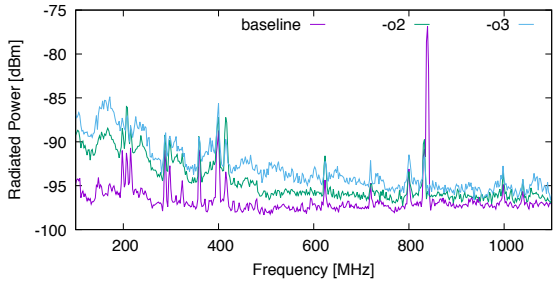

Figure 7: When running sjeng on the A53_K620 processor, the EMI being produced changed based on the optimization the benchmark was compiled with.

(not shown in the plots). Also, as expected, the compilation options can have a big performance impact. For example, in bzip2 the mtune has over 30% improvement when applied in the A15_XU4 board (also not shown in the plots due to length constraints). For the evaluation we do not use the cases with high performance impact.

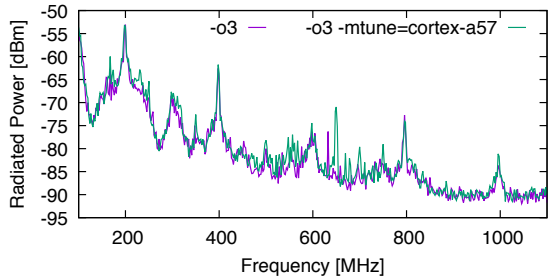

Figure 8: Compiling hmmer with different mtune options has no performance impact but a change in EMI.

In an attempt to further understand the effect of the compiler on the EMI, we performed an exhaustive set of measurements on the A53_K620 board. We measured the EMI across multiple bands, disabling only one minor optimization from the full O3 optimizations at a time. There are 11 gcc flags that differentiate between O2 and O3 with gcc 6.3.1.

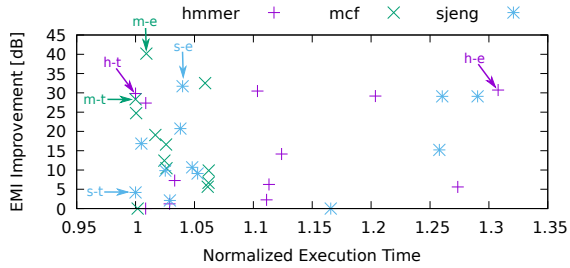

Figure 9: In-band EMI in the LTE 700 band emitted by the A53_K620 board with respect to execution time. We were unable to find a correlation between efficiency and EMI.

Figure 9 shows the normalized execution time versus the normalized EMI for LTE 700 for each of the compilation options. Each dot corresponds to the O3 optimization with one of the 11 flags

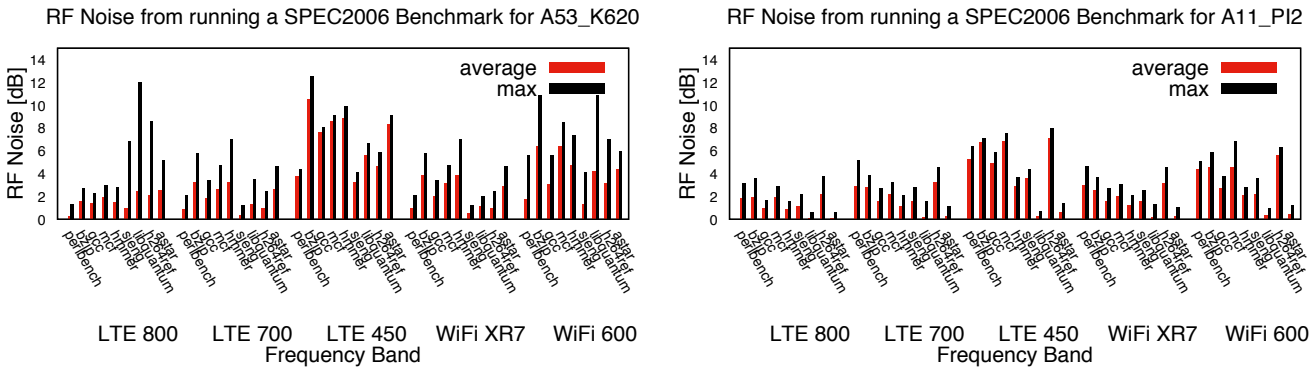

Figure 10: Noise created in each band running SPEC2006 benchmarks on the A53_K620 and the A11_PI2.

disabled. Since we also keep the O3 optimization, there are 12 points per benchmark. The x-axis shows execution time normalized to the fastest execution for that benchmark, and the y-axis shows the EMI improvement with respect to the worst EMI for LTE 700 in dB. Unfortunately, we were unable to discern a consistent correlation between optimization, runtime, and EMI. For example, disabling the fgcse-after-reload optimization has a 40 dB EMI reduction with small slowdown for mcf, but only a 2 dB EMI reduction for hmmer, and 9 dB for sjeng (both also with small slowdowns).

In Figure 9, the points marking the fastest runtime and best EMI improvement are marked for each benchmark. For the hmmer benchmark, the point marked "h-e" for best EMI improvement ran with the fpeel-loops optimization disabled, and had a 1 dB improvement over the fastest run (marked "h-t"), which had the ftree-partial-pre optimization disabled. In the case of this benchmark, there is a 0.3x slowdown between the best EMI and the fastest execution time. The mcf benchmark ran fastest with the ftree-loop-vectorize optimization disabled ("m-t"), but with a 0.009x slowdown, disabling the fcse-after-reload optimization (marked "m-e") offered a 12 dB improvement. Lastly, the sjeng benchmark ran fastest with fpredictive-commoning disabled ("s-t"), but disabling the ftree-slp-vectorize optimization ("s-e") offers over 25 dB improvement with only a 0.03x slowdown.

Clearly, there is no correlation between the optimizations that is standard across benchmarks, even across different bands, there is no discernible relationship. Although potential for future work, we decided not to use multiple binaries, and we restrict ourselves to just O2 vs O3 in the rest of the paper. Keeping many binary files would take up an excessive amount of space and raise tune/selection algorithm issues.

The main conclusion is that by adjusting the compilation options and constraining cases to a small performance impact, we can achieve up to 8dB noise reduction levels with many cases providing 3dB reduction. A source of difficulty managing the system is that the effect is not only compiler dependent but compiler/core/platform dependent. The same compilation options yield opposite results in different processors.

In this paper, when we talk about compiler techniques we mean to change the compiler options (O2/O3/mtune) and select the binary with the lower noise in the band to protect. Although not covered in this work, a JIT based system would be the best platform allowing to dynamically perform small binary changes to mitigate noise while monitoring the noise level impact.

5.2 Benchmark Impact on EMI

Clearly, EMI is different depending on the application [10, 11]. However, EMI also changes on a per-band basis. Figure 10 shows the average and maximum EMI in the five different RF bands on the A53_K620 and the A11_PI2. The noise depicted is the difference between emitted power when running the benchmark at each frequency with respect to an idling processor. However, it is hard to see a relationship between the in-band noise and benchmark that is consistent across processors.

Thus far, all the measured data presented in this paper have been the average of noise per frequency over a sustained amount of time. However, many programs go through different phases during execution. As the different phases tend to run different types of instructions, it stands to reason that the radiation at different frequencies will be different during different phases.

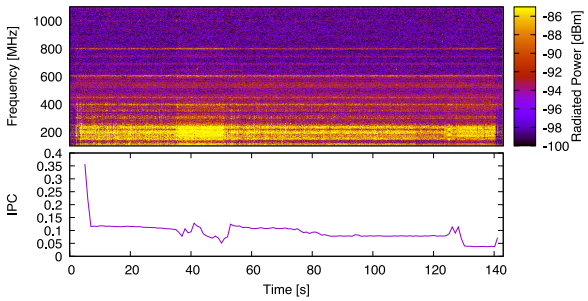

Figure 11: IPC and EMI for mcf on A11_PI2. The processor goes through phases with different EMI patterns. In this case, it seems that when there is lower IPC, there is more EMI.

Figure 11 shows these phases as the mcf benchmark is run on the A11_PI2 processor over time, the noise is shown as a color map per frequency and time. Interestingly, as the IPC decreases, there appears to generally be more RF radiation. From these results it is not clear why this behavior happens, but it could be related to changes in processor activity.

Figure 12 depicts the IPC and phases of the A53_K620 processor running the sjeng benchmark. In contrast to the A11_PI2 running mcf, in many cases the A53_K620 has more noise when the IPC is higher for the sjeng benchmark. Nevertheless, the clear EMI fluctuations are related to IPC changes but it is not a direct function

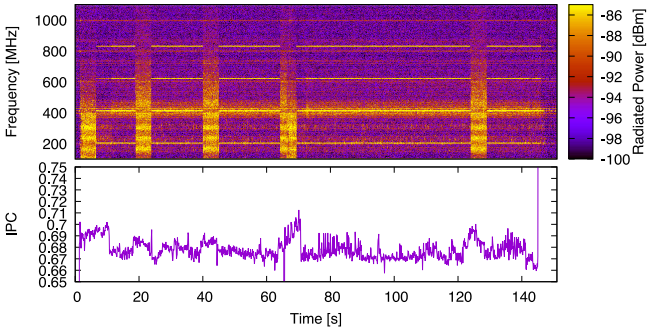

Figure 12: For the sjeng benchmark on the A53_K620, it is more clear that the processor goes through phases with different EMI patterns. When compared with the IPC, there is no clear higher/lower EMI correlation, just change.

of IPC because sometimes higher IPC means lower noise in some bands (which can be seen in in the LTE800 Lower band in Figure 12.

In addition to only observing the phases per benchmark, we also performed some measurements over time with different input sets. Although the different input sets caused each benchmark to spend different amounts of time in each phase, the EMI during each phase remained unchanged for the benchmarks measured in this paper.

The main conclusion is that there are clear program phases in IPC and EMI as the application executes but the IPC phases are not necessarily correlated with EMI phases. Although sometimes an IPC increase results in an EMI increase, in many phases an IPC increase results in an EMI reduction. Adapting through phases has potential benefit and the phases can be detected with IPC phase changes.

5.3 Cache Impact on EMI

Cache accesses consistently affect the amount of RF noise being emitted by each chip. In order to observe how cache accesses affected the RF noise created by each processor, we utilized a synthetic program that emphasizes accessing a large 2D array and injected a delay before the cache misses (Listing 1).

```
1  int main () {
2      int total = 0;
3      for(int i=0;i<8192;i++) {
4          for(int j=0;j<8192;j++) {
5              //asm("nop);
6              total += matrix[j][i];
7          }
8      }
9
10     printf("total=%d\n",total);
11 }
```

Listing 1: Function for testing code with frequent cache accesses. Delays (`nop` calls) were inserted at line 5. The numbers in lines 3 and 4 were manipulated based on the cache sizes for each processor in order to ensure the desired cache misses were occurring (L1 or 2).

The first thing we noticed was that there was a distinct difference in frequency and amplitude between L1 and L2 cache accesses, as shown in Figure 13, which we determined by manipulating lines 3 and 4 in Listing 1. Clearly, L2 cache accesses trigger more noise than L1 cache misses, presumably because an L1 cache access (miss)

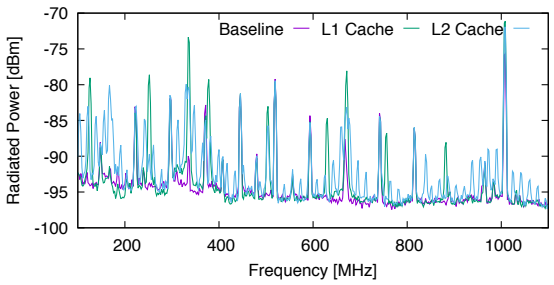

Figure 13: The noise created by accessing the L1 and L2 caches on the A8_A10 processor is significantly different.

is required for an L2 cache access to trigger. However, at some frequencies L1 cache accesses are noisier, which may be attributed to the fact that L2 caches are slower and would therefore interfere with different frequencies. Unfortunately, a processor would take a huge performance hit should it forgo accessing the L1 cache in favor of the L2 cache, so we focused on techniques that would take less critical performance hits, despite the EMI benefits.

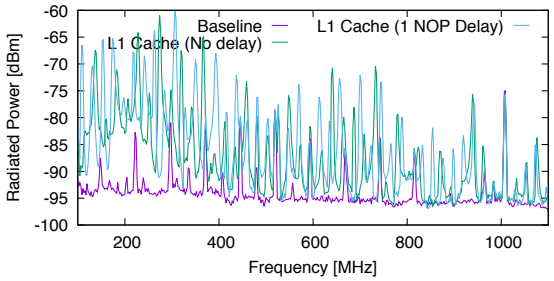

Figure 14: The noise created by accessing the L1 cache with and without delay on the A8_A10 processor is significantly different.

Instead of switching which cache to access, we tried injecting a minimal delay before accessing the L1 cache by uncommenting line 5 in Listing 1. When we delayed the cache access slightly, we were able to reduce the noise, as shown in Figure 14. Clearly, there is a significant affect on the frequency response of the device: consistently a frequency shift of about 15 MHz. Injecting this minimal single `nop` delay was able to trigger a significant response, where the noise spike has moved more than a Bluetooth channel and most LTE channels.

The main conclusion is that caches have a big impact in delay and misses in the L2 tend to have a higher impact at lower frequency bands. Although this has potential to be a good technique, the rest of the paper does not trigger cache delays or misses in the DEMIS evaluation because we are constrained to measurements in real systems, it is not clear how to introduce affect cache behavior at runtime without RTL access.

5.4 Memory Impact on EMI

On the A11_PI2, modifying the DRAM speed was a simple matter of modifying the BIOS parameters. We measured the EMI when the processor was running nothing but the OS. As shown in Figure 15, modifying the `sdram_freq` parameter in the config.txt file that

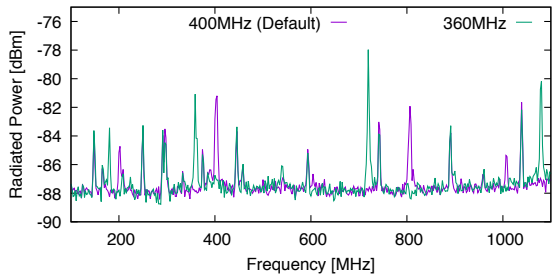

Figure 15: By changing the SDRAM speed for the A11_PI2, we were able to move the noisy peaks in frequency.

serves as boot settings for the A11_PI2 changes the frequency of certain noisy spikes when idling.

These frequency spikes appear to be directly caused by memory accesses because they occur at the DRAM frequency and at its harmonics. When the DRAM frequency was shifted to 360 MHz, we were able to see the spikes shift accordingly: from 400 MHz to 360 MHz, and from 800 MHz (first harmonic) to 720 MHz.

We measured the EMI produced by the A11_PI2 with the default DRAM speed of 400 MHz as well as with a 10% slower speed of 360 MHz. This time, however, we took these measurements while running the mcf and sjeng benchmarks. The results of these measurements are presented in Figure 16. The libquantum benchmark produced negligible EMI on the A11_PI2 processor, so the results are omitted.

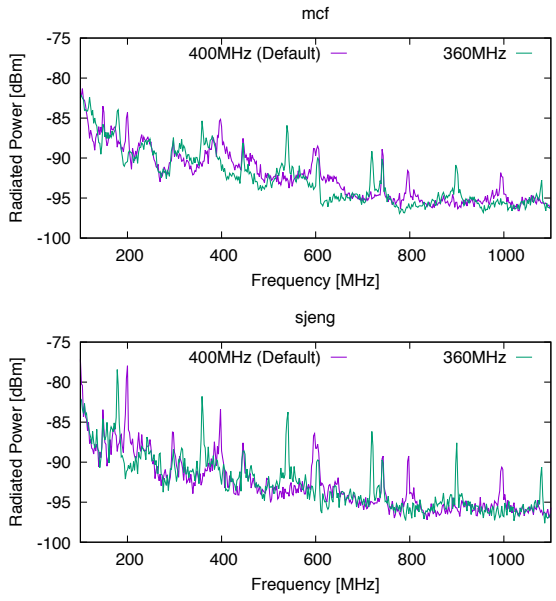

Figure 16: When running the mcf and sjeng benchmarks on the A11_PI2, changing the DRAM speed as little as 10% has a significant impact on EMI.

Reducing the DRAM decreases noise by 3 dB in the SuperWiFi 600 band, but adds a spike in the SuperWiFi XR7 band for the mcf benchmark. However, the default settings have less noise in the SuperWiFi XR7 band and more noise in the SuperWiFi 600 band for the same benchmark, so depending on what type of WiFi is being used, different DRAM speeds are better for this application.

Similar effects can be seen for the sjeng benchmark as well. For example, the default DRAM speed is better for the SuperWiFi XR7 and worse for the LTE 700 frequency bands than the slower DRAM speed.

The main conclusion is that DDR creates big spikes in noise. Even when the processor is idle there are big spikes because the DDR clock is kept running. If the band in use is affected by DDR, the only solution is to shift the DDR operating frequency. For the analyzed cases, small 10% DDR frequency change is enough to move the spikes out of most bands. For many benchmarks like sjeng, this has no performance impact but some benchmarks like mcf are very sensitive resulting in a 10% performance impact.

The DDR clock is one of the strongest EMI spikes, even stronger than the processor clock. The reason is that processor clocks have a more effective modulated clock signal. Future DDR designs may want to consider a better clock modulation requirement.

5.5 Execution Core Impact on EMI

As we saw in Figure 10, even the same benchmark has different EMI when run on different processors. Therefore, we launched a more specific investigation on the impact the execution core itself has on EMI.

```
1  int main () {
2      int total = 0;
3      for(int k=0;k<10000;k++) {
4          for(int i=1;i<10000;i++) {
5              total += k/i;
6              //asm("nop");
7          }
8
9      }
10     printf("total=%d\n",total);
11 }
```

Listing 2: Function for testing computation heavy code. This code was used for testing the FPU by changing all data types from `int` to `float`. Delays were added by uncommenting line 6, and adding as many `nops` as desired.

In order to test the noise generated from using the FPU, we performed multiple divisions, once using the `int` data type and once using the `float` data type. The source code for the benchmark used is in Listing 2 in the A8_A10 hardware.

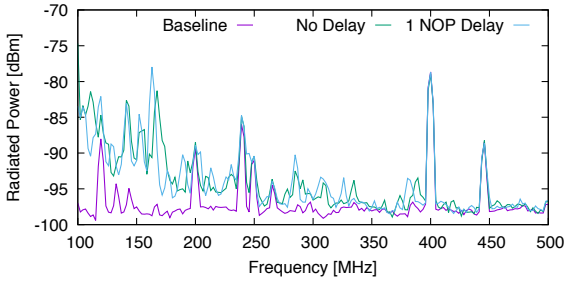

Figure 17: When utilizing the FPU as opposed to doing simply integer calculations on the A8_A10 processor, there was much less RF interference. Furthermore, different frequencies were affected differently.

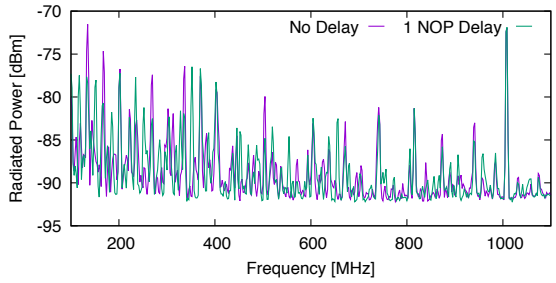

Figure 18: RF noise created when performing integer calculations on the A8_A10 processor. The frequency response differs with the addition of a single nop in the loop.

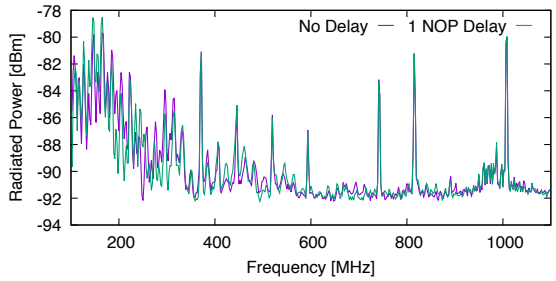

Figure 19: RF noise created when performing floating-point operations on the A8_A10 processor. The frequency response differs with the addition of a single nop in the loop.

We noticed a substantial difference in EM radiation between the two test cases on the A8_A10 as in Figure 17. Not only did the noise decrease, but we also observed that the lower frequencies tended to exhibit a more consistent noise during floating-point calculations, whereas higher frequencies observed consistently more noise from integer-only calculations.

By injecting even a single nop into the calculations (uncommenting line 6 in Listing 2), we were able to observe a distinct difference between frequency responses for both integer calculations and floating point calculations as seen in Figure 18.

As the noise generated from the integer calculations exhibited frequent changes in power in each frequency, it is easy to determine that the power spikes undergo a frequency shift when one nop is injected into the code. Furthermore, the frequency spikes tend to have less power in the benchmark with a delay by almost 7 dB in the lower bands and 5 dB at higher frequencies. The measured power is presented in Figure 18.

Figure 19 shows the difference in frequency responses when running the floating point benchmark with and without a nop delay. As opposed to the integer calculations, the floating point calculations have wide-band low-frequency noise. However, by injecting a nop into the inner `for` loop, we were able to move the noise to different frequencies.

Figure 20 shows the radiated power from the A53_C2 and A53_K620 processors when running the three benchmarks without changing any of the default settings on the boards. The benchmarks were compiled without any optimizations. It is clear in these two figures that each benchmark yields a distinct radiation spectrum on each processor. Also, the benchmarks tend to behave similarly across processors. As the processors all run at different frequencies, we expect

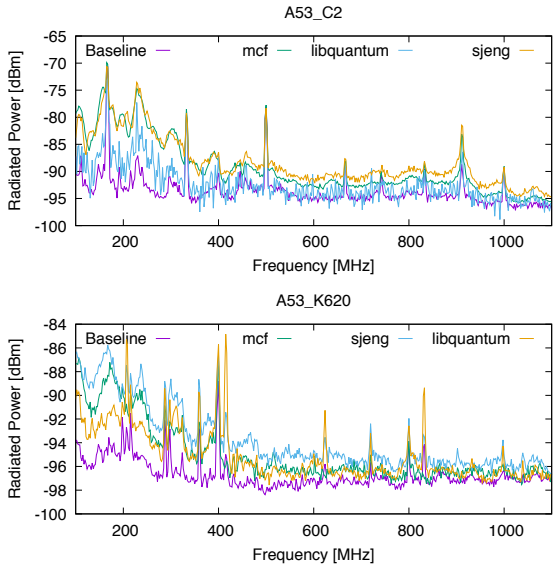

Figure 20: When running the different benchmarks on the A53_C2, and A53_K620 processors, the EMI being produced varied noticeably. A53_C2 and A53_K620 are fabricated from the same RTL.

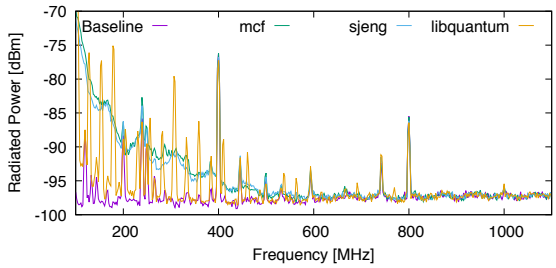

Figure 21: When using the A15_XU4, the EMI varied significantly across benchmarks. However, no significant EMI was produced above 600 MHz, and therefore is irrelevant to most wireless technology frequencies.

some variations, but in general it is clear that specific architectural components produce distinct noise patterns.

It is important to note that the A53_C2 and the A53_K620 processors are fabricated from the same RTL in the same fab and process node. Therefore, it is surprising that the EMI is so different between the two processors.

Additionally, on the A53_C2 processor, the libquantum benchmark produces less noise than sjeng, but on they are switched on the A53_K620 processor. We found this to be quite common throughout our investigation.

We were also able to measure these benchmarks on an out of order (OoO) processor, the A15_XU4 which features four OoO cores and four in-order cores. The results are provided in Figure 21. Interestingly enough, the OoO core appears to only emit noticeable radiation in frequencies less than 600 MHz, which is below most wireless communication bands. However, running the libquantum benchmark introduces multiple 10 dB spikes into the LTE 450 band, which could cause significant connectivity degradation.

The main conclusion is cores have a high impact in processor noise, with some cores like the A15_XU4 having less noise at higher frequencies. The high power blocks like the FPU can produce interference but adding delays which sort of behaves like a frequency modulation of the FPU changes the noise level. Even the same RTL (A53) has a very different behavior in absolute and relative terms between applications. This means that other factors besides architecture can have a big impact, but as show small core fluctuations can compensate for the noise.

5.6 Further Insights

In addition to the hardware described in Table 1, some experiments were also performed on a MYiR z-turn Board, running Ubuntu on a Xilinx ZYNQ chip that includes an FPGA on chip. Unfortunately, the measurements were all too noisy to provide usable results, as the processing noise was overpowered by noise that we believe was generated by the FPGA itself. However, we were able to note that when starting and ending a process, all frequencies measured experienced a significant increase in noise (more than 30 dB). We hypothesize that this noise may have been caused by components being powered on or off during that time.

6 DEMIS EVALUATION

While previous sections have provided insights, this section evaluates a DEMIS approach. In this section, we evaluate the efficacy of the techniques proposed in Section 5 using the same hardware running SPEC2006 benchmarks. Once again, we utilized the hardware described in Table 1 and performed measurements using the same N9342C Handheld Spectrum Analyzer from Agilent Technologies and Keysight Technologies' Near Field Probe Set.

We propose utilizing the techniques in Section 3.4 for a DEMIS-enabled core to mitigate the EMI from the processor during execution. As shown in Figure 2, a DEMIS processor will be able to monitor its own EMI, and when the in-band interference exceeds a certain threshold, will switch to another processing configuration. Because DEMIS is only useful for devices with wireless communication, we suggest using the existing antenna on the device to monitor the EMI, leveraging the data from the modem to determine when the processor is generating significant interference. However, the purpose of this paper is to show that using computer architectural techniques is effective in mitigating in-band EMI, and therefore designing and implementing this fully DEMIS-enabled core is beyond the scope of this work.

To evaluate DEMIS, we run all the different compile options, core, and DDR frequencies. We evaluate the effectiveness for each of the 5 analyzed bands (LTE 800, LTE 700, LTE 450, WiFi XR7 and WiFi 600). For the baseline, we pick the compile option and frequencies with the highest performance. For the DEMIS solution, we pick the configuration with the highest noise reduction as long as it that does not have more than 10% slowdown. To illustrate the selection process, Figure 22 shows all the options for just the LTE 800 band in an Exynos A15 core. Each bar shows the noise level. For the baseline and hmmer run, the fastest (speed not shown on the plot) is the configuration with default CPU speeds and using the O2 compilation option with the default scheduling algorithm. The lowest EMI with less than 10% slowdown is the configuration with the CPUs running with a 5% slowdown and using the O3 optimization. While the first configuration is selected for the baseline hmmer point, the second configuration is selected for the DEMIS hmmer point.

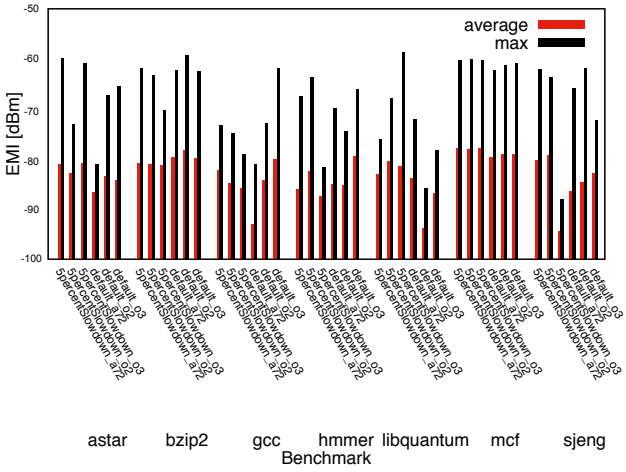

Figure 22: Noise for all the configurations in the LTE 800 band with the A15_XU4 core.

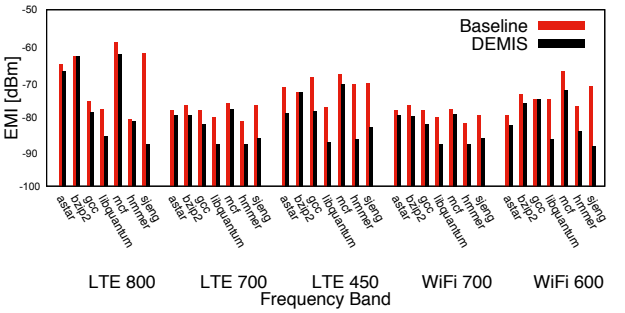

Figure 23: DEMIS has a significant EMI reduction across all the bands on the A15_XU4.

We apply this process over all the bands. The result is shown in Figure 23. This plot summarizes the main results with a 6% average EMI reduction, with an average of only 5.5% performance reduction

While the main results only show data for the A15 core, we performed the same process on all the platforms. Due to submission space constraints of 11 pages we can not add the other cores, but if another page can be added we will add these plots. Different cores have different benefits, the A11_PI2 has an average noise reduction of 6.5% across the five bands analyzed with the highest reduction being in the SuperWiFi XR7 band with a 8dB reduction. This is done with less than 10% slowdown.

The main conclusion is that DEMIS provides opportunities to reduce noise across the different frequency bands. Doing architectural and frequency changes we have been able to measure EMI reductions from 3 to over 8dB across multiple platforms. We do so capping the maximum slowdown under 10% with a modest average of 6% across SPEC2006 applications. We think that these results open a new opportunity to dynamically manage EMI.

7 CONCLUSIONS

In this paper, we introduce to the computer architecture community the wireless communications challenge of EMI, and the opportunities for applying architectural techniques to address this challenge

in a novel, dynamic solution. We propose DEMIS, utilizing architectural and/or compilation changes in order to dynamically reduce or shift in-band EMI.

We offer insights into multiple different architectural parameters, and investigate their effect on EMI. Of the multiple parameters we studied, cache access schemes, clock speed, and small process delays had little impact on processor efficiency. However, small dynamic changes of these produced a significant reduction of in-band interference. These observations were expanded to include more system-level techniques for reducing in-band EMI such as changing compilation options.

Section 5 shows techniques that can be applied in current systems like compiler options or frequency parameters, but it also points to opportunities with the creating of synthetic benchmarks pointing to further opportunities.

DEMIS is our architectural solution, described and evaluated in Section 6, that integrates the dynamic manipulation of these parameters to reduce the interference in the frequency bands used by cellular, WiFi and Bluetooth communication technologies. Our results show that our 15 dB EMI reduction for LTE can represent over 3x bandwidth improvement for EMI bound communication.

This work shows that more research on RF interference emitted by processors with on-chip FPGAs and out of order processors should be conducted. Additionally, current wireless standards take into account a small amount of noise. Therefore, improving the SNR as we do using DEMIS would open up opportunities for improved bandwidth, as these standards would be able to take into account higher maximum throughput as the EMI decreases. Furthermore, we would like to look into dynamically switching between binaries of the same code (with different compilation arguments) during runtime, and if it would be a feasible solution for mitigating in-band EMI as a program enters noisy phases. Another option would be to investigate switching from one core to another as a program executes and see the effects on the EMI. Finally, DRAM clock modulation and introducing a time delay between two cores are interesting prospects that we were unable to fully test at this time due to lack of resources. Despite these constraints, the findings we were able to make based on purely architectural techniques show significant promise in improving wireless communications.

8 ACKNOWLEDGMENTS

We like to thank the reviewers for their feedback on the paper. This work was supported in part by the National Science Foundation under grants CNS-1059442-003, CNS-1318943-001, CCF-1337278, and CCF-1514284. Any opinions, findings, and conclusions or recommendations expressed herein are those of the authors and do not necessarily reflect the views of the NSF.

REFERENCES

[1] 3GPP. [n. d.]. 3GPP TS 36.211: E-UTRA:Physical Channels and Modulation. ([n. d.]). www.3gpp.org/dynareport/36211.htm.
[2] HFSS Ansoft. 2007. ver. 11. *Ansoft Corporation, Pittsburgh, PA* (2007).
[3] John Aycock. 2003. A brief history of just-in-time. *ACM Computing Surveys (CSUR)* 35, 2 (2003), 97–113.
[4] Naoya Azuma, Tetsuya Makita, Satoshi Ueyama, Makoto Nagata, Satoshi Takahashi, Motoki Murakami, Kenji Hori, Shoji Tanaka, and Masaki Yamaguchi. 2013. In-system diagnosis of RF ICs for tolerance against on-chip in-band interferers. In *Test Conference (ITC), 2013 IEEE International*. IEEE, 1–9.
[5] Bonnie Baker. 2012. EMI Problems? Part two: Where does EMI come from? (2012). http://www.edn.com/electronics-blogs/bakers-best/4369076/EMI-problems-Part-two-Where-does-EMI-come-from-.
[6] V. Bala, E. Duesterwald, and S. Banerjia. 2000. Dynamo: A Transparent Dynamic Optimization System. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*. Vancouver, Canada, 1–12.
[7] Bluetooth. [n. d.]. What is Bluetooth Technology? https://www.bluetooth.com/what-is-bluetooth-technology. ([n. d.]). Accessed: 2016-04-04.
[8] Bluetooth. 2002. *Wi-Fi and Bluetooth - Interference Issues.* Technical Report. HP. Accessed: 2016-04-04.
[9] Dan Bornstein. 2008. Dalvik vm internals. In *Google I/O developer conference*, Vol. 23. 17–30.
[10] Robert Callan, Alenka Zajić, and Milos Prvulovic. 2014. A practical methodology for measuring the side-channel signal available to the attacker for instruction-level events. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 242–254.
[11] Robert Callan, Alenka Zajic, and Milos Prvulovic. 2015. FASE: finding amplitude-modulated side-channel emanations. In *Computer Architecture (ISCA), 2015 ACM/IEEE 42nd Annual International Symposium on*. IEEE, 592–603.
[12] Jay Conrod. 2013. A tour of V8: full compiler. (Dec. 2013). http://jayconrod.com/posts/51/a-tour-of-v8-full-compiler
[13] Paul Dean. 2016. Here's everything new in Android Nougat 7.1. (2016).
[14] Xue Fuqiao, Florian Scholz, Karen Scarfone, Berker Peksag, Till Schneidereit, Eric Shepherd, and Chris Leary. 2014. Tracing JIT. (May. 2014). https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey/Internals/Tracing_JIT
[15] Robin Getz and Bob Moeckel. 1996. Understanding and Eliminating EMI in Microcontroller Applications. *National Semiconductor* (1996).
[16] Neal Gompa. 2015. Deep dive: What is LTE? http://www.extremetech.com/mobile/110711-what-is-lte. (April 2015). Accessed: 2016-04-04.
[17] Mentor Graphics. 2004. Hyperlynx Signal Integrity Simulation software. (2004).
[18] Brian Hackett and Shu-yu Guo. 2012. Fast and Precise Hybrid Type Inference for JavaScript. *SIGPLAN Not.* 47, 6 (Jun. 2012), 239–250. https://doi.org/10.1145/2345156.2254094
[19] Keith B Hardin, John T Fessler, and Donald R Bush. 1994. Spread spectrum clock generation for the reduction of radiated emissions. In *Electromagnetic Compatibility, 1994. Symposium Record. Compatibility in the Loop., IEEE International Symposium on*. IEEE, 227–231.
[20] Keith B Hardin, John T Fessler, and Donald R Bush. 1995. A study of the interference potential of spread spectrum clock generation techniques. In *Electromagnetic Compatibility, 1995. Symposium Record., 1995 IEEE International Symposium on*. IEEE, 624–629.
[21] J. L. Henning. 2006. SPEC CPU2006 benchmark descriptions. *SIGARCH Comput. Archit. News* 34, 4 (Sept. 2006), 1–17.
[22] Xuchu Hu and Matthew R Guthaus. 2011. Clock tree optimization for electromagnetic compatibility (EMC). In *Proceedings of the 16th Asia and South Pacific Design Automation Conference*. IEEE Press, 184–189.
[23] S I-The, A Chen, and J Keip. 2000. Spread spectrum and PLL technology combine to reduce EMI. *RF DESIGN* 23, 4 (2000), 20–25.
[24] Mark A McHenry, Dennis Roberson, and Robert J Matheson. 2015. Electronic Noise Is Drowning Out the Internet of Things. *IEEE Spectrum: Technology, Engineering, and Science News, Available at: http://spectrum.ieee. org/telecom/wireless/electronic-noise-is-drowning-out-the-internet-of-things.[Accessed: 3 Jun. 2016]* (2015).
[25] Leo Janghwan Oh. 2017. RF Desense story 2. (2017). https://www.linkedin.com/pulse/rf-desense-story-2-leo-janghwan-oh.
[26] Kailas Patil. 2011. JaegerMonkey Architecture. (Aug. 2011). http://kailaspatil.blogspot.com/2011/08/jaegermonkey-architecture.html
[27] Erik Rolf, Anders Petersson, Ola Samuelsson, and Pontus Nelderup. 2009. Desense with adaptive control. (Aug. 6 2009). http://www.google.ch/patents/US20090197591 US Patent App. 12/025,254.
[28] Florian Scholz. 2014. SpiderMonkey Internals. (May. 2014). https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey/Internals
[29] Nader Sehatbakhsh, Alireza Nazari, Alenka Zajic, and Milos Prvulovic. 2016. Spectral profiling: Observer-effect-free profiling by monitoring EM emanations. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*. IEEE, 1–11.
[30] Microwave Studio. 2008. CST-Computer Simulation Technology. *Bad Nuheimer Str* 19 (2008), 64289.
[31] Toshio Suganuma, Takeshi Ogasawara, Mikio Takeuchi, Toshiaki Yasue, Motohiro Kawahito, Kazuaki Ishizaki, Hideaki Komatsu, and Toshio Nakatani. 2000. Overview of the IBM Java just-in-time compiler. *IBM systems Journal* 39, 1 (2000), 175–193.
[32] R. Thomas, N. Sedaghati, and R. Teodorescu. 2016. EmerGPU: Understanding and mitigating resonance-induced voltage noise in GPU architectures. In *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 79–89. https://doi.org/10.1109/ISPASS.2016.7482076
[33] Andy Wingo. 2011. v8: a tale of two compilers. (Jul. 2011). http://wingolog.org/archives/2011/07/05/v8-a-tale-of-two-compilers
[34] Byung-Sun Yang, Soo-Mook Moon, Seongbae Park, Junpyo Lee, SeungIl Lee, Jinpyo Park, Yoo C Chung, Suhyun Kim, Kemal Ebcioglu, and Erik Altman. 1999. LaTTe: A Java VM just-in-time compiler with fast and efficient register allocation. In *Parallel Architectures and Compilation Techniques, 1999. Proceedings. 1999 International Conference on*. IEEE, 128–138.