AN ABSTRACT OF THE THESIS OF

Ryan K. Albright for the degree of Master of Science in
Electrical and Computer Engineering presented on April 20, 2012.

Title: Optimizing Performance/Watt of Embedded SIMD Multiprocessors through a priori Application Guided Power Scheduling

Abstract approved: _____

Patrick Y. Chiang

A method for improving performance/watt of an embedded single-instruction multiple-data (SIMD) architecture using application-guided a priori scheduling of hardware resources is presented. A multi-core architectural simulator is adopted that accurately estimates power, performance, and utilization of various processor components (logic, interconnect and memory). A greedy search is then performed on each algorithm block of a signal processing chain in order to schedule each component's throughput and power. The proposed software-directed hardware rebalancing, applied to a typical electroencephalography (EEG) filtering chain, is analyzed for two different SIMD architectures. The first, representing a super V_{th} processor demonstrates a 51%-86% improvement in performance/watt at 1%-10% throughput reduction using block level or algorithm level a priori scheduling. The second architecture used is Synctium, a near V_{th} processor which demonstrates

50%-99% performance/watt improvement across the same throughput reduction range and optimization techniques.

Optimizing Performance/Watt of Embedded SIMD Multiprocessors through a priori Application Guided Power Scheduling

by

Ryan K. Albright

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented April 20, 2012
Commencement June 2012

Master of Science thesis of Ryan K. Albright presented on April 20, 2012.

APPROVED:

Major Professor, representing Electrical and Computer Engineering

---

Director of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Ryan K. Albright, Author

ACKNOWLEDGEMENTS

I would like to thank everyone who has helped me on my project, specifically my advisor Dr. Patrick Chiang. I would also like to thank my committee: Dr. Terri Fiez, Roger Traylor and Dr. Joe Zaworski. I would also like to thank my research group: Jacob Postman, Robert Pawlowski, Joe Crop, Ben Goska, and Thomas Ruggeri whose countless advice has made this work possible.

CONTRIBUTION OF AUTHORS

Optimization could not have happened without the simulator work and power model exploration done by Benjamin Goska and Thomas Ruggeri as well as the work on Synctium done by Robert Pawlowski.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF TABLES

DEDICATION

For my parents, Mike and Becky Albright, and my wife Heather, whose countless guidance and support has shaped me into who I am today.

Chapter 1 – Introduction

Power consumption has become one of the most critical design decisions for next-generation embedded computing platforms, placing tight constraints on battery size, cost and lifetime. Power consumption in wireless embedded applications is often greatly dependent on the choice of microprocessor. The designer will look and analyze many processors for ease of programability, hardware resources, and power consumption. The problem with this is that the designer also often has a timeframe in which this system must be complete, so even with the vast amount of choices for microprocessors in today's market, they will often choose one that is almost certainly capable of the computation they are performing at the risk of under-utilizing the processor.

Most advanced applications are very expensive in terms of energy when executing on a typical embedded microprocessors and are often either logged to non-volatile memory or transmitted wirelessly off the system for post processing. Both of these options are power hungry and shift where energy is used rather than solving the problem. The real issue is that it typically costs too much energy to perform the advanced processing on chip, and in order to limit the power profile of the system a designer will choose to use a secondary system to analyze and manipulate data later.

Total system power has been previously shown to improve dramatically when sensor signal processing is performed locally as opposed to transmitting the raw input data through the power expensive radio [44]. Therefore, future wireless system-on-chips

(SoCs) will benefit from significant improvements in parallel processing that is both low power and high performance in order to minimize radio utilization. This local parallel computing unit within the SoC will need to perform all of the necessary tasks required for local signal processing, such as filtering, compressive sensing [18], multichannel decomposition, anomaly detection, compression, encryption, error correction and network radio processing.

This presents a key issue in that most designs are approached by only limiting the designer to a power envelope, without an efficiency requirement. For instance, a designer is tasked with building a device that can last X hours, and already knows the power profile of everything but the processor. They now will often choose a processor that will fill this unused power to avoid the risk of the processor not being able to perform the computation in time. At first glance there is nothing wrong with this, as the designer will probably meet the specification requirements but could most likely have achieved similar performance while using less energy. How can we accomplish this?

Typically, embedded processing applications execute on a fixed set of algorithms across a well defined signal processing chain repeatedly, as opposed to the uncertain program execution performed on a general purpose microprocessor. Due to our prior knowledge about the algorithm that will be executed, the amount of time we are allocated to perform computation, and the necessary power profile, a priori characterization of the hardware utilization patterns for the embedded application can be performed. Further, it is possible for these utilization patterns to be used in order to balance hardware resources.

For data-parallel applications such as signal processing, wide parallel architectures

can significantly improve energy-efficiency when compared with conventional scalar pipeline architectures [25]. These efficient parallel architectures can achieve improved performance/watt through system-level power management, based on application-aware hardware utilization characterization and power scheduling.

Due to the afore mentioned issues, I present an efficient greedy search optimization technique for maximizing performance/watt in SIMD embedded architectures through a priori scheduling of hardware resources based on algorithm and utilization patterns on both a super V_{th} and near V_{th} processor.

Chapter 2 – Motivation

The primary motivation for this research was due to our design of biomedical devices which typically must be:

- Small & Lightweight
- Reliable
- Have long battery life

This presents a host of problems for a designer however. One example of such as device is the Oregon State Univerisy Life & Activity Monitor [2].

2.1 OLAM: Oregon State University Life & Activity Monitor

OLAM was designed for the Linus Pauling Institute at Oregon State University for an IRB approved clinical trial on the effects of lipoic acid on the circadian rhythm of patients. This required the ability to monitor heartrate and general activity over a course of many weeks. The required specifications for OLAM were as follows.

Design contraints:

- Ability to collect data for two weeks on a single charge
- Able to detect heartbeats using a non-contact sensor

- Able to detect motion and activity levels
- Data retrieval must be as quick as possible
- Device must be less than 2”L x 2”W x 0.5”H

2.2 OLAM: Design Decisions

What we came up with was to use an accelerometer and gyro for motion tracking and an analog front end ADC for measuring heartbeats. These can be seen in Figure 2.1. As for the heartbeat sensor, we developed our own non-contact sensor as seen in Figure **??**. You may also notice that the battery was a large portion of the total package. In order to meet the battery life contraints, we chose to fill the available volume that we were allocated with battery as seen in Figure 2.3.

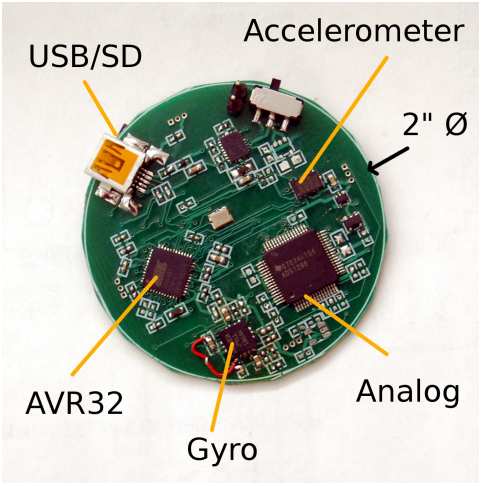


Figure 2.1: OLAM: PCB

Figure 2.2: OLAM: Bottom View

Figure 2.3: OLAM: Top View

The necessity to increase battery size should be remedied with better energy efficiency. While OLAM was designed using as low energy components that could be found, the processor was still underutilized and therefore should have the ability to scale resources to increase utilization and lower energy.

Chapter 3 – Literature Review

The following is a comprehensive review of previous work as it relates to my research. Firstly I cover various parallel processing techniques. Next I review power scaling and optimization techniques already used. Then I cover some parallel processor simulation environments. Lastly, I review the costs and benefits of near threshold operation of digital circuits.

3.1 Parallel Processing

Parallel processing has become increasingly important as the need for higher throughput processors exceeds our ability to execute instructions in serial. To achieve a performance speedup (the ratio between running time of a program on a single core system versus that of a multi-core system), designs are becoming more parallel. This speedup can only be realized however, if algorithms tend towards parallization as well. According to Amdahl's law, speedup follows:

$$S(N,P) = \frac{1}{(1-P) + \frac{P}{N}}$$

Where S is the speedup, N is the number of cores and P is the percent of the algorithm that is parallelizable. Figure 3.1 shows this relationship. In [41] it is shown that a two core system can perform almost as well as an infinite core system for entirely

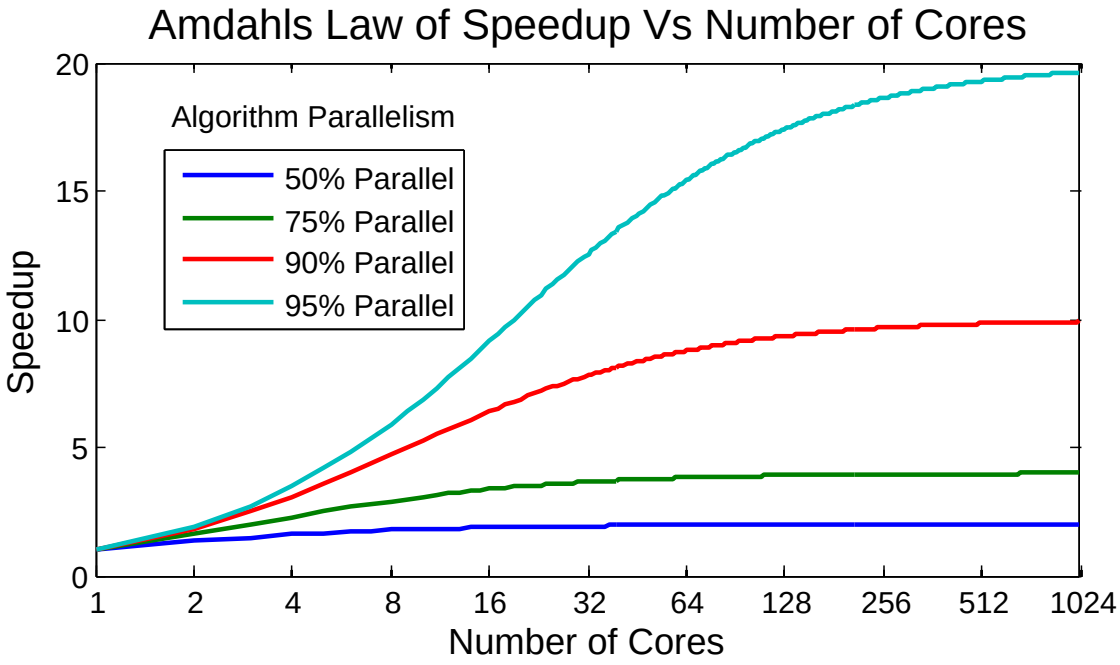


Figure 3.1: Amdahl's Law: Speedup vs Number of Cores

serialized code which shows that the benefits of parallelization of architectures can only be realized if algorithms also become wide parallel.

There are several different forms of parallel processing: bit-level, instruction level, data, and task level parallelism.

3.1.1 Bit Level Parallelism

Bit level parallelism is typically implemented by increasing a processors word size [33] and is used when increasing word size is more efficient than many smaller instructions. By increasing word size, it is possible to reduce the total number of instructions when dealing with very large variables. For example, consider the case where an 8 bit proces-

sor must perform 32 bit addition. This would require 4 addition instructions with 4 bit carries whereas a 32 bit processor would be able to accomplish the same goal in a single instruction. In previous processor generations, 4-bit processors were replaced by 8-bit, then 16-bit and 32-bit and has recently settled at a 64-bit typical word size for general computing. System level trade-offs of area, performance, and power are evaluated for very large instruction word (VLIW) architectures in [4].

3.1.2 Instruction Level Parallelism

Instruction Level Parallelism (ILP) is a processor and compiler design technique wherein execution speedup can be obtained through the ability to run multiple instructions at once [45]. This technique is most commonly known for its use in superscalar architectures [24] wherein a multi-stage pipeline allows for multiple instructions to be executed at different stages in the pipeline at once. This allows for faster throughput at a given clock rate. The superscalar technique can be identified by the following characteristics:

- Sequential Instruction Stream
- CPU dynamically checks data dependencies between instructions at run time
- CPU can accept multiple instructions per clock cycle

This type of parallelism is limited mostly by the depth of the pipeline. For large pipelines such as the Intel Pentium series [46] this technique worked very well, and achieved some level of speedup for most general computing applications.

Instruction	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	...
4				IF	ID	EX	...
5					IF	ID	...
Cycle	1	2	3	4	5	6	7

Stage Definitions	
IF	= Instuction Fetch
ID	= Instruction Decode
EX	= Execute
MEM	= Memory R/W
WB	= Register Writeback

Figure 3.2: Basic Five Stage Pipeline

ILP can also be seen in smaller configurations such as the 5 stage pipeline adopted by many architectures such as RISC [26]. The principle can be seen in Figure 3.2. It can be seen that rather than waiting for an instruction to finish execution before starting the next one, it can launch another instruction in the pipeline in order to use the hardware resources that are currently idle.

3.1.3 Single Instruction Multiple Data

Also referred to as loop-level parallelism or data parallelism, single instruction multiple data (SIMD) architectures are a form of parallelization of computing by performing the same task on parallel sets of data. This can be achieved in many ways. The most common SIMD architectures involve multiple processing lanes where the same instruction is sent to each lane, but different data might be operated on. A diagram of SIMD can be seen in Figure 3.3.

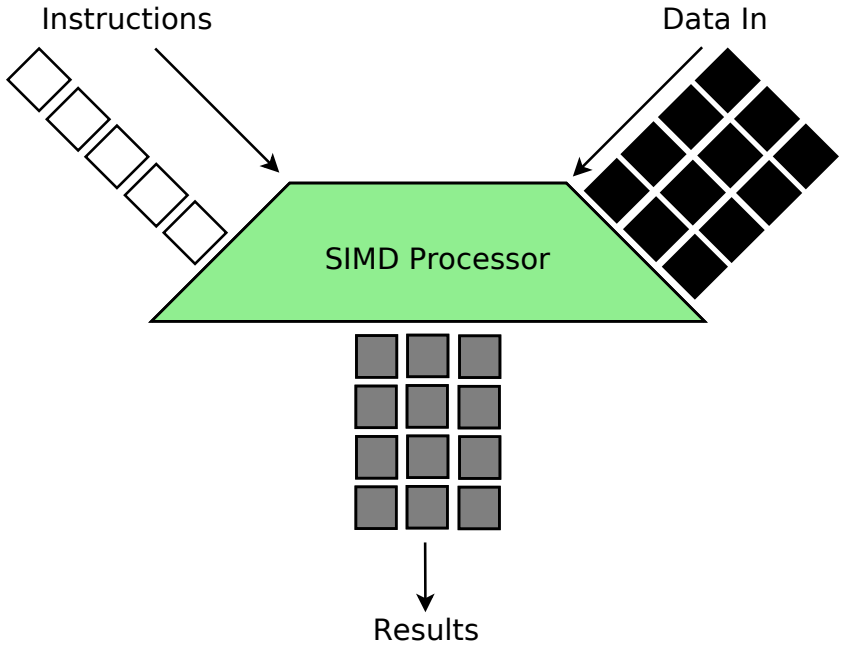


Figure 3.3: SIMD Processor Example

3.1.4 Single Instruction Multiple Thread

Single instruction multiple thread (SIMT) processors are typically SIMD systems that allow for warps to handle branches in parallel. A good example of this style of architecture is the Nvidia GPU, which has adopted the Compute Unified Device Architecture (CUDA). CUDA has made possible the idea of having hundreds to thousands of cores in a single processor that can account for many branches at the same time through very finely controlled instruction scheduling [35].

Nvidia's CUDA [28] accomplishes SIMT through the use of many graphics processing clusters (GPC) with multiple streaming multiprocessors (SM), each with many CUDA cores.

3.2 Multi-Core Simulation Environments

There are a number of useful simulators available for evaluation of digital circuits, however only a few seem to focus on energy and utilization characteristics of multi-core processors. Given the scope of this work, I will limit my review to those simulators that provide useful insight into the characterization and evaluation of multi-core processors.

3.2.1 GPGPU-Sim

GPGPU-Sim is a Graphics Processing Unit simulator developed at the University of British Columbia meant to test and evaluate wide-parallel SIMD/SIMT processors similar to that of an NVIDIA GPU. It provides many options ranging from number of cores, size of memory (cache or external dram), interconnect configurations and many more. This tool provides a very fast way to simulate a processor running custom, user defined algorithms. It also allows the flexibility of programming these algorithms in the well documented CUDA programming libraries for C/C++.

In [6] the authors use GPGPU-Sim to characterize the performance of existing CUDA applications. They show that decreasing the number of threads running concurrently on the hardware can improve performance by reducing contention for on-chip resources. They also provide an analysis of application characteristics such as dynamic instruction mix, SIMD warp branch divergence properties, and DRAM locality characteristics. Similarly [19, 50] evaluate workloads and efficiency in a GPU like processor using GPGPU-Sim.

The only drawback to GPGPU-Sim is that while it is a great tool for evaluation of

utilization patterns in a GPU-like architecture, it currently does not allow for power or energy measurements.

3.2.2 MV5

Another simulator commonly used is the MV5 Simulator [37] (based on gem5 [8]), which incorporates widely cited tools for power and area models. These power models are: Cacti 4.2 [51] for on-chip caches (which are in each core), Wattch [11] for computation cores and [43] for on-chip routers. With these power and performance models, MV5 allows for simulation of the total power consumption of a user defined multi-core processor.

MV5 simulates each event including cache hits and misses, memory accesses, alu operations, clock cycles, and many more. Essentially, almost any event that happens in the processor can be output to a log file making this simulator arguably one of the more useful tools for someone who is trying to evaluate a SIMD processor on an application level.

The MV5 simulator uses hgfractal, a SIMD thread framework that allows the coding of generic SIMD applications. This framework breaks the algorithm chain into a set of functions wherein each runs at a programmer-defined width. Each kernel instance runs on a specific subset of the data known as a "block". Partitioning the data into these smaller subsets allows for parallelization.

While MV5 has more processor event logging capabilities, it lacks the elegant programming interface that was provided for GPGPU-Sim. Nevertheless, it is a proven

simulator that provides large amounts of timing and energy data.

3.3 Power Scaling & Optimization Techniques

There are many techniques to optimize processors for area, performance, and power. This work focuses on performance/watt optimization through power scaling techniques.

3.3.1 Dynamic Voltage & Frequency Scaling

Dynamic Voltage & Frequency Scaling (DVFS) is a very common and widely used power management scheme. DVFS techniques scale supply voltages and operating frequencies to reduce power consumption which follows:

$$P \propto CV^2 f$$

There are many well-known control techniques for determining when to scale the supply voltage or clock frqeuency. In [12], the authors introduce a method for accurately estimating the potential power savings using common DVFS control techniques.

In [29], a method for using DVFS in multicore processors is presented. It is meant to scale the clock and voltages of each core to decrease power consumption. In very wide processors, they propose that in order to reduce the need for many voltage regulators per core, they can cluster cores together and scale each cluster appropriately. They limit the scope of DVFS to just core clocks.

For modern CPUs, the advanced Configuration and Power Interfcace (ACPI) [16]

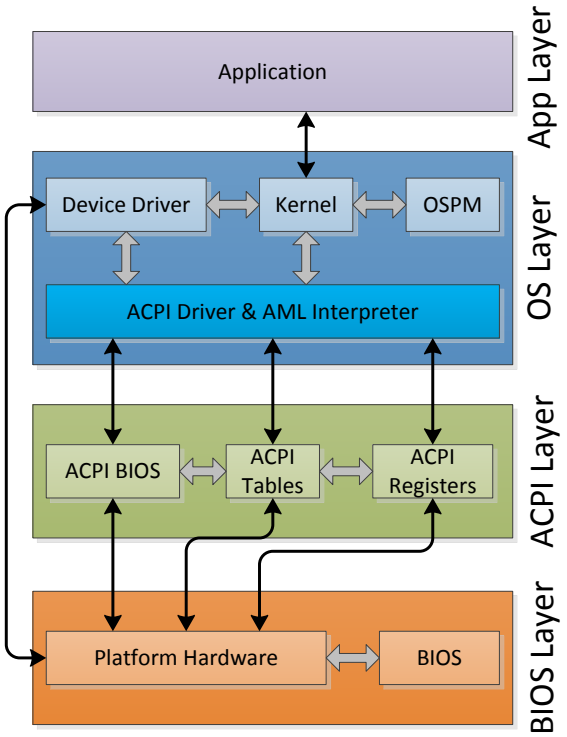


Figure 3.4: ACPI Layer Diagram

which was developed by Intel, Microsoft, Toshiba, HP, and Phoenix was adopted. ACPI allows for control of power management from every layer of the system, from the application layer through the hardware and bios as seen in Figure 3.4. One power management technique used in ACPI is the use of performance states ($P_0 \rightarrow P_n$) where P_0 operates at maximum power and frequency, and each subsequent P-state continues to follow the pattern where P_n has less performance than P_{n-1}. In these states, voltage and frequency are scaled appropriately to specific values that have previously been tested to work. The purpose of this is to give the ability to use stable DVFS points in order to reduce power consumption based on utilization and/or application needs.

3.3.2 Race To Sleep

Another power management technique popular with embedded systems is race to sleep. This technique attempts to complete a computation as quickly as possible, and then transition into a sleep state to save energy. Power savings arise from maximizing the amount of time spent in this low-power sleep state, thereby avoiding many power state transitions. In [5] the authors describe a control scheme for a race-to-halt technique that results in significant energy savings.

In [34] the authors evaluate when race to sleep is a good technique to use rather than traditional clock scaling. They look into three classes of processors. They use an OMAP, Intel Atom, and an Intel core i7. It was found that, in general scaling is better as long as it doesn't negatively affect the throughput or overall runtime of the system. It was also found that in cases where utilization is lower, using sleep states can often greatly improve energy efficiency.

Because many processors have sleep states now, using these low power modes is often the easiest way to save power as it requires no extra hardware beyond the power gating circuitry. In the previously mentioned ACPI [16] control, in addition to performance states there are also CPU power states and Sleeping states. These both allow for switching into lower power modes, but in this case the "deeper" the sleep state the processor is in, the fewer capabilities the processor has, and the higher the latency before the processor can return to normal operation.

3.3.3 System Level Power Optimization Techniques

There have been a number of proposed system level optimizations for power. The authors of [47] lower the power consumption of an embedded core by clustering clock gating to improve energy efficiency. This is done on an 8051 based multi-core processor.

In [7], an accurate and fast simulator of the power consumed in various IP blocks of a System-on-Chip (SoC) designs are developed. In [42], the authors decrease the power consumed within a single-core microprocessor using software based optimization techniques.

In [53], the authors propose optimizing a multi-core processor for specific scientific computations using sparse matrix-vector multiplication as an example (as it is one of the most heavily used kernels in scientific computing). They demonstrate significant performance increases compared to previous implementations. Game theory is used for scheduling tasks in [1] on multi-core processors for optimization of performance and energy.

Clock gating is another commonly used technique for power savings in synchronous logic, minimizing dynamic clock power. Previously, clock gating has been shown to exhibit approximately a 25% power improvement [36], however performance/watt does not change, as no computation occurs when the clock is stopped. The theory behind clock gating is that by not clocking portions of logic controlled by or containing flip-flops, it keeps that portion of the circuit from switching states. By forcing a circuit to remain in the same state, less dynamic energy is used. An example of how to gate a

clock is shown in Figure 3.5.

Power gating is a well known technique for minimizing leakage current during idle periods. Recent work in [49] suggests energy savings up to 19% for SIMD multi-core architectures, based on resource utilization patterns.

The basic theory behind power gating is that if a logic block can be deactivated so there is no power to the transistors, then there will be less leakage current. An example of this is shown in Figure 3.6. This typically means that the internal capacitances of the logic will take longer to charge when the power gate returns power to the circuit.

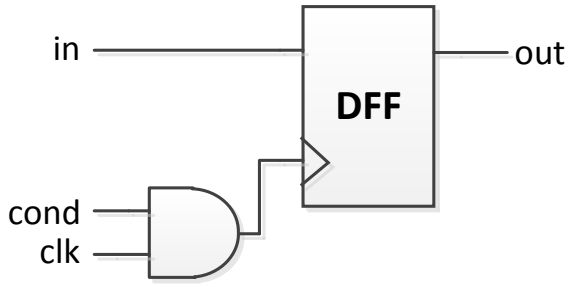


Figure 3.5: Clock Gating Example Circuit

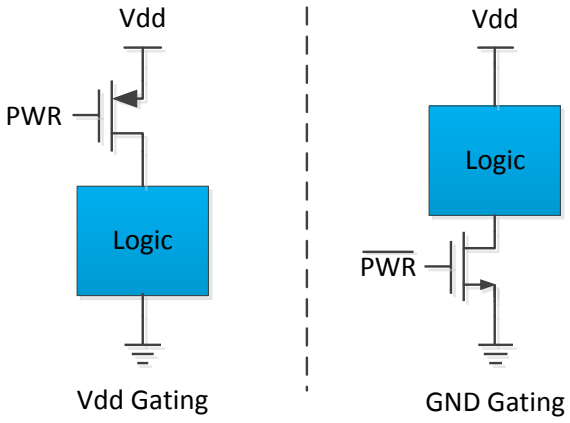


Figure 3.6: Power Gating Example Circuit

3.4 Near/Sub-V_{th} Operation For Low Energy Computation

Near/Sub-V_{th} operation of digital circuits is becoming more prevalent as transistor sizes decrease. Moore's law states that the number of transistors that can be placed inexpensively in an integrated circuit will double every two years [39, 40, 48]. This can be seen in Figure 3.7.

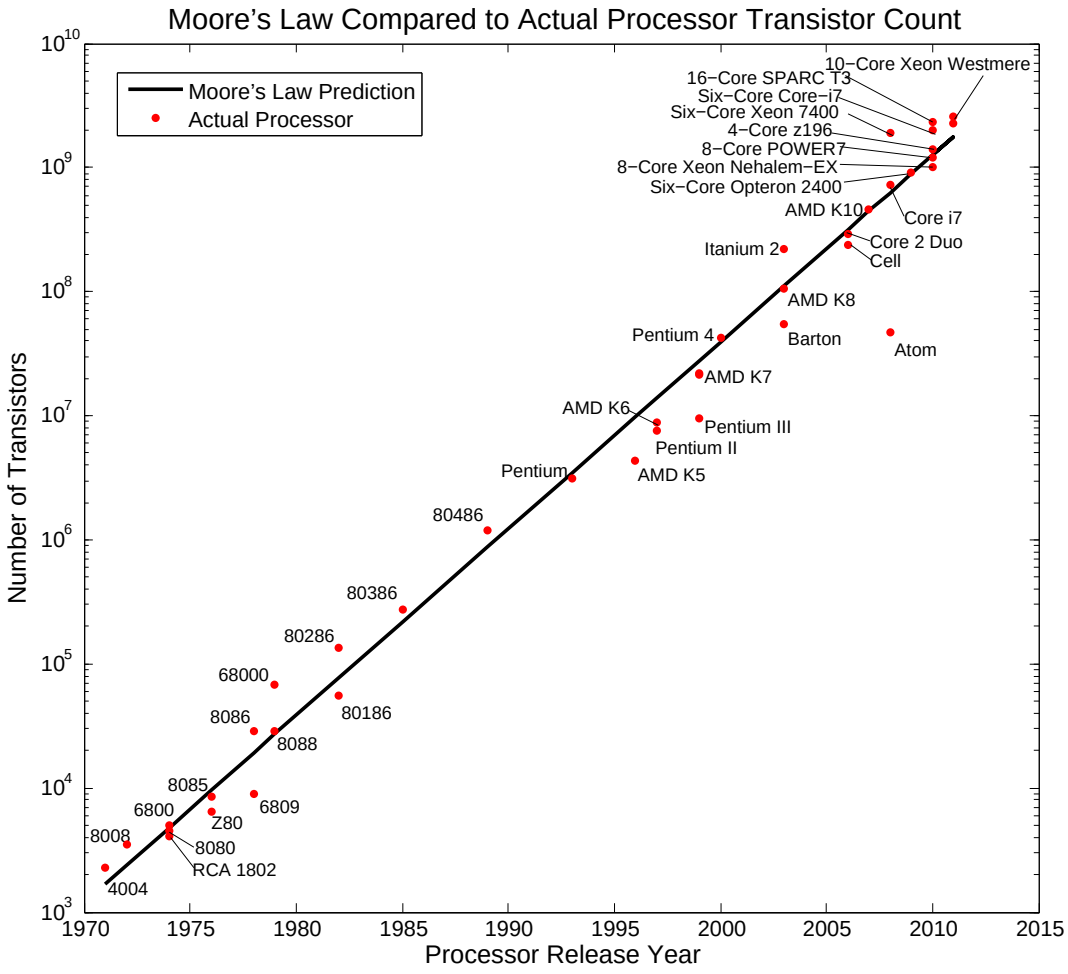


Figure 3.7: Transistor Number vs Time Compared to Moores Law

With the doubling of transistors, it makes sense to also expect the size of these transistors to trend downward. With higher density transistors, total leakage current rises due to the quantum phenomenon in semiconductors for mobile charge carriers to tunnel through an insulating region. This occurs exponentially more often as the thickness of the insulating region decreases. This also leads to the inability to scale the transistor threshold voltage at the same rate as transistor size (Figure 3.8).

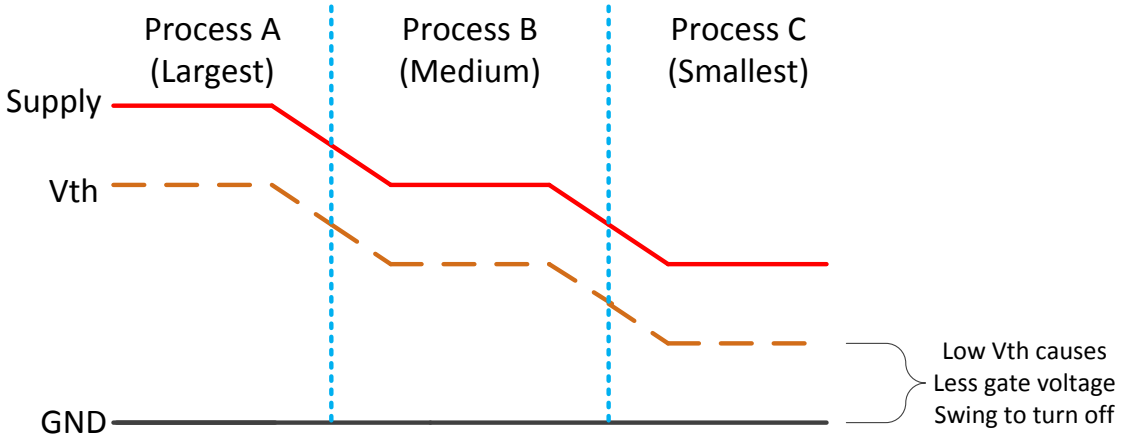


Figure 3.8: V_{th} Scaling with Process

3.4.1 Benefits

One way to combat these scaling issues is to operate devices in the sub-near threshold region. Research has shown that operating in these regions can lead to order of magnitude improvements in energy efficiency over conventional processors at the same technology node [21]. One example, a 16-bit 1024-point .18m FFT test-chip with Vdd of 0.35V (V_{th} = 0.45V) achieved 155nJ per FFT at 10KHz [52]. As a comparison, an 8-bit processor [22] achieved 540fJ /operation at a Vdd of 0.3V and 160KHz operating frequncy.

3.4.2 Complications

These high computational efficiencies, however, come with a cost in that operating frequencies must be lowered (often significantly) and timing variability/timing violation are more common due to added sensitivities to process variation such as random dopant fluctuations (RDF) while operating below the threshold voltage [21]. Because of the low clock frequency requirement, processors operating within these regions are forced to move towards wide parallel architectures for increased performance. This requirement will extrapolate higher as device sizes trend downward due to the afore mentioned affects of transistor scaling. Programmable architectures will also need to rely on efficient control, such as wide SIMD/vector execution [30, 54] to maintain these efficiencies.

The solution is not necessarily as simple as adding more parallelism however, due to wide SIMD architectures timing variability which is caused by having many ALUs. Operation at near-threshold is already limited by variability, which is degrading with

transistor scaling. Random dopant fluctuations, such as threshold voltage mismatch due to statistical fluctuations of the dopant atoms severely alter the transistor current, thereby affecting the logic delay. As the supply voltage is reduced, the transistor no longer acts in the traditional velocity saturated manner and begins to behave as a bipolar device.

This strong dependence of current draw on V_{th} in near-threshold results in as much as an 18x degradation in transistor current and logic delay across Monte Carlo variation [21]. To make varation even harder to predict, critical RDF threshold variations are not related to any spatial distances. This means that spatial correlations can not be drawn for parallel functional units which would otherwise allow similar timing delays between vector units as is shown in [15].

3.4.3 Previous Solutions

Previous solutions [22, 52] have shown that using non-minimal device sizes and longer logic chains can reduce variation. This however, effectively moves in the opposite direction of transistor scaling, and will become more problematic moving into deep submicron nodes.

Other research [10, 17] identified methods of dynamic timing variation tolerance within a scalar pipeline. The basis of which is to dynamically detect timing errors and correct them by either flushing the pipeline and re-executing the instruction with more relaxed timing or stalling the pipeline for one cycle while waiting for the correct result to be generated, and then proceeding with execution. The prior approach while easier to implement in faster and more complex pipelines, has a more severe performance penalty.

The latter approach has a smaller performance and power penalty.

In typical parallel designs, all functional units operate off of a single clock in lock step, making any error encountered in a single stage result in a stall for all the functional units within that processing element. This makes the typical method of stalling and flushing the system problematic and inefficient and amplifies the performance/watt penalty by the pipeline width when handling an error. This realization combined with the afore mentioned increase in timing variation as parallization increases width, makes it necessary to come up with an expandable SIMD architecture that can handle these timing variations.

3.4.4 Synctium

Synctium [31], defined as "a near-threshold stream processor for energy constrained parallel applications" is a SIMD processor developed at Oregon State University that focuses on solving the previously mentioned problems with near V_{th} operation of wide SIMD processors. It achieves near energy-optimal operation by combining efficient parallel computation techniques with the use of near-threshold circuits. This architecture, through lane weaving and decoupled instruction queues solves much of the static and dynamic timing variations respectively.

Decoupled instruction queues allow for each ALU to have its own list of instructions making it possible for a single lane to stall, while other lanes may continue through their own queue. A diagram of this can be seen in Figure 3.9. This is limited only by the length of the queue and number of system stalls, such as concurrent memory accesses.

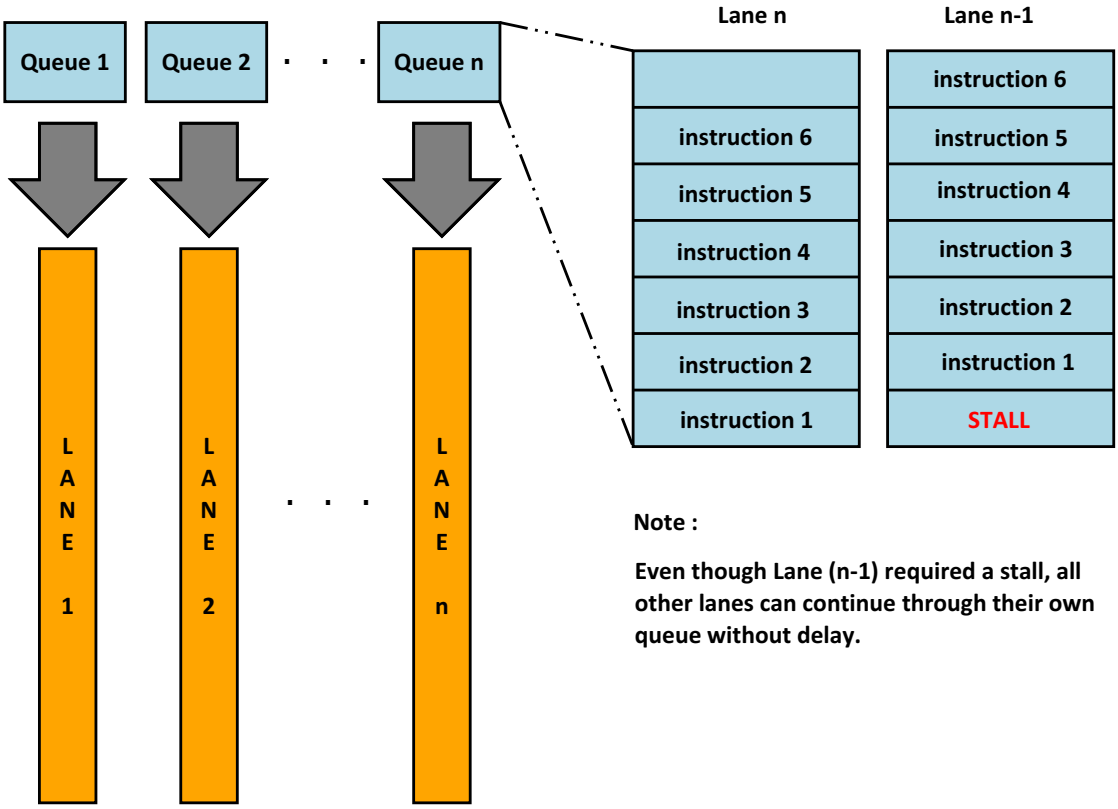


Figure 3.9: Decoupled Queues

Lane weaving allows for slower or defective lanes to be "weaved" around with little delay similar to [20]. Each decode and fetch stage is connected to the processing lanes adjacent to it. A diagram of lane weaving can be seen in Figure 3.10. This technique requires that at least one lane at a time be inactive. In Synctium, 10 lanes were presented with 8 active lanes, allowing for 2 lanes to be inactive at any given time. The two lanes that cause the greatest static timing variation are chosen for this purpose.

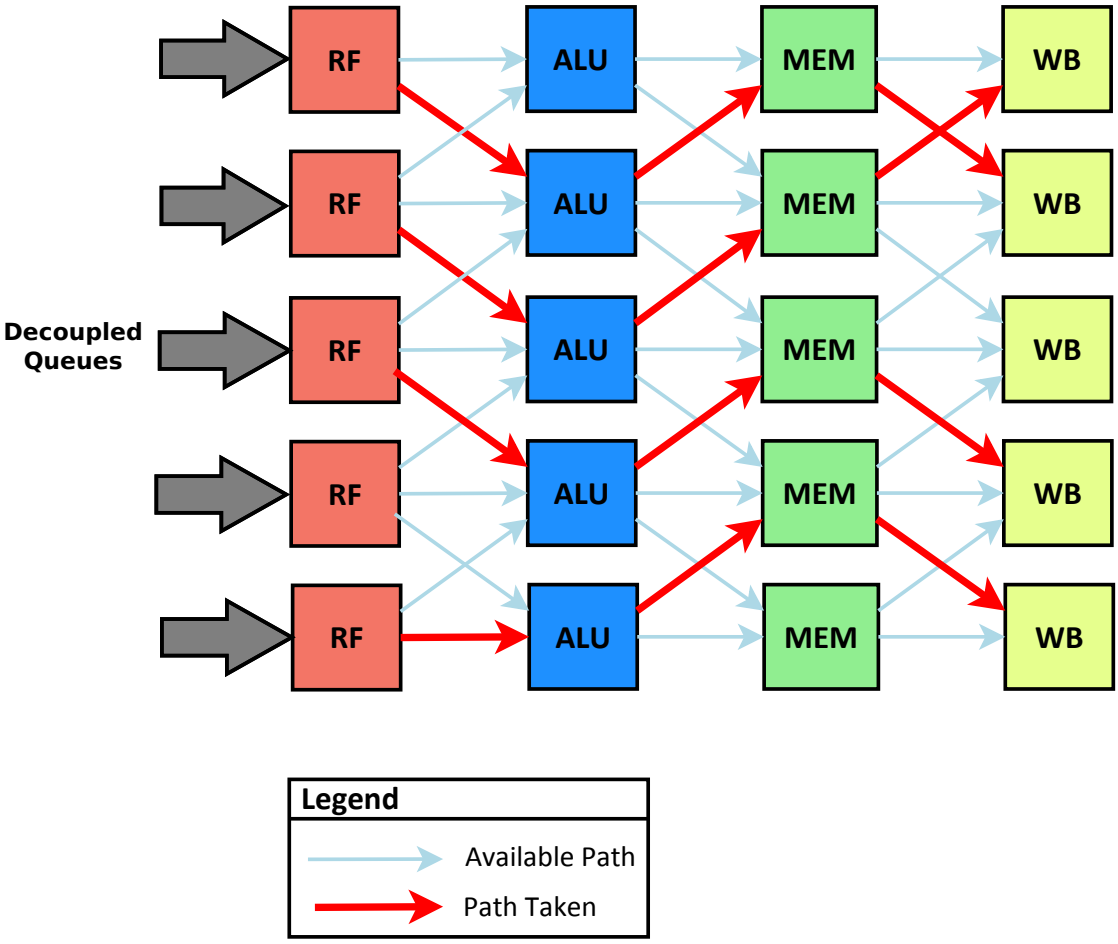


Figure 3.10: Lane Weaving

Chapter 4 – Overview of Approach

A method for improving performance/watt of a multi-core SIMD processor using explicit a priori power and performance scheduling of each hardware component (core, link, and memory) is presented. Due to predictable utilization patterns for a known signal processing chain, an example electroencephalography (EEG) application is simulated and optimized. This application is representative of common bio-medical sensor processing requirements [9]. It should be noted, that while scheduling of individual components has been done in the past, no reference to scheduling all three major components (core, link, and memory) could be found.

4.1 Proposed Developement Flow

First, we model an embedded multi-core system using a simulator based on the MV5 framework [37] running algorithm blocks that follow the Filter Bank Common Spatial Patterns (FBCSP), which is the EEG example application [3]. Next, after accurate characterization of the hardware utilization and power consumption using well-known power models [51, 11, 43, 38] that have been embedded into the simulator, a greedy search algorithm is applied to find the optimal performance/watt, varying each component's bandwidth and power. This greedy search optimization schedules the required bandwidth and power settings for each component of the embedded multi-core processor

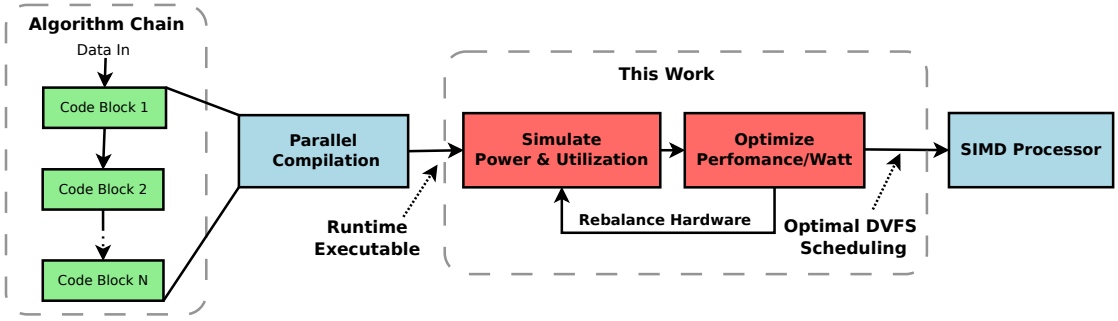


Figure 4.1: Proposed Development Flow

before run time. This approach to pre-scheduling hardware utilization, based on explicit software algorithm workloads, provides significant improvements in performance/watt with small degradations to execution time. The final step is to compare the application-directed power scheduling method with traditional power savings techniques such as conventional Dynamic Voltage and Frequency Scaling (DVFS) [12] and "race to sleep" [5]. The proposed development flow is outlined in Figure 4.1.

4.2 Example Application Used

The FBCSP [3] based EEG classification, which is representative of other similar signal processing applications [9] was used as an example application. An overall overview of this EEG classification algorithm is detailed, and each block within this signal processing chain is defined. This processing chain has similar components as would be required in most embedded medical devices. The importance of this application to this work is that it both represents a real need in embedded medical electronics, and is comprised of very different algorithms that have different hardware utilizations and timing

requirements making it a good example to attempt to optimize.

4.2.1 Application Description

In continuous encephelaography (EEG) sensing applications, EEG data must be gathered and processed constantly in order to detect possible medical conditions such as seizures [13]. EEG data is continuously gathered over 16-channels. It is then processed and classified in 1-second windows (1000 samples at a 1kHz sampling rate). The result of this classification can be used to determine whether an anomaly has occured. If it does produce a properly classified event, the anomalous data is then encrypted (RSA) and sent to memory and/or transmitted. Non-anomalous data does not need to be stored. An overview of the example filter chain can be seen in Figure 4.2. This allows only the useful information to be stored or transmitted saving precious memory and RF energy. It also makes the overall output of system consist of exclusively useful information.

4.2.2 Application Details

The classification technique used is based on the Filter Bank Common Spatial Patter algorithm (FBCSP). There are many crucial steps in the FBCSP classification process. The first step is to to notch out specific frequencies using a bank of band-pass filters. In this implementation we use 16 banks, each with a 4Hz wide passband, starting at 4Hz (e.g. bank-1 filters 4-8Hz, bank-2 8-12Hz, etc). These filters are the first kernel functions in our application, implemented as finite impulse response (FIR) filters with

32 coefficients.

The second step in FBCSP is to perform spatial filtering. These spatial filters perform mixing of the EEG channels to construct correlating spatial channels to the output classes. Some common training methods for the spatial filters can be found in [3]. We construct sixteen spatial channels per bank. Common spatial filtering (CSP) is the second kernel function in our signal processing chain.

The third process in the chain is to reduce spatial channels to features. This is done in a two-step process commonly known as feature extraction. In the first step, the spatial channels are multiplied by a mixing matrix. The second step is to find the variance of each of these mixed channels resulting in a 512-element feature vector. These two steps are the third and fourth kernel functions in the signal processing chain.

Once the feature vector is obtained, feature classification is performed. This application uses K-Means clustering, where we compute the Euclidean distance that our vector is from a set of pre-trained points. The shortest distance between these points is the class that most resembles our feature vector. This requires computing the distance from all 256 trained locations with a three step processes. The first step is to square the difference between each element of our feature vector and every trained point. The second step sums these squared points. The last step determines the shortest distance again. This three-part process is comprised of the 5th, 6th and 7th kernel functions of the processing chain.

After each of these 7 FBCSP blocks are complete, we know whether the EEG sample is anomalous which is dependant on which class it matches the most. If the EEG sample is determined to be anomalous we encrypt it using RSA encryption, which is

implemented with 4-byte blocks. This encryption forms the 8th and final block of our entire example signal processing application.

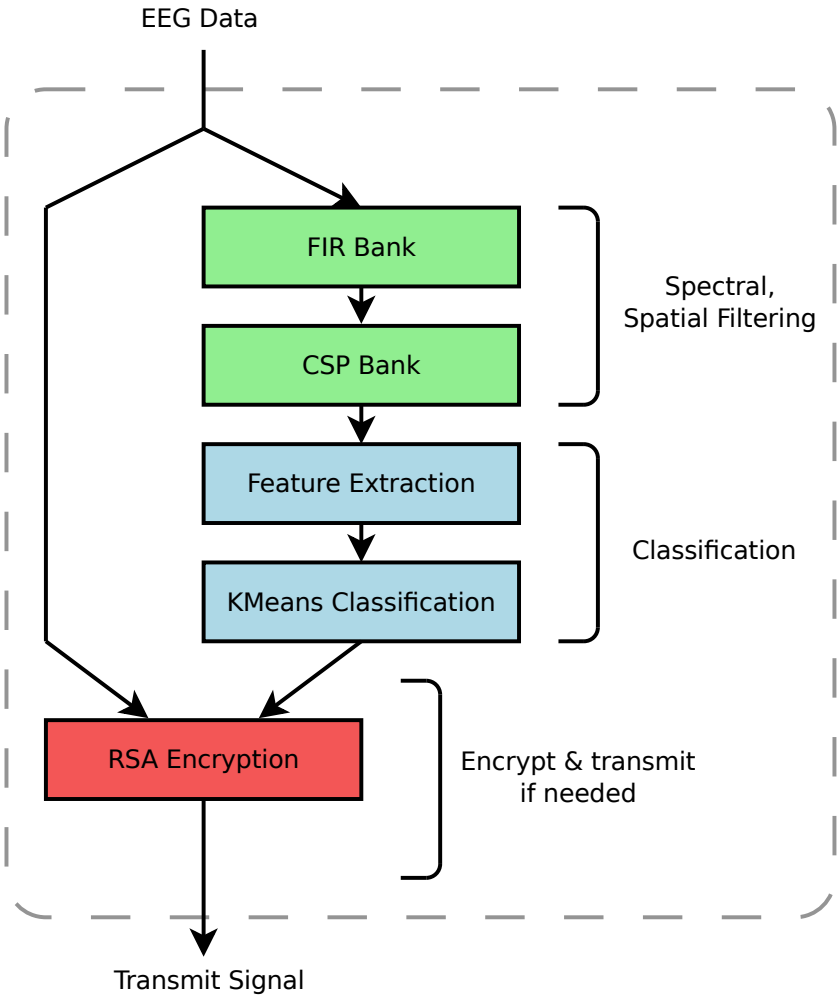


Figure 4.2: EEG FBCSP Processing Chain

4.2.3 Kernel Function Grouping

The final step in preparing the application for performance/watt optimization is to choose which kernel functions should be grouped together before optimization. This is particularly useful when there is a string of many short code blocks. By grouping some kernel functions together it can reduce the number of DVFS changes required to execute the entire application. Grouping kernel functions too much however can limit potential performance/watt gains from finding the optimal operating point. In this example, a few rules were employed for grouping.

Non-adjacent kernel blocks are not grouped together because there will already be a DVFS change expected between these blocks. Also, a limit on the total execution time of a kernel function is adopted, such that if its time duration is too small it is then grouped with an adjacent block. As a requirement, any kernel that executed in less than $20\mu s$ on the super V_{th} processor at full speed was grouped with another kernel function. These kernel functions that require grouping are paired with a neighbor block that exhibits the most similar characteristics determined by analyzing the kernel implementations. In this application there were two short kernel functions. The feature-mixing step is the first. It was combined with the CSP filters due to their similiarity because they both consist of primarily matrix multiplications. The second short kernel is the final step of K-Means, which is combined with the second step of K-Means because they share more similar data than grouping K-Means with the encryption kernel. These groupings were kept the same on the near V_{th} processor as well for comparison, even though all of the kernel functions operated well above the $20\mu s$ threshold. Table 4.1 shows all of the kernel

mappings.

Kernel Functions	Kernel Numbers	Block Numbers
FIR Bandpass	1	1
CSP, Feature Mixing	2, 3	2
Feature Variance	4	3
K-Means Step 1	5	4
K-Means Step 2,3	6,7	5
Encryption	8	6

Table 4.1: Kernel Function to Block Mapping

4.3 Greedy Search

A greedy search algorithm is used to optimize the algorithms of the proposed filter chain, maximizing performance/watt using DVFS of each component (core, memory and interconnects). A flowchart of our greedy search can be seen in Figure 4.3. The greedy search algorithm starts with default DVFS settings for each component of the simulated system. We define a maximum and minimum step size for each component with respect to that components clock frequency. Each loop of the optimization launches six simulations in parallel (one maximum stepsize above and below each of the three components described above). This results in initial frequency settings for each component to be set to both the the maximum and minimum values for the current loop. Whichever of the six simulations that resulted in the greatest performance/watt improvement within the simulation time constraints is selected for the next iteration of the greedy search. If none of the simulations improve performance/watt within the specified time increase, the optimizer halves the maximum step size and iterates again with these new settings.

This process is iterated until all three maximum step sizes have been reduced to less than their corresponding minimum step size.

Greedy search was used for the optimization algorithm because of the benefit in simulation time over a fine-grained grid search, which would require weeks→months in order to optimize depending on granularity. The proposed greedy search allows for optimization within 24 hours in most cases. So while a greedy search by definition is not guaranteed to find the globally optimal solution, it can be shown that we are still able to get significant gains with a greedy optimal solution. Greedy search algorithms have been explored to optimize the expected variation in system-level DVFS in [23].

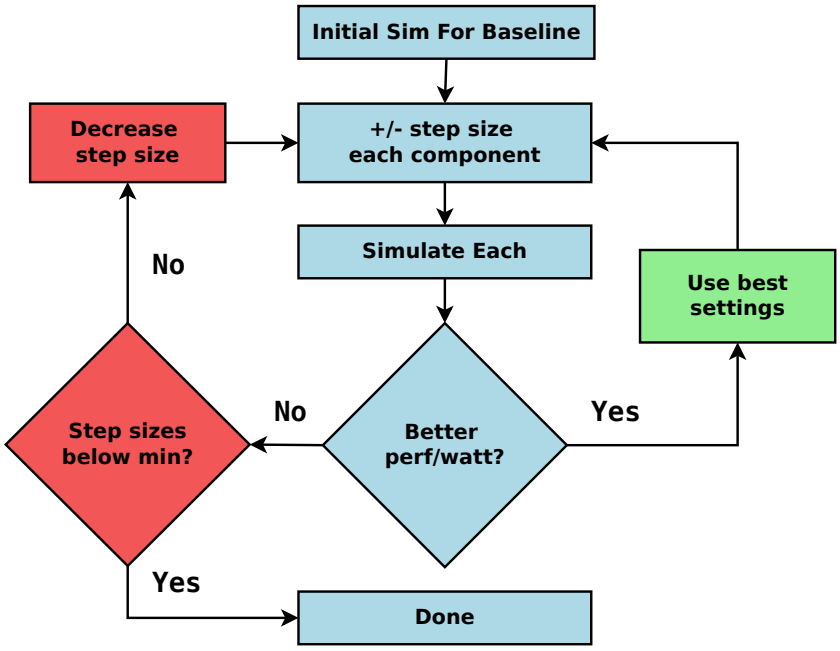


Figure 4.3: Greedy Search Flowchart

Chapter 5 – Architectures Explored

As a comparison, two architectures were explored. The first is a 4-core with 8-lanes/ core architecture. This was built into the MV5 simulator by default, and was used as an example of a SIMD system operating at supplies above V_{th}. Synctium [31] was used as the core architecture as a comparison to a system that can move its DVFS curve to near threshold operation.

As part of the power management for the entire system, the assumption is made that we have the capability to apply DVFS to each of the major components independently (core, on-chip routers, and off-chip memory). Previous works [27, 32] suggest that efficient on-die voltage regulators with multiple outputs are soon becoming a viable substitute for slow off-chip regulators. Another possibility for applying separate DVFS is to utilize multiple power gates for multiple supply rails [14]. In this work, we do not advocate a particular circuit implementation for either of these approaches, as the power overhead for scaling between each voltage domain is negligible for our application.

It should be noted that the nominal starting frequencies for both architectures were the same, and are outlined in Table 5.1.

Component	Nominal Operating Frequency
Cores	500MHz
Memory	600MHz
Links	1 GHz

Table 5.1: Component Nominal Operating Frequencies

5.1 Super V_{th} Architecture

The simulated system architecture can be seen in Figure 5.1. The system consists of four homogeneous cores, all capable of running SIMD applications. The details of the system configuration are summarized in Table 5.2. These specifications were chosen because they are common configurations for high-performance, quad-core embedded processors with SIMD extensions. The cache sizes were chosen to be small, as is common with embedded applications.

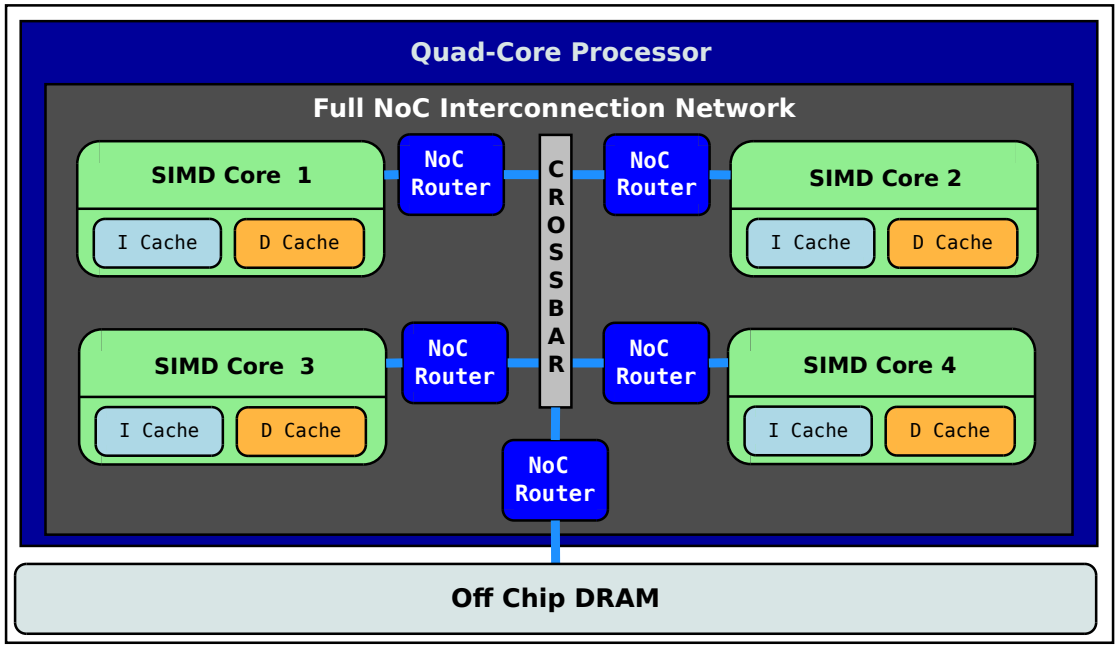


Figure 5.1: Super V_{th} Simulated Architecture

Component	Quantity	Unit
# Cores	4	Alpha Cores
# SIMD Lanes/Core	8	Lanes
Core I-cache Size	512	Bytes
Core D-cache Size	1k	Bytes
DRAM Memory Size	1G	Bytes
NoC Routers (Full Duplex)	5	Routers

Table 5.2: Super V_{th} System Specifications

5.2 Near V_{th} Architecture

The simulated system architecture can be seen in Figure 5.2. The system consists of a single synctium [31] core with 8 lanes. The system configuration is summarized in Table 5.3. These specifications were chosen to match the previous architecture as closely as possible.

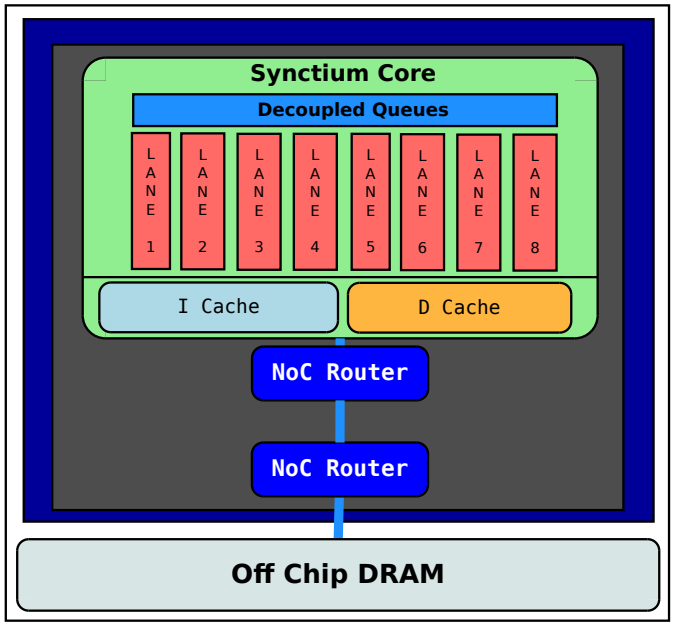


Figure 5.2: Near V_{th} Simulated Architecture

Component	Quantity	Unit
# Cores	1	Synctium Cores
# SIMD Lanes/Core	8	Lanes
Core I-cache Size	512	Bytes
Core D-cache Size	1k	Bytes
DRAM Memory Size	1G	Bytes
NoC Routers (Full Duplex)	2	Routers

Table 5.3: Near V_{th} System Specifications

Chapter 6 – Optimization Techniques Explored

In order to analyze the effectivness of a priori scheduled DVFS, two metrics for optimization were considered. The first was to optimize performance/watt within a percent of performance loss. This was considered for the 1%, 5%, and 10% cases. This technique is compared with convential DVFS. The second style of optimization is when the designer knows how often the algorithm must run giving an effective time window. This style of optimization is compared with the traditional race to sleep method.

6.1 Conventional DVFS

Conventional DVFS in this case is where the clocks are scaled at the beginning of the algorithm based on how the device is utilized at that point in time and will remain the same across the whole algorithm. This is how most systems use DVFS, assuming that the algorithm will remain at a similar utilization. They will find clock settings that work every time and might save some energy consumption but do not consider the change in utilizations that occur in the different sections of the algorithm.

6.2 Race To Sleep

As mentioned in Chapter 3, this technique is used in many embedded applications where sleep states are available. These states allow the processor to consume less energy as it

turns portions of the processor off. For this example, the assumption was made that in the best case scenario only the leakage current of the circuit would consume energy.

6.3 Algorithm Block a priori Scheduled DVFS

There are two methods for constraining greedy search. The first is to search for the optimal DVFS settings for each section (block) of the filter chain algorithm independently. For each of these blocks, different DVFS settings for each hardware component results in optimal performance/watt. Hence, this method improves performance/watt by exploiting the differences in hardware utilization of the block within the filter chain algorithm. For example, a block that is very compute intensive will exhibit different optimal DVFS settings from a block that is very memory intensive.

6.4 Full Algorithm a priori Scheduled DVFS

The second method for constraining the greedy search is to only allow DVFS settings for the entire filter chain. Here each block of the chain will operate with the same DVFS settings as adjacent settings, but the entire filter chain is still optimized for performance/watt. This method would be most appropriate with an application where changing the DVFS settings was not feasible on a fine-grained time basis.

Chapter 7 – Results

The results are broken into two sections. The first consists of the simulations and optimizations performed on the super V_{th} architecture and the second on the near V_{th} example.

7.1 Super Threshold Scaling of Core, Link and Memory Clock

As mentioned previously, there are two main categories for comparison. The first consists of optimizations of the power consumption given a small performance loss, which is compared to traditional DVFS. Improvements in total power were evaluated given execution time increases (performance loss) of $1\%, 5\%$ and 10%. The other category consists of power optimizations within a given execution time window, which is compared to a race to sleep technique. Given the example end-to-end algorithm nominally takes $72ms$ to execute, time windows of $100ms, 500ms$ and $1000ms$ were explored.

7.1.1 Perf/Watt Optimizations for Small Performance Losses

In Figure 7.1, the performance/watt gains are plotted for the three performance loss values explored ($1\%, 5\%$, and 10%). It can easily be seen that both the block DVFS and algorithm DVFS with a priori scheduling exhibit significantly better performance/ watt gains. One interesting observation is that the algorithm DVFS and standard DVFS

optimizations result in less performance/watt gains for 10% performance loss than the 5% performance loss case. This can be explained due to greedy optimization, which cannot guarantee a globally optimal result. As seen in Figure 7.1 our greedy search still provided significant gains over conventional DVFS. Another interesting result is when only a 1% performance loss was allowed. In this scenario, both block DVFS and algorithm DVFS optimization results in 51% and 61% performance/watt improvements respectively. This is a substantial improvement for very little performance cost.

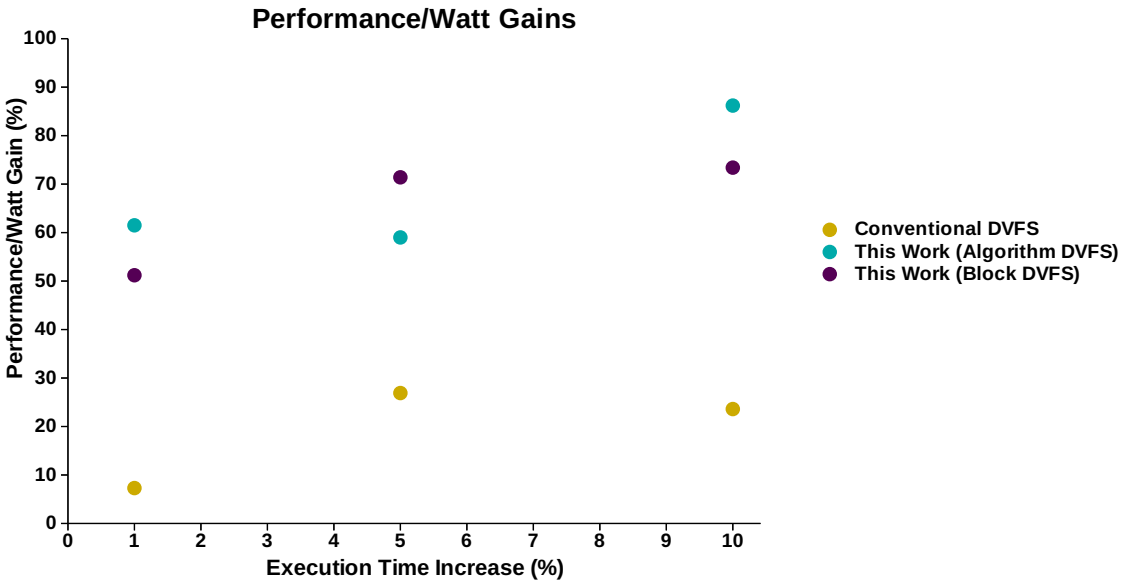
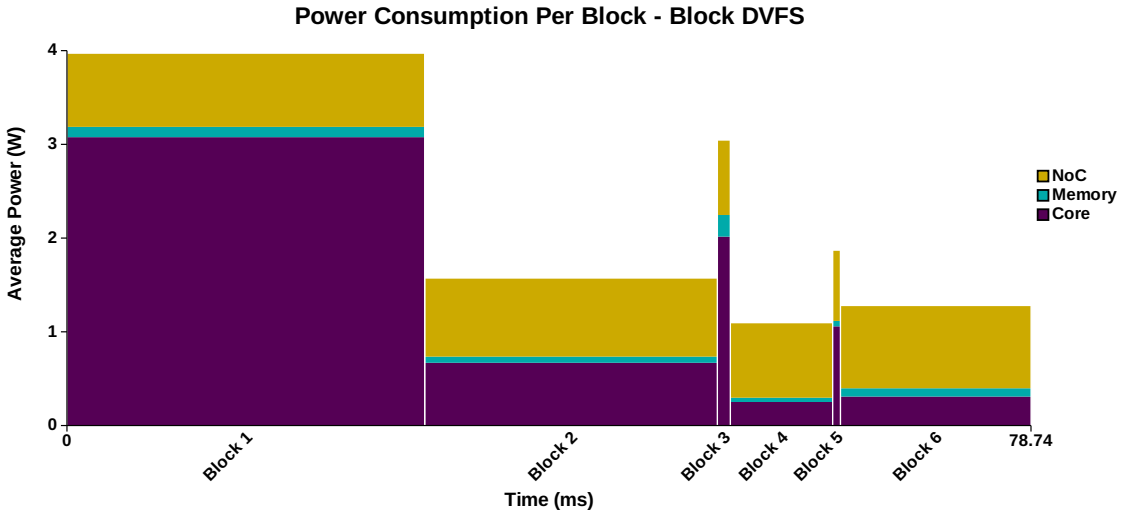


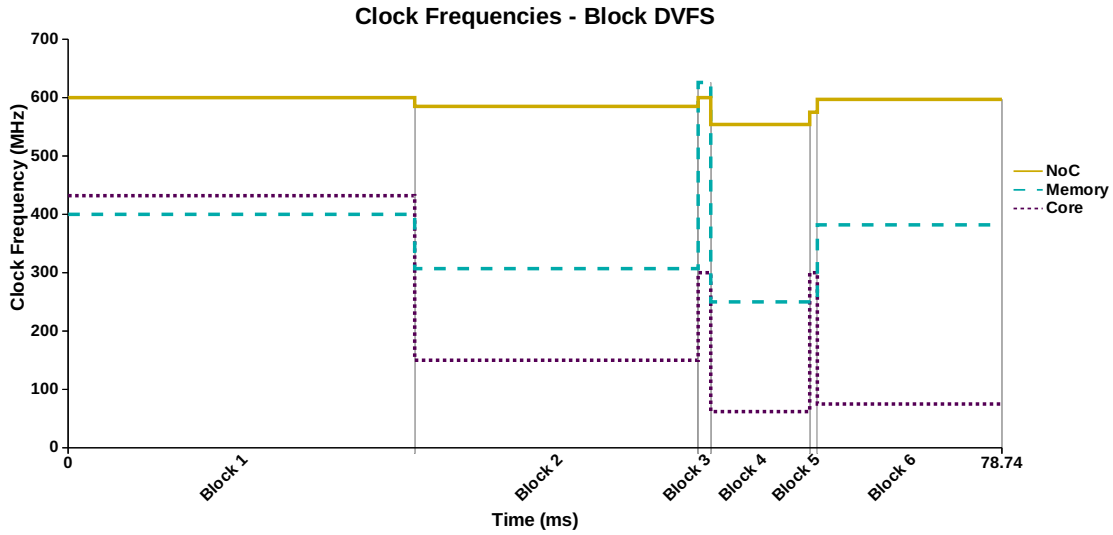
Figure 7.1: Super V_{th} Performance/watt comparison between conventional DVFS and this work (optimized for algorithm DVFS and block DVFS).

The comparison between block DVFS versus algorithm DVFS optimization provides further insight into the optimization process. Initial thoughts might create the assumption that the finer control of block DVFS would result in higher performance/watt than algorithm DVFS (because each sub-block is tuned independantly). This was not the

case however due to the large portion of the total power that was spent within the FIR algorithm (kernel-1). The algorithm DVFS optimization was able to allocate more of the total allotted time to this block in order to decrease the clock further, thereby resulting in improved performance/watt. Block level a priori scheduling results can be seen in Figure 7.2 while algorithm a priori scheduled results are in Figure 7.3.

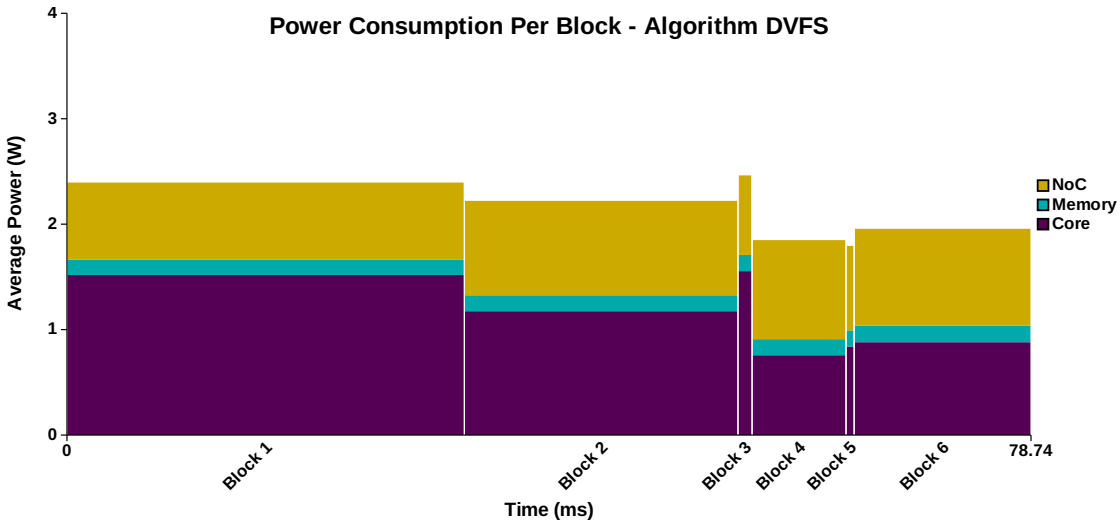


(a)

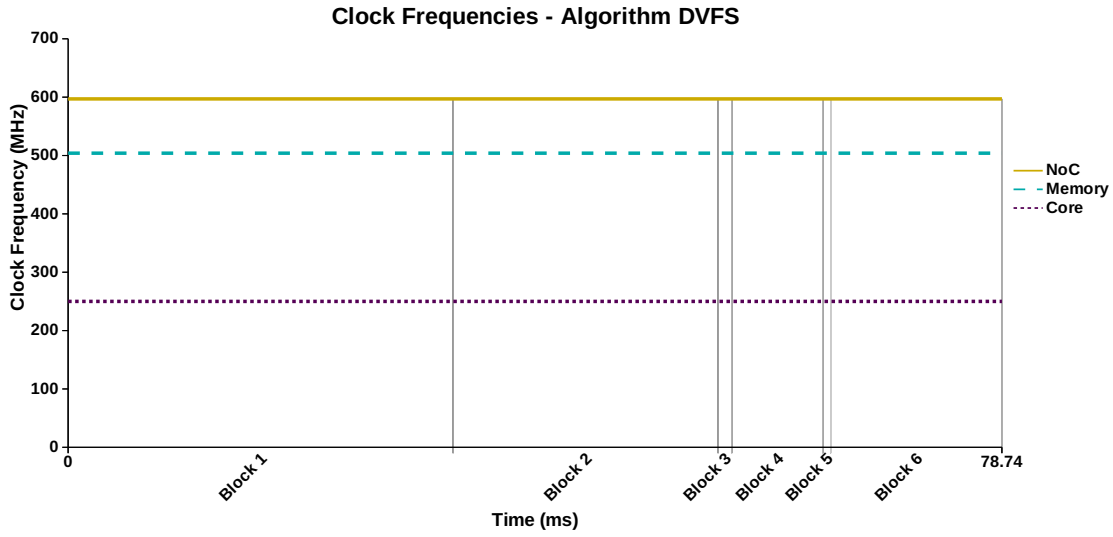


(b)

Figure 7.2: Super V_{th} - Small performance loss optimization results (at 10% performance loss): (a) Block DVFS power consumption, (b) Block DVFS component frequencies



(a)



(b)

Figure 7.3: Super V_{th} - Small performance loss optimization results (at 10% performance loss): (a) Algorithm DVFS power consumption, (b) Algorithm DVFS component frequencies.

7.1.2 Real-Time Constraint Optimization

The time window constrained optimization results can be seen in Figure 7.4. These results clearly show that both of our a priori scheduled DVFS optimization methods (block DVFS and algorithm DVFS) greatly decrease average power when compared with the conventional race to sleep method. Since the algorithm takes a minimum of $72ms$ to complete, we observe a much larger power reduction as the time window becomes much greater than $100ms$.

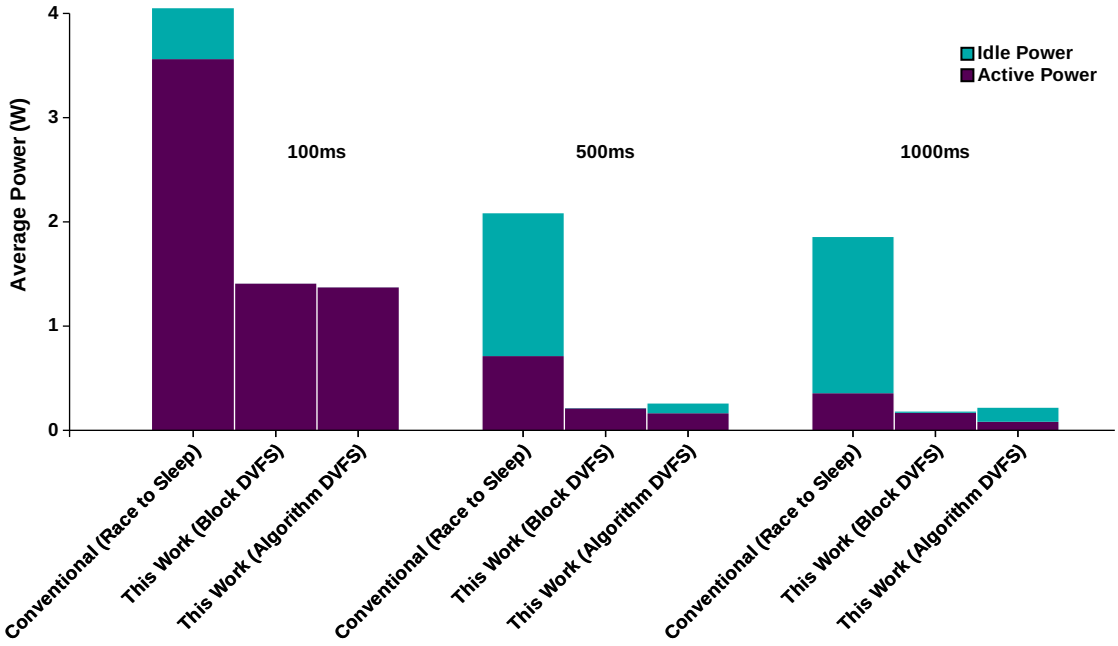


Figure 7.4: Super V_{th} Power comparisons, given a real-time constraint, between conventional race-to-sleep and this work.

7.2 Near-V_{th} Scaling of Core, Link and Memory Clock

The same tests for the Super V_{th} case were performed with the near V_{th} models. One test was of optimizations of performance/watt given a small performance loss, which is compared to traditional DVFS. These small performance losses were $1\%, 5\%$ and 10% runtime increases. The second category was performance/watt optimization within a given execution time window, which is compared to the race to sleep technique. Given the example end-to-end algorithm nominally takes $3.88s$ to execute, time windows of $4s, 7s$ and $10s$ were explored. It should be noted that these times are significantly larger than the Super V_{th} case, this is due the the architecture differences and that Synctium is operating with 8 lanes only rather than 4 cores with 8 lanes as in the previous architecture.

7.2.1 Perf/Watt Optimizations for Small Performance Losses

In Figure 7.1, the performance/watt gains are plotted for the three performance loss values explored ($1\%, 5\%$, and 10%). It can easily be seen that, like the super V_{th} case, both the block DVFS and algorithm DVFS with a priori scheduling exhibit significantly better performance/watt gains over conventional DVFS. This can be seen in Figure 7.5. One interesting result is that all optimization techniques have better numbers than in the super V_{th} example. This could be a manifestation of the advantages of near V_{th} operation. It is also important to note that at 1% performance loss, both block DVFS and algorithm DVFS optimization results in 50% and 95% performance/watt improvements respectively. This is once again a substantial improvement for very little performance

cost.

Once again, the algorithm level a priori scheduled DVFS was generally the best option, while block level a priori scheduled DVFS was in some cases very close (see 10% in Figure 7.5).

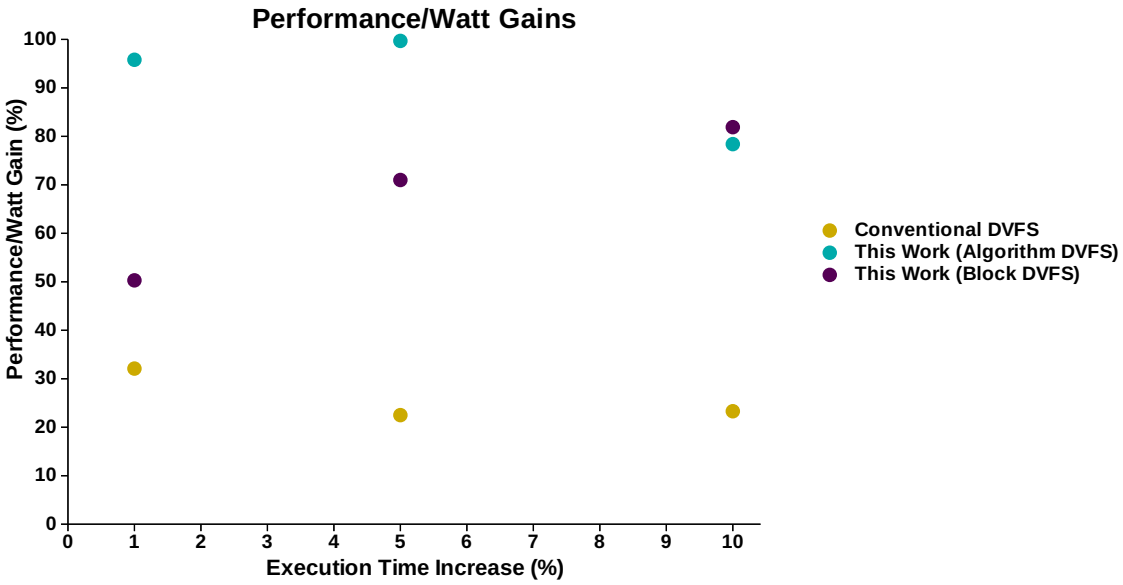
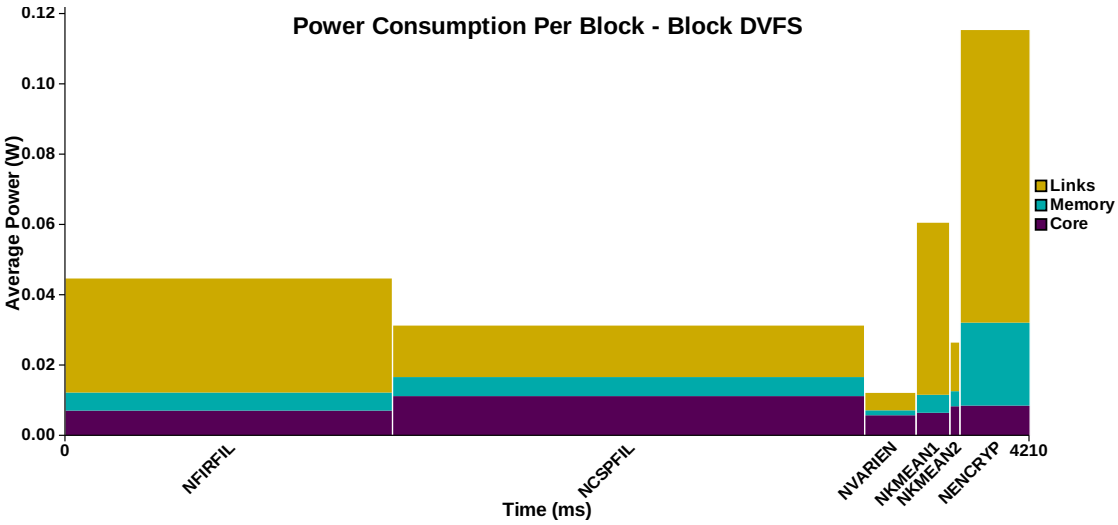
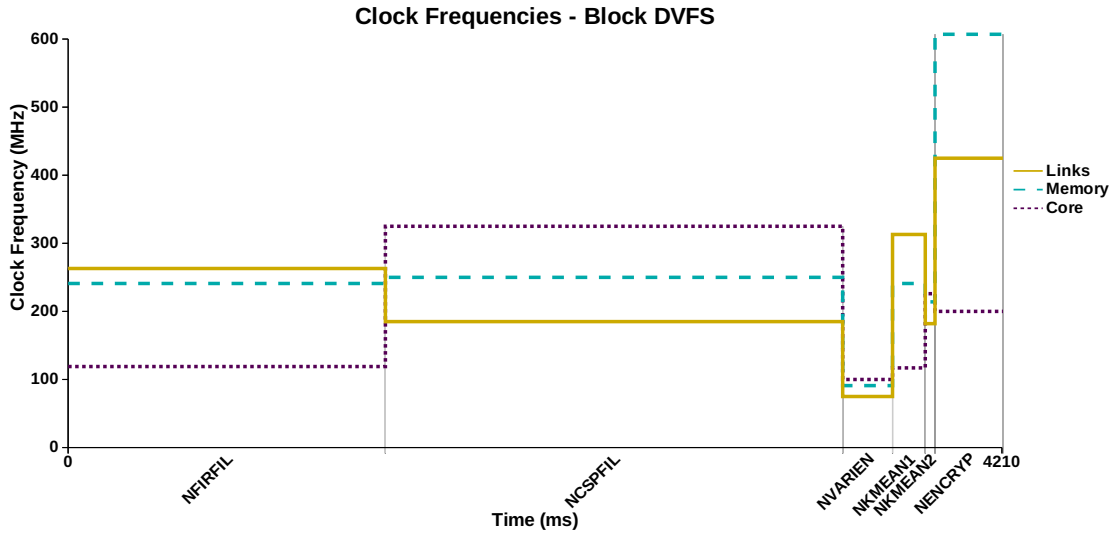


Figure 7.5: Near V_{th} Performance/watt comparison between conventional DVFS and this work (optimized for algorithm DVFS and block DVFS).

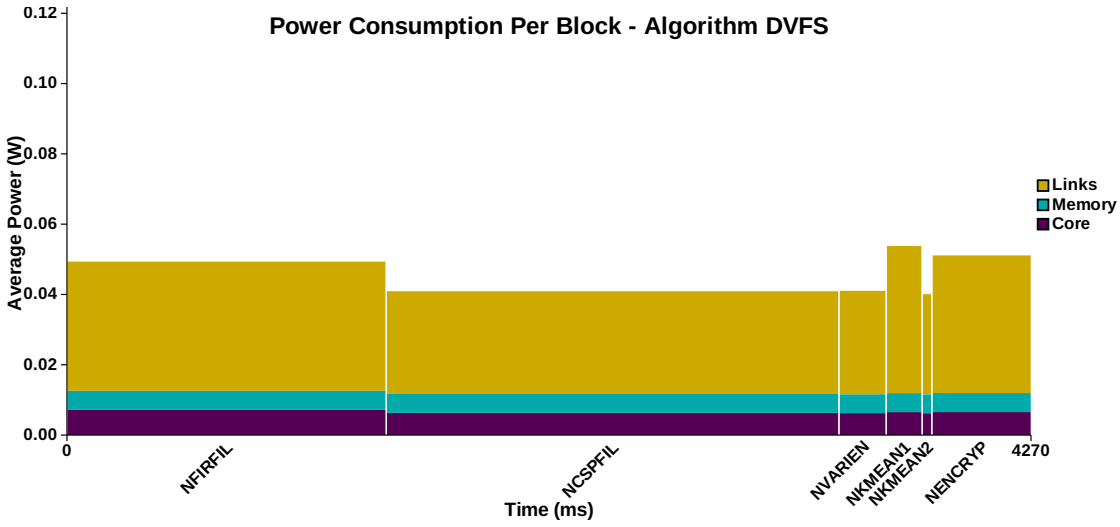


(a)

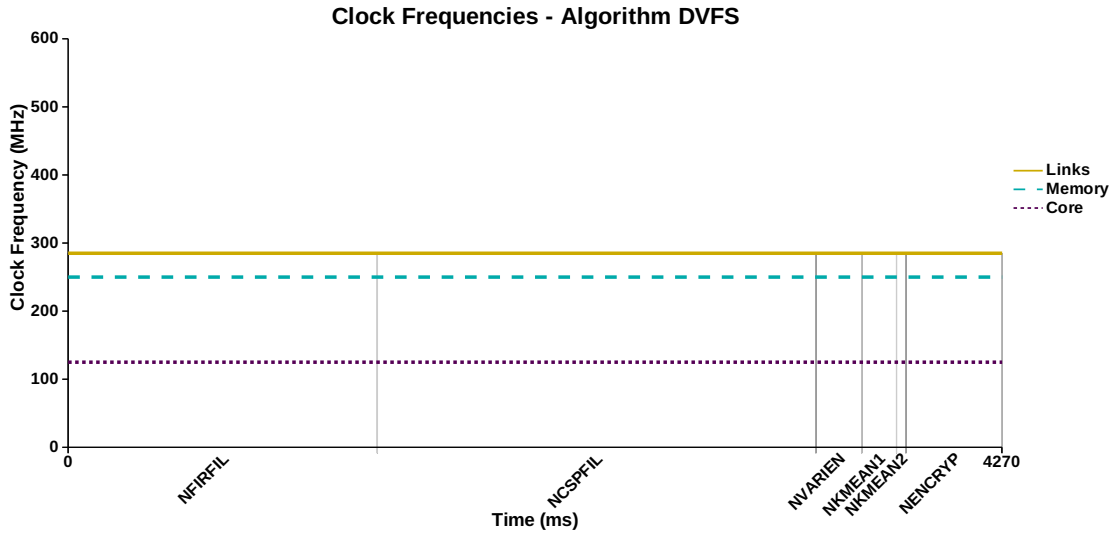


(b)

Figure 7.6: Near V_{th} - Small performance loss optimization results (at 10% performance loss): (a) Block DVFS power consumption, (b) Block DVFS component frequencies



(a)



(b)

Figure 7.7: Near V_{th} - Small performance loss optimization results (at 10% performance loss): (a) Algorithm DVFS power consumption, (b) Algorithm DVFS component frequencies.

7.2.2 Real-Time Constraint Optimization

The time window constrained optimization results can be seen in Figure 7.8. These results once again clearly show that both of our a priori scheduled DVFS optimization methods (block DVFS and algorithm DVFS) greatly decrease average power when compared with the conventional race to sleep method. Since the algorithm takes a minimum of $72ms$ to complete, we observe a much larger power reduction as the time window becomes much greater than $100ms$.

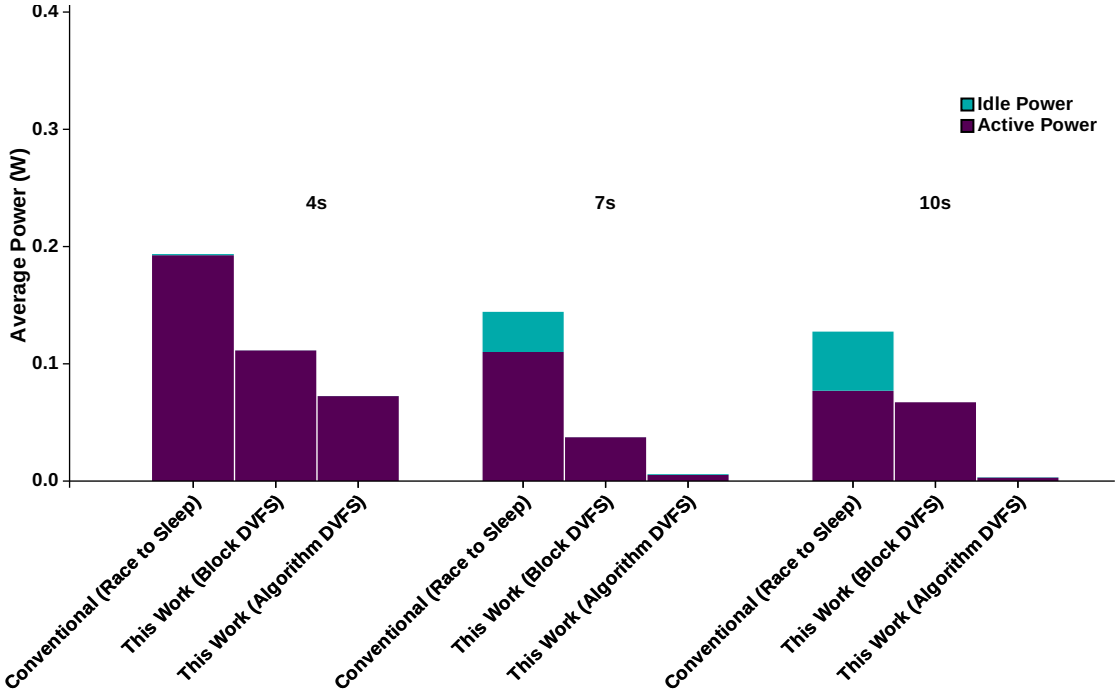


Figure 7.8: Near V_{th} Power comparisons, given a real-time constraint, between conventional race-to-sleep and this work.

Chapter 8 – Conclusion

I have shown that a priori scheduling of DVFS settings in a parallel architecture vastly out performs traditional DVFS methods. Because embedded designs are typically run the same algorithms repetitively, this method can be applied to achieve large performance/watt improvements. Due to the increased prevalence of wide parallel embedded devices, and the technology scaling effects on power, any methodology that is adopted will need to be able to counter these negative effects because while the high performance computing world can be matched with almost unlimited power, the embedded market will remain limited to a few watts. This work has shown that large improvements in performance/watt in both super and near V_{th} architectures, and will be useful in either design spaces as embedded SIMD microprocessors move forward. This work demonstrates a minimum of 50% improvement in performance/watt at 1% performance reduction, whereas a typical DVFS scheme achieved a maximum of 32% improvment at the same point.

This technique, if implemented in a real processor, would make it possible to reduce the battery size of OLAM [2] significantly. Considering that OLAM primarily used the race to sleep method of power reduction, we should get at least 50% improvement in performance/watt.

Bibliography

[1] I. Ahmad, S. Ranka, and S.U. Khan. Using game theory for scheduling tasks on multi-core processors for simultaneous optimization of performance and energy. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1 –6, april 2008.

[2] R.K. Albright, B.J. Goska, T.M. Hagen, M.Y. Chi, G. Cauwenberghs, and P.Y. Chiang. Olam: A wearable, non-contact sensor for continuous heart-rate and activity monitoring. In *Engineering in Medicine and Biology Society,EMBC, 2011 Annual International Conference of the IEEE*, pages 5625 –5628, 30 2011-sept. 3 2011.

[3] K.K. Ang, Z.Y. Chin, H. Zhang, and C. Guan. Filter bank common spatial pattern (fbcsp) in brain-computer interface. In *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pages 2390–2397. IEEE, 2008.

[4] G. Ascia, V. Catania, M. Palesi, and D. Patti. A system-level framework for evaluating area/performance/power trade-offs of vliw-based embedded systems. In *Design Automation Conference, 2005. Proceedings of the ASP-DAC 2005. Asia and South Pacific*, volume 2, pages 940 – 943 Vol. 2, jan. 2005.

[5] M.A. Awan and S.M. Petters. Enhanced race-to-halt: A leakage-aware energy management approach for dynamic priority systems. In *Real-Time Systems (ECRTS), 2011 23rd Euromicro Conference on*, pages 92 –101, july 2011.

[6] A. Bakhoda, G.L. Yuan, W.W.L. Fung, H. Wong, and T.M. Aamodt. Analyzing cuda workloads using a detailed gpu simulator. In *ISPASS 2009*, pages 163–174. IEEE, 2009.

[7] N. Bansal, K. Lahiri, and A. Raghunathan. Automatic power modeling of infrastructure ip for system-on-chip power analysis. In *VLSI Design, 2007. Held jointly with 6th International Conference on Embedded Systems., 20th International Conference on*, pages 513 –520, jan. 2007.

[8] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh

Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, August 2011.

[9] Jo De Boeck. Game-changing opportunities for wireless personal healthcare and lifestyle. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2011 IEEE International*, pages 15–21, feb. 2011.

[10] K. Bowman, J. Tschanz, C. Wilkerson, Shih-Lien Lu, T. Karnik, V. De, and S. Borkar. Circuit techniques for dynamic variation tolerance. In *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE*, pages 4 –7, july 2009.

[11] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *ACM SIGARCH Computer Architecture News*, volume 28, pages 83–94. ACM, 2000.

[12] A. Castagnetti, C. Belleudy, S. Bilavarn, and M. Auguin. Power consumption modeling for dvfs exploitation. In *Digital System Design: Architectures, Methods and Tools (DSD), 2010 13th Euromicro Conference on*, pages 579 –586, sept. 2010.

[13] J. Claassen, SA Mayer, RG Kowalski, RG Emerson, and LJ Hirsch. Detection of electrographic seizures with continuous eeg monitoring in critically ill patients. *Neurology*, 62(10):1743–1748, 2004.

[14] Liang Di, M. Putic, J. Lach, and B.H. Calhoun. Power switch characterization for fine-grained dynamic voltage scaling. In *Computer Design, 2008. ICCD 2008. IEEE International Conference on*, pages 605 –611, oct. 2008.

[15] N. Drego, A. Chandrakasan, and D. Boning. All-digital circuits for measurement of spatial variation in digital circuits. *Solid-State Circuits, IEEE Journal of*, 45(3):640 –651, march 2010.

[16] Loc Duflot, Olivier Levillain, and Benjamin Morin. Acpi: Design principles and concerns. In Liqun Chen, Chris Mitchell, and Andrew Martin, editors, *Trusted Computing*, volume 5471 of *Lecture Notes in Computer Science*, pages 14–28. Springer Berlin / Heidelberg, 2009.

[17] D. Ernst, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, Nam Sung Kim, and K. Flautner. Razor: circuit-level correction of timing errors for low-power operation. *Micro, IEEE*, 24(6):10 –20, nov.-dec. 2004.

[18] S. Feizi, M. Medard, and M. Effros. Compressive sensing over networks. In *Communication, Control, and Computing (Allerton), 2010 48th Annual Allerton Conference on*, pages 1129 –1136, 29 2010-oct. 1 2010.

[19] N. Goswami, R. Shankar, M. Joshi, and Tao Li. Exploring gpgpu workloads: Characterization methodology, analysis and microarchitecture evaluation implications. In *IISWC 2010*, pages 1 –10, dec. 2010.

[20] S. Gupta, Shuguang Feng, A. Ansari, J. Blome, and S. Mahlke. The stagenet fabric for constructing resilient multicore systems. In *Microarchitecture, 2008. MICRO-41. 2008 41st IEEE/ACM International Symposium on*, pages 141 –151, nov. 2008.

[21] S. Hanson, B. Zhai, K. Bernstein, D. Blaauw, A. Bryant, L. Chang, K. K. Das, W. Haensch, E. J. Nowak, and D. M. Sylvester. Ultralow-voltage, minimum-energy cmos. *IBM Journal of Research and Development*, 50(4.5):469 –490, july 2006.

[22] S. Hanson, Bo Zhai, Mingoo Seok, B. Cline, K. Zhou, M. Singhal, M. Minuth, J. Olson, L. Nazhandali, T. Austin, D. Sylvester, and D. Blaauw. Exploring variability and performance in a sub-200-mv processor. *Solid-State Circuits, IEEE Journal of*, 43(4):881 –891, april 2008.

[23] S. Herbert and D. Marculescu. Variation-aware dynamic voltage/frequency scaling. In *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, pages 301 –312, feb. 2009.

[24] Mike Johnson. *Superscalar microprocessor design*. Prentice Hall series in innovative technology. 1991.

[25] A. Kandhalu, Junsung Kim, K. Lakshmanan, and R. Rajkumar. Energy-aware partitioned fixed-priority scheduling for chip multi-processors. In *RTCSA 2011*, volume 1, pages 93 –102, aug. 2011.

[26] Gerry Kane. *MIPS RISC architecture*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[27] Wonyoung Kim, D.M. Brooks, and Gu-Yeon Wei. A fully-integrated 3-level dc/dc converter for nanosecond-scale dvs with fast shunt regulation. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2011 IEEE International*, pages 268 –270, feb. 2011.

[28] David Kirk. Nvidia cuda software and gpu parallel computing architecture. In *Proceedings of the 6th international symposium on Memory management*, ISMM '07, pages 103–104, New York, NY, USA, 2007. ACM.

[29] T. Kolpe, A. Zhai, and S.S. Sapatnekar. Enabling improved power management in multicore processors through clustered dvfs. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, pages 1 –6, march 2011.

[30] R. Krashinsky, C. Batten, M. Hampton, S. Gerding, B. Pharris, J. Casper, and K. Asanovic. The vector-thread architecture. In *Computer Architecture, 2004. Proceedings. 31st Annual International Symposium on*, pages 52 – 63, june 2004.

[31] E. Krimer, R. Pawlowski, M. Erez, and P. Chiang. Synctium: a near-threshold stream processor for energy-constrained parallel applications. *Computer Architecture Letters*, 9(1):21 –24, jan. 2010.

[32] Chien-Wei Kuan and Hung-Chih Lin. Near-independently regulated 5-output single-inductor dc-dc buck converter delivering 1.2w/mm2 in 65nm cmos. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International*, pages 274–275, feb. 2012.

[33] Samuel Larsen and Saman Amarasinghe. Exploiting superword level parallelism with multimedia instruction sets. *SIGPLAN Not.*, 35(5):145–156, May 2000.

[34] Etienne Le Sueur and Gernot Heiser. Slow down or sleep, that is the question. In *Proceedings of the 2011 USENIX conference on USENIX annual technical conference*, USENIXATC'11, pages 16–16, Berkeley, CA, USA, 2011. USENIX Association.

[35] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym. Nvidia tesla: A unified graphics and computing architecture. *Micro, IEEE*, 28(2):39 –55, march-april 2008.

[36] H. Mahmoodi, V. Tirumalashetty, M. Cooke, and K. Roy. Ultra low-power clocking scheme using energy recovery and clock gating. *VLSI Systems, IEEE Transactions on*, 17(1):33 –44, jan. 2009.

[37] Jiayuan Meng and K. Skadron. A reconfigurable simulator for large-scale heterogeneous multicore architectures. In *Performance Analysis of Systems and Software (ISPASS), 2011 IEEE International Symposium on*, pages 119 –120, april 2011.

[38] Micron Inc. *Calculating Memory System Power for DDR3*. Technical Note.

[39] Gordon E. Moore. Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp.114 ff. *Solid-State Circuits Newsletter, IEEE*, 20(3):33 –35, sept. 2006.

[40] Gordon E. Moore. Lithography and the future of moore's law, copyright 1995 ieee. reprinted with permission. proc. spie vol. 2437, pp. 2. *Solid-State Circuits Newsletter, IEEE*, 20(3):37 –42, sept. 2006.

[41] I. Onyuksel and S.H. Hosseini. Amdahl's law: a generalization under processor failures. *Reliability, IEEE Transactions on*, 44(3):455 –462, sep 1995.

[42] D.A. Ortiz and N.G. Santiago. High-level optimization for low power consumption on microprocessor-based systems. In *Circuits and Systems, 2007. MWSCAS 2007. 50th Midwest Symposium on*, pages 1265 –1268, aug. 2007.

[43] A. Pullini, F. Angiolini, S. Murali, D. Atienza, G. De Micheli, and L. Benini. Bringing nocs to 65 nm. *Micro, IEEE*, 27(5):75–85, 2007.

[44] J. Rabaey and D. Markovic. *Low power design essentials*. Springer Verlag, 2009.

[45] B. Ramakrishna Rau and Joseph A Fisher. Instruction-level parallel processing: History, overview, and perspective. In B. R. Rau and J. A. Fisher, editors, *Instruction-Level Parallelism*, volume 235 of *The Kluwer International Series in Engineering and Computer Science*, pages 9–50. Springer US, 1993.

[46] Dave Sager, Desktop Platforms Group, and Intel Corp. The microarchitecture of the pentium 4 processor. *Intel Technology Journal*, 2001.

[47] Sergio Saponara, Luca Fanucci, and Pierangelo Terreni. Architectural-level power optimization of microcontroller cores in embedded systems. *Industrial Electronics, IEEE Transactions on*, 54(1):680 –683, feb. 2007.

[48] R.R. Schaller. Moore's law: past, present and future. *Spectrum, IEEE*, 34(6):52 –59, jun 1997.

[49] H. Singh, K. Agarwal, D. Sylvester, and K.J. Nowka. Enhanced leakage reduction techniques using intermediate strength power gating. *VLSI Systems, IEEE Transactions on*, 15(11):1215 –1224, nov. 2007.

[50] S. Sirisup, S. U-raekolan, and E. Kijsipongse. Multi-level parallelism, global arrays, gpgpu programming: Unify programming paradigms on grid computing with efficiency. In *ECTI-CON 2011*, pages 455 –458, may 2011.

[51] D. Tarjan, S. Thoziyoor, and N.P. Jouppi. Cacti 4.0. *HP Laboratories Palo Alto, Tech. Rep. HPL-2006-86*, 2006.

[52] A. Wang and A. Chandrakasan. A 180-mv subthreshold fft processor using a minimum energy design methodology. *Solid-State Circuits, IEEE Journal of*, 40(1):310 – 319, jan. 2005.

[53] Samuel Williams, Leonid Oliker, Richard Vuduc, John Shalf, Katherine Yelick, and James Demmel. Optimization of sparse matrixvector multiplication on emerging multicore platforms. *Parallel Computing*, 35(3):178 – 194, 2009.

[54] M. Woh, Sangwon Seo, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner. Anysp: Anytime anywhere anyway signal processing. *Micro, IEEE*, 30(1):81 –91, jan.-feb. 2010.