TOWARDS CO-DESIGNED ENERGY EFFICIENT COMPUTING AND RUNTIMES: SIMULATION-FRAMEWORK AND EXPERIMENTS

by

Kelly Livingston

A dissertation submitted to the Faculty of the University of Delaware in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Electrical and Computer Engineering

Fall 2015

© 2015 Kelly Livingston
All Rights Reserved
TOWARDS CO-DESIGNED ENERGY EFFICIENT COMPUTING AND RUNTIMES: SIMULATION-FRAMEWORK AND EXPERIMENTS

by

Kelly Livingston

Approved: 

Kenneth E. Barner, Ph.D.
Chair of the Department of Electrical and Computer Engineering

Approved: 

Babatunde Ogunnaike, Ph.D.
Dean and William L. Friend Chaired Professor

Approved: 

Ann L. Ardis, Ph.D
Interim Vice Provost for Graduate and Professional Education
I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: ___________________________________________________________
        Guang R. Gao, Ph.D.
        Professor in charge of dissertation

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: ___________________________________________________________
        Xiaoming Li, Ph.D.
        Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: ___________________________________________________________
        Stephane Zuckerman, Ph.D.
        Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: ___________________________________________________________
        John Cavazos, Ph.D.
        Member of dissertation committee
I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: ________________________________________________________________
Andres Marquez, Ph.D.
Member of dissertation committee
ACKNOWLEDGEMENTS

Many people have supported me in my time here. Guang Gao has been an excellent advisor through my wandering journey through this dissertation. I also know that without Dr. Stephane Zuckerman, Andres Marquez, and my committee, this dissertation would not be nearly as high in quality. I’d also like to thank many colleagues who helped in producing ideas, graphics, and conversation during my time at CAPSL. Specifically I’d like to thank Dr. Stephane Zuckerman, Jose Monsalve Diaz, and Aaron Landwher for working with me to the bitter end. I can truly say that PhDs are fought and won in the trenches, and you were a great band of brothers. I’d also like to thank Rishi Khan and Jean-Christophe Beyler who were both great bosses, mentors, and friends in my early graduate years with many great technical discussions, some that spurred on much of my academic work. I’d also like to thank my family and friends in accepting my ever pushed back timelines and supporting me along the way. I’d like to thank Zach and Nina Schafer specifically for providing the best writing environment a student could have and a surrogate family when mine was farther away. And, lastly, I’d like to thank Chelsea Tracy, who has patiently encouraged me in these past few years while I made my journey through the PhD program.
TABLE OF CONTENTS

LIST OF TABLES ................................................. x
LIST OF FIGURES ............................................... xi
ABSTRACT ......................................................... xiv

Chapter

1 INTRODUCTION ............................................. 1

1.1 Motivation .............................................. 1
1.2 Contributions .......................................... 2
1.3 Overview ................................................ 3

2 BACKGROUND ............................................... 5

2.1 Landscape of Computing Hardware: A Convergence ......... 5

2.1.1 You Can Have Any Processor You Want as Long as It’s Multi-core .................................................. 5
2.1.2 Accelerators Become General Processors .................. 9
2.1.3 Embedded and Mobile Processors Compete for General Purpose ......................................................... 11
2.1.4 Specializing for Energy ..................................... 12

2.2 Landscape of System Software .............................. 13

2.2.1 The Rise of Virtualization of Everything .................. 13
2.2.2 Map/Reduce and the Era of Big Data ...................... 14

2.3 The Future of Computing Hardware .......................... 15

2.3.1 The Threat of Dark Silicon ............................... 15
2.3.2 The Promise and Challenge of Near Threshold Voltage (NTV) ....................................................... 16
2.3.3 3D Stacked Memory using Through Silicon Vias ........... 17

2.4 Conclusion of the Motivation/Background ................... 17
# PREVIOUS MODELS: A PRIMER ON TRANSISTORS AND NORMAL DISTRIBUTIONS

## 3.1 Transistor Basics

- 3.1.1 EKV Model
- 3.1.2 Subthreshold Model
- 3.1.3 Transregional Model
- 3.1.4 Factors that Affect Threshold Voltage

## 3.2 Statistical Models and Derivations

- 3.2.1 Expected Values and Variance
- 3.2.2 Generating New PDFs using Change of Variables
- 3.2.3 Deriving the Normal or Gaussian Distribution
- 3.2.4 Deriving the LogNormal Distribution

## 3.3 Conclusion

# RUNTIME WITH ENERGY SAVING TECHNOLOGY

## 4.1 Introduction

## 4.2 Phase Detection

## 4.3 Energy Consumption

## 4.4 REST

## 4.5 Experimental Results

## 4.6 Related Work

## 4.7 Conclusions

# VARIANCE MODELING

## 5.1 Background

## 5.2 Random Variance

## 5.3 Systemic Variance

### 5.3.1 Weighting for Spatial Correlation

### 5.3.2 Using Pelgrom to Calculate Weights between Levels

### 5.3.3 Using Pelgrom to Interpolate within a Level

### 5.3.4 Verifying the Variance

## 5.4 Related Work
## 6 TIMING MODEL

### 6.1 Background

- 6.1.1 Alpha Power Law
- 6.1.2 Statistical Static Timing Analysis

### 6.2 Intraprocessor Timing

- 6.2.1 Methodology
- 6.2.2 Transistor Level
- 6.2.3 Gate Level
- 6.2.4 Processor Level
- 6.2.5 Bad Case Scenario

### 6.3 Interprocessor Timing

- 6.3.1 Developing the Model
- 6.3.2 Alternative Transistor Model for NTV
- 6.3.3 Determining $V_{Th}$ and $C_{Eff}$ through Observation
- 6.3.4 Predicting Frequency Capabilities
- 6.3.5 Validation

### 6.4 Conclusion

## 7 LEAKAGE MODEL

### 7.1 Intraprocessor Leakage

- 7.1.1 Developing the Model

### 7.2 Interprocessor Leakage

- 7.2.1 Developing the Model
- 7.2.2 Acquiring the Coefficients through Observation

### 7.3 Validation

### 7.4 Conclusion
LIST OF TABLES

4.1 Experimental Testbed ........................................ 47
8.1 The Duality of Conductions .................................... 91
9.1 TG Strawman Specifications .................................... 96
9.2 Assumed TG Operating Conditions .............................. 98
9.3 Resulting Coefficients for Conditions in Table 9.2 with 10% Variance 100
10.1 Simulation Parameters ........................................ 120
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Lateral Cross Section of a Planar MOS Transistor</td>
<td>19</td>
</tr>
<tr>
<td>3.2</td>
<td>EKV Transistor Model Current Voltage Relationship</td>
<td>22</td>
</tr>
<tr>
<td>3.3</td>
<td>Subthreshold Transistor Model Current Voltage Relationship</td>
<td>24</td>
</tr>
<tr>
<td>3.4</td>
<td>Transregional Transistor Model Current Voltage Relationship</td>
<td>25</td>
</tr>
<tr>
<td>4.1</td>
<td>Energy consumption from the wall and in the CPU on a 3.3 GHz Intel Xeon E3-1240 processor for the Quantum Monte Carlo for Chemistry application, see Table 4.1 for the complete hardware configuration.</td>
<td>35</td>
</tr>
<tr>
<td>4.2</td>
<td>Hardware counters during the execution of a synthetic benchmark: showing how the counters evolve between a memory bound and a compute bound execution</td>
<td>39</td>
</tr>
<tr>
<td>4.3</td>
<td>Hardware counters during the execution of a real world application (Reverse Time Migration)</td>
<td>40</td>
</tr>
<tr>
<td>4.4</td>
<td>Wall energy consumption and time execution for the SPEC program Libquantum depending on frequencies</td>
<td>42</td>
</tr>
<tr>
<td>4.5</td>
<td>Wall energy consumption and time execution for the Reverse Time Migration program depending on frequencies</td>
<td>44</td>
</tr>
<tr>
<td>4.6</td>
<td>Comparison between the naïve and branch-predict decision makers</td>
<td>46</td>
</tr>
<tr>
<td>4.7</td>
<td>REST energy savings and performance degradation on the SPEC 2006 benchmark suite, with the most naïve decision maker</td>
<td>49</td>
</tr>
<tr>
<td>4.8</td>
<td>REST energy savings and performance degradation on the parallel NAS benchmarks using naïve decision maker</td>
<td>50</td>
</tr>
<tr>
<td>5.1</td>
<td>Example Quadtree Grid</td>
<td>57</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>5.2</td>
<td>Intuition for Quadtree Weighting for Proper Spatial Correlation</td>
<td>58</td>
</tr>
<tr>
<td>5.3</td>
<td>A Blocky Quadtree Result</td>
<td>60</td>
</tr>
<tr>
<td>5.4</td>
<td>Spatial Smoothing Algorithm</td>
<td>62</td>
</tr>
<tr>
<td>5.5</td>
<td>Final Approximating Algorithm</td>
<td>63</td>
</tr>
<tr>
<td>5.6</td>
<td>High Resolution Variance Map</td>
<td>63</td>
</tr>
<tr>
<td>5.7</td>
<td>Histogram to Verify WID Variance Portion</td>
<td>64</td>
</tr>
<tr>
<td>5.8</td>
<td>Statistical Analysis of Chip Variance</td>
<td>66</td>
</tr>
<tr>
<td>6.1</td>
<td>Probability of Transistor Speed from Nominal</td>
<td>73</td>
</tr>
<tr>
<td>6.2</td>
<td>Probability of Gate Speed from Nominal</td>
<td>74</td>
</tr>
<tr>
<td>6.3</td>
<td>Probability of Processor Speed from Nominal</td>
<td>76</td>
</tr>
<tr>
<td>6.4</td>
<td>Bad Case Probability of Transistor Speed from Nominal</td>
<td>77</td>
</tr>
<tr>
<td>6.5</td>
<td>Bad Case Probability of Gate Speed from Nominal</td>
<td>77</td>
</tr>
<tr>
<td>6.6</td>
<td>Bad Case Probability of Processor Speed from Nominal</td>
<td>78</td>
</tr>
<tr>
<td>8.1</td>
<td>Example of a 4 Lump Thermal Circuit</td>
<td>92</td>
</tr>
<tr>
<td>8.2</td>
<td>Proposed Method for Thermal Simulation</td>
<td>94</td>
</tr>
<tr>
<td>9.1</td>
<td>Organization of the Traleika Glacier Architecture</td>
<td>97</td>
</tr>
<tr>
<td>9.2</td>
<td>Finding a Reasonable Dynamic Capacitance</td>
<td>98</td>
</tr>
<tr>
<td>9.3</td>
<td>Solutions with Different Variances</td>
<td>99</td>
</tr>
<tr>
<td>9.4</td>
<td>Leakage per Processor as a Function of Temperature</td>
<td>100</td>
</tr>
<tr>
<td>9.5</td>
<td>Energy per Operation for Different Temperatures in a 10% Variance Scenario</td>
<td>101</td>
</tr>
<tr>
<td>10.1</td>
<td>Traleika Glacier Strawman Architecture</td>
<td>105</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>10.2</td>
<td>Graphic Representation of Capacity per Capita Inversion</td>
<td>106</td>
</tr>
<tr>
<td>10.3</td>
<td>Illustration of Different Tiling Products</td>
<td>107</td>
</tr>
<tr>
<td>10.4</td>
<td>Order 1 through 3 Hilbert and Morton Curves</td>
<td>112</td>
</tr>
<tr>
<td>10.5</td>
<td>Example Tiling and Memory Layout</td>
<td>114</td>
</tr>
<tr>
<td>10.6</td>
<td>$S'$: Equivalent Full Tile Aspect Ratio matching Work:Inputs of a Partial Tile</td>
<td>114</td>
</tr>
<tr>
<td>10.7</td>
<td>HIC Curve: an example $50 \times 80$ Hilbert Inspired Curve</td>
<td>115</td>
</tr>
<tr>
<td>10.8</td>
<td>A Portion of the Base Case Scanline Rules Look Up Table</td>
<td>116</td>
</tr>
<tr>
<td>10.9</td>
<td>$S'$ for 3 Different Aspect Ratios</td>
<td>117</td>
</tr>
<tr>
<td>10.10</td>
<td>Memory Accesses</td>
<td>121</td>
</tr>
<tr>
<td>10.11</td>
<td>Relative Energy Consumption</td>
<td>123</td>
</tr>
</tbody>
</table>
As silicon field effect transistor based computing continues into the twenty-first century, there has been a shift in the design of computational hardware systems. Having run solidly into several walls including the power and memory walls, computer architects are now taking different tactics to translate Moore’s law into more performance. Whether by using more leakage-aware transistor technology, providing multiple ISA equivalent microarchitectures to match variable computing demands, or including fixed function accelerators, engineers continue to push the limits of physics and mass production techniques to deliver better computing systems. However, this dissertation will provide evidence for an equally important need to shift system software and its organization to both leverage the new hardware capabilities and organization. To that end, the proposal will present 3 case studies: the current ability for system software to manage a processor’s power envelope through hardware mechanisms using a tool called REST as a prototype, multiple novel chip level approximations required to model heterogeneous near threshold many-core systems of the future, and lastly, an initial runtime driven algorithm that performs dynamic tiling and scheduling based on a self-aware runtime. All these pieces combined will guide architects and system software developers in creating a new system of computation that can meet the energy efficiency goals of an exascale machine.
Chapter 1
INTRODUCTION

1.1 Motivation

This dissertation seeks to aid in the design of both system software and hardware simulation for a single purpose: control and efficient trade-off of energy and performance in modern computing systems and their software algorithms. This is a necessary capability both for economic and physical reasons.

We have reached thermal and power limits of circuit design in the past. In the 1960’s, as bipolar junction transistors (BJT) dominated circuit design, a similar increase in power density and energy usage threatened to stunt the growth of computing. Similar to the subthreshold leakage that transistor technology experiences today, BJT technology had a constant and large leakage due to the constant base current. The advent of field effect transistors, first predominantly NMOS or PMOS and then Complimentary MOS or CMOS, allowed significant reductions in power consumption. In fact the first MOS transistor based microprocessor was the MP944 which calculated flight surfaces and pilot displays for the F-14 Tomcat [56]. In the past twenty years, we have seen similar advancements in CMOS technology to reduce leakage using better materials like high K dielectrics for gate insulation as well as better geometry like the fins in three dimensional transistors known as finFETs which are less susceptible to doping variance and have better short channel effects for the leakage and drive current trade-off of a transistor. Unfortunately, these design parameters cannot be scaled like process technologies and are considered a one time improvement in energy efficiency. And as lithography pitches shrink to approach the silicon lattice spacing of 5.4 Angstroms, the opportunities to increase energy efficiency through better transistor
design become severely limited. Aside from using some other transistor technology or quantum or biological computing which could free us of binary calculations, there will be a minimal discharge of thermodynamic entropy to guarantee the Shannon entropy within the state machine of the deterministic logic of a processor, although we are nowhere near the theoretical limit as is stated by the Landauer principle.[78]. Energy efficient computing improvements now fall to the purvue of architectural design and better system software. The goal of this dissertation and its experiments is to give hope that these improvements are possible.

1.2 Contributions

This dissertation seeks to provide a framework for developing system software and hardware with the capabilities to reach exascale which will inevitably require Near Threshold Voltage (NTV) technology to meet the energy efficiency requirements. Such system software will need to efficiently interface with an execution model, schedule tasks optimally, and operate this complex NTV hardware in the most energy efficient manner possible. The operation of the underlying hardware and its behavior is not well simulated for high level functional simulators that exascale system software would be initially built on. Thus, it is not the goal of the dissertation to produce the system software, but instead provide a physically realistic platform that will emulate the behaviors that the system software must control and optimize. Furthermore, we utilize a rudimentary runtime within a newly formed Matrix Multiply algorithm as an example of the potential gains a NTV-aware exascale runtime system could bring to future systems. The following contributions will provide a solid foundation to guide future researchers in exploring the space of designing this software:

1. Experiments using a (developed) prototype runtime built on top of current operating systems and execution models to demonstrate the current state of energy efficient operation

2. A fast and high-performance temperature simulation method which is adequate for system software development
3. A general purpose variance model for Near Threshold Voltage processors which is adequate for system software development

4. An energy and timing model based on subthreshold leakage regression of observations and low-level transistor models which allows for:

   (a) A model that will also provide a lower bound for per instruction energy efficiency given an infinite number of frequency domains

   (b) A model that will provide a methodology to determine optimal frequencies when only a discrete number of domains are allowed

5. An initial algorithm experiment with a prototype execution model on a many-core architecture to demonstrate the necessity of dynamic load balancing within an execution model

1.3 Overview

In covering these contributions, this dissertation will first give a motivation and background of energy efficient computing in both software and hardware aspects in Chapter 2. In Chapter 3 the dissertation will provide a brief primer on transistor models and statistical models on which the novel contributions are built. Next, the dissertation will provide a baseline study for energy efficiency on current commercial platforms and operating systems today in Chapter 4 and give an intuition for how much we can improve system software without any large software overhauls. After our initial study, the dissertation will provide novel contributions in four major areas of simulation for NTV. Chapter 5 will provide a novel approximation for expressing variance between processor chips using a physically derived model and statistical methodology. Chapter 6 will provide a novel timing model built by modifying the alpha power model\cite{110, 26} to help estimate potential frequency capabilities of each processor in a many-core design. The timing model also attempts an approximation to measure the impact NTV operation will have on timing using a technique called Statistical Static Timing Analysis. Chapter 7 likewise creates an intraprocessor and interprocessor energy model improving on \cite{84, 59}. Lastly a temperature model that avoids a system wide synchronization is presented in Chapter 8 to complete all the necessary tools for making a full analysis.
of an NTV based architecture. Having introduced many novel approximations, the dissertation makes an analysis in Chapter 9 of a potential architecture and makes several recommendations for the purpose of improving energy efficient operation. Lastly, the dissertation presents a novel algorithm in Chapter 10 to provide a methodology forward for solving the data movement aspects of energy efficient computing. The dissertation concludes in Chapter 11 with other potential lines of research that could follow the findings presented in this dissertation.
Chapter 2

BACKGROUND

2.1 Landscape of Computing Hardware: A Convergence

An interesting trend has emerged in computing in the twenty-first century. Macroscopic forces in the world economy and microscopic quantum forces within our silicon are both driving hardware designers to create smaller devices and converge the different types of computing processors into becoming very similar designs. While embedded processing has often focused on energy efficiency due to limited power supply in its applications, economics has driven them to become more general purpose in nature to provide more marketability for a device and thus a lower price point due to large fixed costs in research and development. On the opposite side, high performance processors, already designed to provide large amounts of computation for a variety of applications, find that the physics at play inside a processing chip require them to be as energy efficient as possible if the products are to continue to increase performance. This landscape for hardware makes it clear that energy efficient design is now more necessary and more applicable across fields than ever before.

2.1.1 You Can Have Any Processor You Want as Long as It’s Multi-core

Since the mid-aughts, industry has abandoned larger and larger superscalar cores in favor of multi-core architectures. This was not by the choice of any company as the continued gains of uniprocessor process shrinking was a profitable, fairly simple, and low risk path forward from the 80’s into the new millennium. Intel perfected this technique in a method they called “tick tock” [63]. As computer architects would need two to three years developing the next architectural features such as more sophisticated prefetchers, branch predictors, and ISA vector extensions that could utilize more
transistors to provide faster sequential performance, physicists and materials scientist would be working the two to three years in opposite phase to bring the next lithography process online. Thus, approximately every nine months to one and half years there would be a new chip ready for market that would have faster sequential performance. This methodology, along with media confusion, led to a mistaken understanding of Moore’s Law. Moore’s Law is an observation that physicists and engineers have been able to reduce the lithography etching fidelity or pitch every eighteen months which results in having more circuit devices in the same area of silicon [95]. General rule of thumb for Moore’s Law says that transistor density will double every eighteen months. However, because of the tick tock strategy and the continued deepening of pipelines and other architectural features that maintained or increased clock rates, Moore’s Law from the public point of view was defined as processor speed doubling every eighteen months. For a period of time, megahertz and then gigahertz was the only metric consumers thought they needed to know in order to understand the performance of their processor [35]. The reality is that since the mid-aughts, doubling of transistors, computational power, or clockrate, has failed to be true [46]. Furthermore, observations like May’s law or similarly Wirth’s law [128] state that “Software efficiency halves every 18 months, compensating Moore’s Law” have unfortunately been accurate observations in the last fifteen years. Without changing, the industry was on a trajectory where hardware and software were to stagnate in performance to the user.

Industrial chip designers were forced away from uniprocessor improvement. If they could economically sell the consumer a faster uniprocessor, they would have. But now whether a consumer wants a multi-core or not, this is the choice everyone must embrace for the future. These design choices were driven by the three walls: the power wall [76], the memory wall [130], and the Instruction Level Parallelism (ILP) wall [14] [36] [17].

The power wall could be renamed as the thermal wall. That is to say the main impediment to increasing power density necessary to place more computation on a chip is the physical ability to economically remove heat. The standard method to
remove heat is to place a metal heat sink, typically aluminum or copper, over the silicon chip to conduct the heat away. These heat sinks are then cooled either by air or water through convective forces. However, even the micrometer gap caused by the interface between materials causes a conductive thermal resistance that can only be overcome with a fairly large temperature gradient between silicon and metal. In an attempt to alleviate this, thermal pastes with better thermal conducting properties are employed to fill this gap, but even these pastes cannot overcome this physical limitation. It is worth mentioning that other cooling methods exist such as submersion in mineral oil or the more exotic fluids used in famous systems such as the Cray-2 like tetradecafluorohexane and other fluorocarbons branded by 3M as Fluorinert which are electrically non-conductive and non-corrosive to metal and thus would not damage circuit boards or processors. However, these methods add large complexities and capital expenses so are not commercially employed. For standard air cooling, there will be a maximum delta in temperature and thus a maximum power density in order to insure the circuits operate below a certain temperature. Higher temperatures where transistor leakage is higher, wire resistance is higher, and switching frequencies are slower as driving currents go down can lead to thermal runaway where the inefficiencies of the circuit drive the temperature higher which then exacerbate circuit inefficiency until permanent processor failure. Hitting the power wall is like driving into a brick wall; the passenger is lucky to survive.

Before discussing the next wall, there is an interesting trend in commercial data-centers that places even more consideration on the power wall. While cooler circuits provide for more energy efficient processing, cooling can be a large opposing force to energy efficiency in terms of total energy consumption. Total energy consumption is also affected by the efficiencies of vapor compression cooling systems which operated much more efficiently at higher temperatures. Additional cooling equipment is not even required if ambient air temperatures are acceptable to cool a datacenter. Since data centers and computing systems in general look at total computation per total energy consumed, the energy consumed by cooling over computer systems has led to
the standard of efficiency called Power Usage Effectiveness or PUE [9]. This metric
determines how much of the energy actually goes into computing compared to the cool-
ing and power supply infrastructure of a data center. Many centers have now looked
at hot water or air cooling where special microchannel heat sinks can convect heat
away for a very low delta in temperature. This also leads to circuits operating at high
junction temperatures, but the energy savings in not maintaining cool temperature
control outweigh the losses in energy efficiency by having a hot processor. However,
running processors hotter also makes for unreliable devices, necessitating hardware and
software redundancy to insure proper computation.

The next wall is the physical limits of memory and their effect on processor
speed. The first aspect related to memory is the bandwidth limit as a consequence of
both physical pin count out of the chip and the signaling speed of each pin. Regardless
of the architecture, this bandwidth limit plays a critical role in how much computation
can be accomplished for a given algorithm. If the algorithm is able to take advan-
tage of on-chip caches, it is possible to avoid this bandwidth limitation. The ratio of
an algorithm’s amount of local computation to remote memory operations is called
computational intensity. Some algorithms can use techniques like tiling or other forms
of scheduling to increase computational intensity; however, many algorithms simply
cannot be designed to have a high computational intensity and suffer from this band-
width limitation. The second and more traditionally defined aspect of the memory
wall is the role that latency to memory plays on sequential processors. The traditional
way an out of order (OoO) processor maintains performance is to issue loads to be
retrieved from memory and find other instructions to perform that do not depend on
that load until it returns back from main memory. The time to make the round trip
from memory is limited at minimum by the speed of light, meaning that an increase
in processor frequency will proportionally increase the latency to memory [130]. In or-
der to tolerate these latencies without losing performance, additional hardware which
scales nonlinearly in both power and area are required to keep more memory operations
in flight from the same instruction stream at the same time. Thus, in trying to break
the memory wall, we run into the final obstacle.

As mentioned, the hardware in a modern sequential processor is anything but sequential. There are many architectural features that are employed to take the list of instructions in a program and exploit the parallelism available. But no matter what method is used, there is a limit to the number of instructions that do not depend on each other in a given program. Some of the limitations that reduce the parallelism are instruction window size, branch prediction, and register size to name a few. This means that for every architectural feature employed, there will be lower marginal return in performance and higher requirements in energy, die area, and system design cost [14][36].

These walls have forced the hand of chip designers to shift to symmetric multi-threading and then multi-core to speed up programs through Thread Level Parallelism (TLP) rather than ILP, to innovate how temperature and active processors can be managed on a chip, and to maintain and in some cases lower the clock frequencies of their processors.

2.1.2 **Accelerators Become General Processors**

In the field of graphics computing, processor cores have evolved to attempt to overcome the 3 walls in unique ways. And while Graphics Processing Units (GPU) are not entirely architected like a CPU, the trend of GPUs is to become more like CPU at every iteration. Originally GPUs had a very simple pipeline that was designed to accelerate rasterizing triangles on a screen. As Moore’s Law progressed, the GPU began to take on more flexible computational roles in computer graphics. The first trend was to make more fixed function capabilities, for example transformations, clipping, and lighting (T&L), available in hardware so that the CPU could offload these calculations. Since these calculations must be performed for each frame, dedicating logic for this purpose provided coarse grained pipeline parallelism and could allow for more complex geometries. The first T&L hardware unit was available in 1999 in the GeForce 256 graphics card by NVidia [1]. As video games demanded more complex visualization
as well as geometry, the units for calculating the shading of pixels needed to be more flexible and thus general. Likewise, the processing of vertices to do bump mapping \cite{24} and other realistic shading techniques required more general calculations in the vertex portion of the graphics pipeline. It was this reversal of hardware strategy to more general purpose that led researchers to consider the GPU a more general purpose streaming processor \cite{89}. Hijacking textures as a method to store results and create a form of stream-based computing that leveraged all the parallelism of the GPU to do general computations, projects like BrookGPU were able to demonstrate that GPUs could outperform CPUs for many scientific applications that had large amounts of data-independent computation \cite{28}. The final transition from GPU to general purpose GPU (GPGPU) came with the evolution of the cache hierarchy to allow for some level of coordinated sharing. Prior to this evolution, textures and vertex data were the only cached data that were fairly small and had high hit rates. This meant that caches and the data they stored were not shared across all the chip’s processors and needed very little if any coherence since they were considered read-only. This also made the programming model and coherence management very difficult to write even simple programs for GPGPU. The creation of frameworks like CUDA \cite{111} by nVidia made this much more simple by creating constructs to imply the level of coherence needed and generally simplify programming for an algorithm.

As the GPU transformed into GPGPU, many of the features of a CPU such as ECC, hierarchical caches, and IEEE double precision floating point in hardware, made their way into subsequent designs. An interesting observation with GPUs was their original design focused on streaming data from memory very fast since operations worked on datasets much larger than cache. In order to increase memory bandwidth, a specialized memory technology requiring more specialized connections to the board and reducing the capacity of memory to the chip was used called GDDR \cite{68}. Even with the added bandwidth of GDDR, high amounts of compute capability could make the GPU very often bandwidth bound. Simple algorithms such as Matrix Multiply, which will be discussed later, could become bandwidth bound due to the small cache sizes
in the first GPUs design. This is especially true as more algorithms and acceleration schemes were ported to the GPU that had locality of reference. Depending on this locality and aspects of the computation, GPUs could range from 50% to 1500% or more of an equivalent CPU [81].

GPGPU has also extended into the embedded space such as the automobile industry where vehicles are using advanced cameras, sonar, and/or radar sensors to detect slower vehicles ahead, to parallel park, or simply power the graphics operations of the touchpad dashboards. All these advanced features require large amounts of computing with minimal power consumption and in high temperature environments which would usually have been in the scope of embedded systems. Yet as more processing is required and more computing systems need to work in parallel, the economic advantage of having a single GPGPU perform these is more economical and well suited to the GPU architecture [12].

2.1.3 Embedded and Mobile Processors Compete for General Purpose

Originally competing in the embedded and mobile space, ARM, which stands for Acorn or Advanced RISC Machine, was a simple processor design created in the UK in the 1980s by the Acorn computing company. It was founded on the principle of Reduced Instruction Set Computing (RISC), which has the goal of simplifying fetch and decode of instructions by having a small number of fixed length instructions which typically operate directly on a sizable register file and using direct load/store operations [103]. This allows for great reduction in transistor count with the first design using less than 25,000 transistors which was 14% less than the 8086 from Intel, and because of the simplified instructions and static registers, compilers are able to directly expose parallelism in an instruction stream without any needed hardware. Doing this gave a sizable reduction in energy usage which specifically allowed ARM to be the chosen processor in the Newton Personal Digital Assistant (PDA) designed by Apple and sold in 1993, a harbinger of what was to come in the mobile industry.
Interestingly, ARM chose a different business model from other companies and licensed their microarchitecture designs to any manufacturer that needed these compute capabilities. They even went as far as to license a synthesizable architecture such that the design could be ported to any chip technology using automated circuit design tools. This had the effect of spreading microarchitecture and compiler design costs over a very large number of manufacturers while they could still add additional custom circuits to their chips and maintain proprietary control over the design. As ARM gathered more customers by moving into the newly created mobile space, some needed more performance and so a large assortment of microarchitectures that were ISA compatible were designed for cell phones, tablets, and other devices. The diverse performance energy tradeoffs of ARM based devices interested the HPC community to the point that now there are projects out of Barcelona Supercomputing Center looking to specifically use mobile-based processors and GPUs in a supercomputer simply due to their energy efficiency [107].

2.1.4 Specializing for Energy

Of particular interest to this dissertation is the beginning of energy efficient specialization and scheduling. In the case of GPUs, nVidia developed their Tegra line of multicore mobile processors which included a companion core. This companion core is identical in microarchitecture to the other cores; however, it is created using Low Power transistors thus making it more energy efficient but lower in performance [10]. In order to harness this energy efficiency, nVidia provided specialty drivers that would disable and migrate work from or to the companion core as time slices were consumed. A different approach was taken by ARM, whereby the transistors were all the same but the microarchitecture was varied. For example, the quadcore Cortex-A7, an in-order 8 stage pipelined processor that is highly energy efficient, and the quadcore Cortex-A17, an out-of-order 11 stage pipelined processor with reasonable energy efficiency and high performance, can be paired together to form what ARM calls a big.LITTLE processor group [11]. There are several scheduling schemes but the initial design was
to have these clusters of cores be virtualized into a single cluster. This would mean the operating system would see a 4 core processing group and when it asked for a lower performance via cpufreq (later chapters will explain) from a particular core, the underlying device drivers and hardware would transition the work seamlessly from the energy inefficient high performance core to the lower performance energy efficient core and back as requested. Later schedulers placed in the Linux 3.1 kernel as a patchset [87] were even able to view all 8 cores and assign specific background tasks to very efficient cores since they did not impact the user experience and would extend the battery life of the device. This is a great start to developing schedulers that are more aware of heterogeneous systems, but still only the beginning.

2.2 Landscape of System Software

2.2.1 The Rise of Virtualization of Everything

A common practice in all computing is to utilize the most valuable resource with the best efficiency even if less valuable resources may be underutilized. To this point, original programming of processors was vastly different than modern day programming. With small amounts of memory, no caches, and limited speed of computation, programmers were responsible not only for loading their data into physical memory but also their program code into memory and manually swap blocks, also called overlays, of the program. All this effort was made by the programmer because the computer and memory was expensive and the programmer time was cheap comparatively. As computers became more powerful and memory density increased, the capital acquisition of a system became the largest expense, and so time sharing software was developed so that pauses between user commands and file system requests wouldn’t stall the machine. It instead could be shared between multiple different users without there being any appearance of sharing. First examples like the Dartmouth Time Sharing System [3] in 1962 even had basic commands to query the state of the machine to help the user make better use of programs in the memory. Thus many system software features were developed to aid in this including virtual memory, processes, and blocking system
calls to name a few. While mainframes would fall by the wayside as Beowolf clusters from commodity hardware would dominate the landscape, virtualization was just beginning.

2.2.2 Map/Reduce and the Era of Big Data

With the rise of the Internet, web servers and databases increased in count very quickly. At first, every business or institution had their own webserver. It was realized that having an IT department, system administrator, and hardware that was always powered on to serve several hundred webpages a day was highly inefficient. There needed to be a way for a person to have their operating system environment and libraries that the webpage or database was built on to use hardware that also could run another customer’s job. Thus, the hypervisor and virtual machine were created to become the operating system of the different operating systems and allow for additional virtualization. While this reduced overall costs, the placing of many jobs on a single system reduced redundant hardware which lower energy consumption, but the sharing of resources between opaque operating systems impedes energy aware scheduling.

Having been reduced significantly, hardware that has been dispatched meant that one final cost remained in the commercial space: the programmer. Many abstractions like MPI [48] had already been developed to reduce programmer effort to decompose a problem, map the decomposition to a cluster of processors, and manage communication of the processors. For Google, a large company with thousands of servers running their web searching services, a programming framework that also managed reading vasts amount of data from files and also automated redundancy of computation in case a commodity processor failed was necessary to ensure a whole job would complete in a reasonable time. They named this programming model MapReduce due to the paradigm of having 2 phases of the model, one to map keys to values and another to reduce the keys to a single dataset the user wished to view [38]. This gave Google the ability to reliably process thousands of requests quickly but at the
expense of large amounts of disk access and CPU under-utilization. Subsequent implementations of MapReduce like Hadoop have perpetuated energy inefficient scheduling in the commercial realm with a renewed interesting in increasing energy efficiency [129].

With all these layers of virtualization in place, it will be difficult as a runtime scheduler to optimize for energy efficiency or performance, although some methods have been attempted [132] [124].

2.3 The Future of Computing Hardware

2.3.1 The Threat of Dark Silicon

As mentioned previously, the power wall dictates that the power density of the chip must remain the same for conventionally cooled chips. During the late 20th century, this was achieved due to Dennard scaling [41], which stated that the scaling down of devices had the added advantage of lowered dynamic capacitances which allowed for faster switching frequencies if drive currents remained the same. In order to maintain these drive currents, the lithographic process was altered to lower the threshold voltage and supply voltage proportionally. Since dynamic power consumption is defined as:

\[ P_{\text{Dyn}} = C_{\text{Dyn}} \times F \times V^2 \]  \hspace{1cm} (2.1)

where \( F \) is frequency, \( C \) is the dynamic capacitance, and \( V \) is voltage, so as the device was scaled down by \( S \) in one dimension, usually by a factor of seven tenths, the area was reduced by \( S^2 \), or typically one half, and the dynamic power was reduced to

\[ \left( \frac{F}{S} \right) (CS)(VS)^2 = FCV^2 \times S^2 \]  \hspace{1cm} (2.2)

which meant the power density defined by

\[ \frac{\text{power}}{\text{area}} = \frac{FCV^2 \times S^2}{S^2} = FCV^2 \]  \hspace{1cm} (2.3)

remained constant through subsequent scalings. At larger lithography processes, this was true because leakage and quantum effects had a much smaller impact compared to the capacitance discharged from transistor switching. However, since the mid-aughts,
the properties of scaling as Dennard described have stopped as we approach the minimum threshold voltage to control leakage which means $V$ cannot be reduced and thus the dynamic power equation is now

$$\left(\frac{F}{S}\right)(CS)V^2 = FCV^2$$

and thus the power density is

$$\frac{FCV^2}{S^2}$$

In a post-Dennard era, an increase in power density necessitates either more energy efficient architectures or the dilution of energy inefficient cores with lower power density devices such as caches filling the new area provided by the process shrink. This has led many in the industry and academia to claim that the future will have dark silicon, areas of a chip that cannot afford to be powered because of the heat output it would create [122].

### 2.3.2 The Promise and Challenge of Near Threshold Voltage (NTV)

As stated prior, the voltage of a circuit determines both energy consumption and frequency. In the past, to maintain driving currents the voltage would operate in what is called the superthreshold region of a transistor. This operating region of a transistor had nice properties, such as similar drive currents even with process variation due to being in the saturation region. However, if we lower the operating voltage closer to the threshold voltage, a theoretical voltage where a transistor makes the transition from the “ON” state to the “OFF” state, we see a linear to squared effect in frequency in performance, but a fourth order reduction in dynamic power and reduction in static power due to lessened short channel effects. This means that energy per instruction can be greatly reduced as the dissertation will explain in later chapters, but at the expense of much lower performance. If lower performance is acceptable for small periods of the program, for example during network traffic or in certain small phases of a program, then this can be a good solution for the dark silicon problem.
However, the downside to operating near the threshold is that every processor has a slightly different “ON” voltage. Due to the steep subthreshold slope prior to the threshold voltage, the maximum frequencies between multiple processors can be drastically different and the energy efficiency can vary greatly as well. However, if these variances can be tolerated, the power wall can be managed in the future [115].

2.3.3 3D Stacked Memory using Through Silicon Vias

Overcoming the memory wall may also be possible in future computing systems. First, latencies will naturally be less in an NTV design because frequencies will be lower. To rephrase this previous statement, while the round trip time to memory cannot be shortened due to physics, the number of cycles that the architecture must be able to hide the outstanding load will be lower due to the lowered frequency of the processor. Second, recent developments of 3D stacked memory would allow very high bandwidth to memory, by removing the pin limits and signaling speed limits that wire bonds have by placing memory on the top of the chip that could make hundreds or thousands of electrical connections with Through Silicon Vias (TSV). This would place gigabytes of memory within only hundreds of cycles of processor logic at a low cost and with high bandwidth connections [114]. However, challenges still exist with TSV based designs. Going back to the power wall, placing material between the processor and the heat sink makes additional problems for cooling. Also, it’s still unclear the economics of this design compared to traditional methods.

2.4 Conclusion of the Motivation/Background

As this chapter has explained, there is a great challenge if the scientific community is to reach exascale. With NTV being a potential solution, the problem of large variation between different processors will need to be simulated and solved. Additionally, the added efficiencies and security of virtualization will need to be reconciled with the lose of energy efficient scheduling in some kind of runtime or OS. The coming chapters will continue to show problems with controlling hardware, scheduling software, and
finally provide an overall hardware simulation framework with intuitions for enhancing a scheduler as well as an initial algorithm that has been built from the ground up for energy efficient runtime prototypes.
Chapter 3

PREVIOUS MODELS: A PRIMER ON TRANSISTORS AND NORMAL DISTRIBUTIONS

Before diving into the novel work, it is helpful to have a quick overview of previous models utilized by the industry and academia that will be used as base models to build all novel approximation models in future chapters. There are five models we will explore: three transistor models and the normal distribution derivations with changes of variables in two applications, each with its own set of useful transistor operating regions and mathematical usefulness for the purposes of this dissertation.

3.1 Transistor Basics

Figure 3.1: Lateral Cross Section of a Planar MOS Transistor

---

1 Graphic taken from https://commons.wikimedia.org/wiki/File:Lateral_mosfet.svg with GFDL License
A field effect transistor as pictured in Figure 3.1 is a nonlinear electrical device with three terminals (with untied bulk terminal it can be four but for our explanations only three are necessary). The three terminals are the gate, drain, and source with an electric insulator between the gate terminal and the semiconductor beneath it with the drain and source terminals on either end. The electric insulator was traditionally a silicon oxide, thus creating the MOS namesake, but is now typically silicon oxynitride or some other hi-K dielectric material. Similarly, the metals of a MOSFET can be aluminum, copper, gold, or polysilicon depending on the desired speed and cost of the circuit. No matter the materials, the field effect principle is the same: if the gate is delivered a charge of electrons by a voltage past a certain threshold, the electric field created by the charge of electrons exerts a force that creates an inversion layer within the doped semiconductor channel below and by having this depleted channel between the drain and source terminals, hole/electron recombination, through the action of thermal diffusion, will generate a current. As the voltage across the drain and source increases, the channel will eventually reduce and pinch due to electron velocity saturation and this mode of operation is called saturation. However, before this state, the transistor behaves much like a resistor in that an increase in voltage between the drain and source causes a linear increase in current, thus the mode of operation is the linear mode or sometimes called the trimode. Both of these modes require the gate voltage to generate the inversion layer, otherwise the transistor is in weak inversion or the off-state. While these explanations may seem a bit detailed for creating system software frameworks, the extrapolations using each section’s transistor model for the modes of operation will have a profound effect on NTV simulations.

3.1.1 EKV Model

The first model we will address is the most physically accurate as compared to the explanation above and has some very nice properties. The EKV model can be utilized in any state of inversion and any mode of operation making it a continuous model. It has a fairly simple equation composed of transcendental functions. But it
can’t be easily used in normal distribution integration as future chapters will show. The EKV model of a transistor was developed by Enz, Krummenacher and Vittoz in the mid 90s based on work from the 80s [45]. It uses a charge sheet approximation of the geometry assuming perfectly uniform doping and leverages an interpolation function from [99] to model the influences from gate charge inversion. It can be used for analog circuits, RF design, and digital design as well. Having described the model, the equations for the EKV model are:

\[ V_P = V_{\text{Gate}} - V_{\text{th}} \]

\[ I_{\text{Spec}} = \frac{2\mu W_{\text{eff}}}{L_{\text{eff}}} C_{\text{ox}} m \ast V_T^2 \]

\[ IC = \ln^2(e^{\frac{V_{P-S}}{2V_T}} + 1) \]

\[ i(x) = I_{\text{Spec}} \ast IC \]

\[ i_f = i(V_S) \]

\[ i_r = i(V_D) \]

\[ I_{D-S} = i_f - i_r \]

where,

\[ m = 1 + \frac{C_{\text{dm}}}{C_{\text{ox}}} = 1 + \frac{\xi_w}{\xi_{ox}} \frac{W_{\text{dm}}}{t_{ox}} = 1 + \frac{3t_{ox}}{W_{\text{dm}}} \]

and

\[ V_T = \frac{KT}{q} \]

\[ V_P \], is the pinching voltage defined as the gate voltage minus the threshold voltage (discussed in Chapter 5). The specific current, \( I_{\text{Spec}} \), is the current at the beginning of inversion and is composed of \( \mu \), the electron mobility; \( W_{\text{eff}} \), the effective channel width; \( L_{\text{eff}} \), the effective channel length; \( C_{\text{ox}} \), the capacitance formed by the oxide layer of the gate; and \( V_T \), the thermal voltage defined by Equation 3.3 and is a function of Temperature \( T \) in Kelvin scaled by Boltzman’s constant, \( K \), divided by \( q \), the charge on an electron. Additionally, equation 3.2 defines the Subthreshold
Slope coefficient, $m$, in several ways as a function of $C'_{dm}$, cumulative diffusion-to-body (substrate) capacitance; $\xi_{si}$, the permittivity of silicon; $\xi_{ox}$, the permittivity of the oxide; and $t_{ox}$, the gate insulator oxide thickness. Lastly, there is an inversion coefficient, $IC$ which interpolates the specific current from weak inversion to strong inversion which is very important for doing near threshold analysis since that region is between the two. Using $IC$ and $I_{Spec}$, we can describe the forward current as the channel based on the pinching voltage and the source voltage while the reverse current is the channel bias based on the pinching voltage and the drain voltage. Thus, the final voltage-current relationship for the current from drain to source, $I_{D-S}$, is a function of the forward current subtracted by the reverse current for a given set of gate, source, and drain voltages.

![Figure 3.2: EKV Transistor Model Current Voltage Relationship](image)

**Figure 3.2:** EKV Transistor Model Current Voltage Relationship
3.1.2 Subthreshold Model

The next model describes the Voltage to Current relationship for a transistor but specifically only for when the gate voltage is below its threshold voltage and in weak inversion. This model is called the subthreshold model and is defined as:

\[
I_{D-S} = \frac{\mu W_{eff}}{L_{eff}} C_{ox} * (m - 1) V_T^2 * e^{\frac{|V_{G-S}| - V_{th}}{m V_T}} * (1 - e^{\frac{-|V_{D-S}|}{V_T}})
\] (3.4)

Just as in the EKV model, \(\mu\) is the electron mobility, \(W_{eff}\) is the effective channel width, \(L_{eff}\) is the effective channel length, \(C_{ox}\) is the capacitance formed by the oxide layer of the gate, and \(V_T\) is the thermal voltage defined by Equation 3.3. Additionally, \(V_{G-S}\) and \(V_{D-S}\) represent the voltage from gate to source and drain to source respectively. In addition to the minimal gate voltage, current will not flow without a potential voltage across the channel to induce a weak current, as the last term of our equation indicates.

Figure 3.3 indicates the narrow range that this model will accurately predict the current for a transistor. However, while it only maintains accuracy in a narrow range, it also has a nice property that it has a simple exponential relationship to the threshold voltage which will be very handy for solved integral expressions in later chapters.
Figure 3.3: Subthreshold Transistor Model Current Voltage Relationship

3.1.3 Transregional Model

The next model, described in Keller et al. [73], similar to the subthreshold model remains fairly simple in its expression by using an exponentiated second order polynomial instead of the first order used in the subthreshold model. Three fitting constants are then required to adjust the expression to more accurately model the charge approximation found in the EKV model, making this model a nice blend of the
two previous models.

\[ V_P = V_{Gate} - V_{th} \]

\[ I_{Spec} = \frac{2\mu W_{eff} C_{ox} m * V_T^2}{L_{eff}} \]

\[ i(x) = K_0 \cdot I_{Spec} \cdot e^{K_1 \left( \frac{V_P - x}{x_m V_T} \right) + K_2 \left( \frac{V_P - x}{x_m V_T} \right)^2} \]  

\[ K_0 = .54 \quad K_1 = .69 \quad K_2 = -.033 \]

\[ i_f = i(V_S) \quad i_r = i(V_D) \]

\[ I_{D-S} = i_f - i_r \]  

(3.5)

**Figure 3.4:** Transregional Transistor Model Current Voltage Relationship

Figure 3.4 indicates that while the Transregional model isn’t valid for all regions, it can apply to a wide range of gate voltages that aren’t zero but instead fairly close
to the threshold voltage with more accuracy than the Subthreshold model.

3.1.4 Factors that Affect Threshold Voltage

So far we’ve described the threshold voltage as some static value that is designed into a transistor using different transistor geometry. However, the threshold voltage can also be affected by other transistor variables. In recent processor design, DIBL has become a larger factor in low power circuit design. DIBL, short for Drain Induced Barrier Lowering, is the effect where the electrostatic charge at the drain will act similar to the charge at the gate that induces the electric field needed for channel formation. This has the effect of requiring a lower voltage to be applied to the gate and effectively a lower threshold voltage. This effect is typically linear with the drain to source voltage and so we can measure and simulate it using a linear constant with units of \( \frac{mV}{V} \). Likewise, temperature effects the surface potential of the channel and can also be adjusted for using a linear constant dependent on the transistor technology. This means that anytime the threshold voltage is considered in a dynamic system it should be represented as:

\[
V_{Th} = \delta \cdot V_{D-S} + k(T - T_0) + V_{Th0}
\]  

(3.6)

where \( \delta \) is the DIBL constant, \( k \) is the temperature dependence constant, \( T \) is the temperature, and \( V_{Th0} \) is the intrinsic threshold voltage at temperature \( T_0 \).

3.2 Statistical Models and Derivations

This section will now introduce some common statistical models and derivations that will be very useful in producing NTV based simulations and predictions. Probability and statistics is a very broad field but this section will focus on the basic workhorses: probability density functions (PDF) and cumulative density functions (CDF). PDFs are a continuous function that gives the probability of a random variable (RV) instantiating as a infinitely small specific value, and are related to CDFs which give the probability of a RV instantiating as a certain value or lower. Typical notation
is for a random variable, $X$, to have a PDF denoted $f_X()$, and CDF denoted $F_X()$ where all the following expressions are true:

$$
\text{Pr}[X \leq a] = F_X(a) = \int_{-\infty}^{a} f_X(u) \, du
$$

$$
\text{Pr}[a \leq X \leq b] = \int_{a}^{b} f_X(u) \, du = F_X(b) - F_X(a)
$$

$$
f_X(u) = \frac{d}{du} F_X(u)
$$

(3.7)

$$
\text{Pr}[\infty \leq X \leq \infty] = \int_{-\infty}^{\infty} f_X(u) \, du = 1
$$

$$
\text{Pr}[a \leq X] = \bar{F}_X(a) = \int_{a}^{\infty} f_X(u) \, du = 1 - F_X(a)
$$

Explaining line by line, the probability for the RV, $X$, is less than $a$ is, by definition, the CDF of $X$ evaluated at $a$ and can be computed using the PDF by integrating from $-\infty$ to $a$. Using the properties of integration, we can also compute the probability for a range of values by changing the bounds of integration or by directly subtracting the CDF evaluated at the maximum value from the CDF evaluated at the minimum value. Again using the properties of integration, it should be clear that the PDF can be determined by taking the derivative of the CDF and vice versa using the antiderivative. While a PDF can be described using any function, there is a requirement that the probability of any event being no more than 100% possible which means that the area under the curve for that PDF must be 1. And lastly, rather than calculating the possibility of an RV being less than a value, it is instead useful to know the probability that an RV is greater than a value. For convenience, this evaluation is named the complementary cumulative distribution function (CCDF) and is denoted as $\bar{F}_X(a)$.

### 3.2.1 Expected Values and Variance

Within probability, there are moments which provide information relevant to the distribution. The general form for the $n$th moment around a point, $p$, is defined
as:
\[ M_n = \int_{-\infty}^{+\infty} (x - p)^n f_X(x) \, dx \]  \hspace{1cm} (3.8)

For the first moment, \( p \) is usually 0 and the result is called the expected value and typically denoted as \( \mu \). In the case of randomly occurring events or samples, \( \mu \) also corresponds to the average or mean. The expected value is denoted as \( E[X] \) and described using:
\[ \mu = \int_{-\infty}^{+\infty} x \cdot f_X(x) \, dx \]  \hspace{1cm} (3.9)

In the case of 2\(^{nd} \) order or higher moments, \( p \) is chosen as \( \mu_0 \) and these are clarified by calling them central moments which indicates the moment is about the center of the distribution. The 2\(^{nd} \) central moment in particular is important for this dissertation and is called the variance, denoted as \( Var[X] \) defined as:
\[ Var[X] = \int_{-\infty}^{+\infty} (x - \mu)^2 \cdot f_X(x) \, dx \]  \hspace{1cm} (3.10)

It is also important to note that variance is a dimensionless measurement of the spread of the distribution and also equal to \( \sigma^2 \) which is proven in Equation A.2. Because of this relation to \( \sigma \) of a normal distribution, it is common to instead describe variance in terms of \( \sigma \).

### 3.2.2 Generating New PDFs using Change of Variables

For a PDF named \( f_X(x) \), it is possible to calculate the PDF of some other RV defined by \( Y = g(X) \). This is also called a “change of variable” and will become very important as we analyze NTV approximations for both timing and energy. If the function \( g \) is monotonic, then the resulting PDF for \( g \) is
\[ f_Y(y) = \left| \frac{d}{dy}(g^{-1}(y)) \right| \cdot f_X(g^{-1}(y)) \]  \hspace{1cm} (3.11)

The multiplication of the inverse function’s derivative is very important since we need to normalize the area under the curve to 1 to satisfy the fourth property in
Equation 3.7 and in essence is exactly the procedure performed for doing change of variables in integration. Also, we can evaluate moments, such as the expected value, directly from the original PDF rather than transforming to the developed PDF and then using Equation 3.9.

\[
\mu = \int_{-\infty}^{+\infty} g(x) \cdot f_X(x) \, dx
\]

or

\[
\mu = \int_{-\infty}^{+\infty} y \cdot \left| \frac{d}{dy} (g^{-1}(y)) \right| \cdot f_X(g^{-1}(y)) \, dy
\]  

(3.12)

Many times, the former equation is much simpler to derive than the latter, and can also be used for other techniques like partial expectations which will be explained in future sections.

For illustration, we will now derive the PDF, CDF, and E[x] for two distributions: normal and lognormal.

### 3.2.3 Deriving the Normal or Gaussian Distribution

One of the most common distributions for a RV is the Normal or Gaussian distribution named after Fredrick Gauss who developed it in 1809 along with contributions by Laplace. It is based on the Gaussian integral defined as:

\[
\int_{-\infty}^{+\infty} e^{-t^2} \, dt = \sqrt{\pi}
\]

(3.13)

Now we wish to make a change of variables to fulfill three properties: we want the expected value to be \( \mu \), we want the variance to be \( \sigma^2 \) as we mentioned in Equation 3.10, and we need the PDF to have an area under the curve of 1 as we mentioned in
Equation 3.7. Making a change of variables for \( t \) we can normalize the Gaussian Integral to:

\[
\begin{align*}
    t &= \frac{x - \mu}{\sqrt{2\sigma}} \\
    dt &= \frac{1}{\sqrt{2\sigma}} \, dx \\
    \int_{-\infty}^{+\infty} \frac{1}{\sigma\sqrt{2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \, dx &= \sqrt{\pi} \\
    \int_{-\infty}^{+\infty} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \, dx &= 1 \\
    f_X(x) &= \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}
\end{align*}
\]

(3.14)

Thus, by making these changes in variable, we have a PDF, \( f_X(x) \), whose area under the curve is 1 as proved in line 4. To see the derivations for \( E[X] = \mu \) and \( Var[X] = \sigma^2 \) please see Appendix A.

As we’ve explained in the general case, the Normal Distribution also has a CDF which is the evaluation of the integral of the PDF from line 5 of Equation 3.14 but from the bounds of \(-\infty \to x\). This doesn’t not have a closed form solution and thus is given it’s own function and symbol, \( \Phi(X) \), for the standard case of \( \mu = 0 \) and \( \sigma = 1 \). It may be useful in other parts of the dissertation to also utilize the special function,
erf() found in equation 3.15 which is closely related to Φ.

\[ erf(a) = \frac{2}{\sqrt{\pi}} \int_{0}^{a} e^{-t^2} dt \]
\[ \Phi(a) = \int_{-\infty}^{a} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx \]

\[ s = \frac{x - \mu}{\sigma} \rightarrow ds = \frac{dx}{\sigma} \]
\[ F_X(a) = \int_{-\infty}^{\frac{a-\mu}{\sigma}} \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{s^2}{2}} \sigma ds \]
\[ F_X(a) = \Phi\left(\frac{a - \mu}{\sigma}\right) \quad (3.15) \]

or

\[ F_X(a) = \int_{-\infty}^{0} \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx + \int_{0}^{a} \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx \]
\[ t = \frac{x - \mu}{\sqrt{2\sigma}} \rightarrow dt = \frac{1}{\sqrt{2\sigma}} dx \]
\[ F_X(a) = \frac{1}{2} + \frac{1}{2} \int_{-\infty}^{\frac{a-\mu}{\sqrt{2}\sigma}} \frac{2}{\sqrt{2\pi}} e^{-t^2 \sqrt{2}\sigma} dt \]
\[ F_X(a) = \frac{1}{2} \left(1 + erf\left(\frac{a - \mu}{\sqrt{2}\sigma}\right)\right) \]

We see the definition of erf in line 1 followed by the definition of Φ in line 2.

With a change of variables in either case, we can see the integration of \( f_X(x) \) for the desired bounds. In the case of Φ, the derivation is simple and straightforward. For using erf, we can break up the integral into two subintegrals, one of which can evaluate to \( \frac{1}{2} \) due to the evenness of the PDF and the other by using the \( erf() \) function.

Now that we have a full derivation of both the PDF and CDF for a Normal Distribution, we will apply the principles from Section 3.2.2.

### 3.2.4 Deriving the LogNormal Distribution

In many portions of this dissertation, we will analyze phenomenon that are functions of exponeniated polynomial variables that have a uniform distribution, so
it would be helpful to develop the statistical analysis of distributions that follow the function $L = e^X$ where $X$ is a Normal RV that was just derived in the previous section. Thus, we will now derive the PDF using Equations 3.11 and 3.14.

$$g(y) = e^y \rightarrow g^{-1}(y) = \ln(y) \rightarrow \frac{d}{dy}(g^{-1}(y)) = \frac{1}{y}$$

$$f_X(x) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$f_L(y) = \frac{1}{y \sigma \sqrt{2\pi}} e^{-\frac{(\ln(y)-\mu)^2}{2\sigma^2}}$$

(3.16)

We see that the distribution is normalized according to the input to maintain an area under the curve of 1. This also means we can create a CDF by performing a change of variables

$$F_L(a) = \int_{-\infty}^{a} \frac{1}{y \sigma \sqrt{2\pi}} e^{-\frac{(\ln(y)-\mu)^2}{2\sigma^2}} dy$$

$$z = \frac{\ln(y) - \mu}{\sigma} \rightarrow dz = \frac{dy}{y\sigma}$$

$$F_L(a) = \int_{-\infty}^{\ln(a) - \mu} \frac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}} dz$$

$$F_L(a) = \Phi\left(\frac{\ln(a) - \mu}{\sigma}\right)$$

(3.17)

Again, this derivation is very straightforward having developed the background definition of $\Phi$ in Equation 3.15.

$$E[L] = e^{\mu + \frac{1}{2}\sigma^2}$$

$$E[L|a < L < b] = e^{\mu + \frac{1}{2}\sigma^2} \left(\Phi\left(\frac{\ln(b) - \mu - \sigma^2}{\sigma}\right) - \Phi\left(\frac{\ln(a) - \mu - \sigma^2}{\sigma}\right)\right)$$

(3.18)

And lastly, in Equation 3.18, we can define for a lognormal RV, $L$, both the expected value and the partial expected value for a lognormal distribution. This will become very important in developing future models. Please see Appendix B for a full derivation if unfamiliar with lognormal distributions.

3.3 Conclusion

While this chapter is very deep in both transistor and statistical models with much of this information foundational for every trained engineer, every concept will
help prime and motivate the mathematical formulations of every future novel model proposed in the coming chapters.
Chapter 4

RUNTIME WITH ENERGY SAVING TECHNOLOGY

This chapter discusses the use of runtimes within current operating systems. The goal is to quantify and see the possible use of Dynamic Voltage and Frequency Scaling (DVFS) techniques with a standard OS and understand the shortcomings so that a future runtime system can adapt or improve on this.

4.1 Introduction

Energy efficiency of computing systems becomes a major issue, resulting in significantly growing costs to host and operate servers. Focusing on extreme performance, HPC centers optimize the design of machine rooms and computer systems in order to tame the non linear effect of computing and cooling on power consumption [101] [113]. Additional complementary approaches consist in working at a finer level on the control of the energy by the runtime software, which could lead to further improvements to the design of future systems and processors. The following chapter focuses on an innovative development correlating the different phases of an application workload, which typically alternates between memory bound to compute bound phases, to control the frequency of the processor. This chapter shares preliminary and encouraging results. While the Linux kernel schedules time slices or quanta and uses this metric as a method of utilization, this method can be an inaccurate way to measure the balance of system. Many HPC applications are bandwidth bound and so having the processor at a higher voltage and frequency is unnecessary since processors will simply spin I/O or memory requests.
Figure 4.1: Energy consumption from the wall and in the CPU on a 3.3 GHz Intel Xeon E3-1240 processor for the Quantum Monte Carlo for Chemistry application, see Table 4.1 for the complete hardware configuration.
Many components affect the energy consumption of a given architecture: the CPU, memory, fans, hard-disks, network, etc. Figure 4.1 shows the energy consumption when varying the processor frequency of the Quantum Monte Carlo for Chemistry (QMCC) program’s execution on a 3.3Ghz Intel Xeon E3-1240 processor. QMCC uses a powerful stochastic approach to solve the Schrödinger equation applied to computational chemistry [94]. QMC methods are, by nature, compute bound. [29] shows QMCC reaches a nearly perfect parallel efficiency, taking advantage of each additional CPU core provided. Measuring the energy consumption both from the wall and using the processor’s energy related performance counters, the CPU’s energy cost is more than half of the full energy bill for the tested system. Therefore, any energy saving technique generally starts with handling the processor’s energy consumption. In Figure 4.1, the execution time varies with the frequency almost linearly, indicating the program is compute bound and not memory or message bound. Moreover, note the percentage of energy consumption by the on-package performance counter as compared to the wall measurement. Since almost half the energy bill comes from the processor, the motivation to reduce processor energy consumption is one of the goals of our runtime named REST which stands for Runtime with Energy Saving Technology.

The dominant method for reducing the energy consumption is the DVFS technique implemented in both Intel Speedstep [8] and AMD’s Cool’n’Quiet [2] technology. These technologies allow the operating system to lower the frequency and voltage using P-States which the OS can request to change to, greatly reducing dynamic energy consumption caused from synchronous logic in a processor core. However, static power leakage present in all of a core’s transistors constantly consume power; therefore, any delay in completing computation may expend additional energy from the leakage. The problem has inspired the “Race to Sleep Policy” in which the highest frequency is used to complete the task as fast as possible, then the processor is turned off or power gated, which uses an order of magnitude less power. Inevitably, there are cases for both extremes and balancing acts to be performed. The choreography of trade-offs is the main focus of the chapter.
The chapter’s contributions are as follows:

- provide a memory versus non-memory phase detection system on two different Xeon architectures
- study energy consumption on modern architectures using real world applications, the NAS and SPEC benchmark suites
- present a software solution that can inspire how to reduce the energy consumption on a state-of-the-art HPC system

The remainder of the chapter is organized as follows. First, we present a phase detection system and explain why it is important to determine whether a program is accessing memory. Second, we examine energy consumption on two different Xeon architectures with from the wall readings. In the case of the 3.30 GHZ Intel Xeon E3-1240, we present a study on the correlation between the on-chip energy counter and total energy readings using a power meter. Third, both elements are combined into the Runtime Energy Saving Technology system (REST) to provide a cohesive system that reduces energy consumption by an average of 15.01% on the SPEC 2006 and 10.45% on the parallel NAS benchmark suite, while only degrading the execution time by respectively 5.95% and 3.74%. Finally, we compare the REST system to related works.

4.2 Phase Detection

Most programs alternate between memory and compute phases. REST’s purpose is to lower the core frequency during a memory phase and raise it during a compute phase. Therefore, Phase detection is a key component to any dynamic system. Furthermore, phase detection is a method for discovering control-flow profiles, cache behavior, or energy, to name a few parameters. Often, these parameters help a runtime system better schedule resources to a user program; however, its application can also be helpful for effective DVFS methods. There are many different techniques for implementing phase detection: using a previously profiled execution, performing simultaneous
hardware assisted profiling, static compilation, interrupt-based sampling, statically instrumented software, or a hybrid system such as JIT techniques that switches between original code, instrumented code, and optimized code.

In all techniques, the goal is to acquire the best level of information with the least overhead. Some methods offer less information, rendering phase capture difficult. Discovering whether a program is in a compute-bound or memory-bound phase requires knowledge of the entire system, which is generally unavailable in both static and profiling methods. Thus, REST uses performance hardware counters to give an indication of the state of both the processor and chip level memory activity.

In REST’s current implementation, the system considers three counters to determine the program’s behavior:

- **L2_RQSTS:MISS**: represents the number of second level cache misses, which correlates to the core’s contribution to a memory bound state
- **UNHALTED_CORE_CYCLES**: provides the number of cycles between two sampling points
- **SQ_FULL_STALL_CYCLES**: gives the number of cycles spent waiting for the superqueue (the serializing queue that interfaces with the memory controller) to empty itself; when it is full, the system stalls until a spot is available

In order to show the relevance of the three counters for the REST implementation, Figure 4.2 provides a synthetic benchmark showing the variations between phases. The benchmark alternates between a compute bound phase and a memory bound phase. Though the number of cycles per sample remains constant, the two other counters pertaining to memory saturation levels have different values between phases.

However, a SPEC application or a real-world application does not have discrete phases but instead has continuous transitions in phases. Generally, the counter values resemble more the ones from Figure 4.3. The considered program is part of a larger application called RTM, Reverse Time Migration, and is TOTAL’s proprietary code. As Figure 4.3 shows, the variations are more subtle but are still noticeable between
**Figure 4.2:** Hardware counters during the execution of a synthetic benchmark: showing how the counters evolve between a memory bound and a compute bound execution
Figure 4.3: Hardware counters during the execution of a real world application (Reverse Time Migration)
processing phases and memory oriented phases, especially in the zoomed box on the top right.

REST uses statistical sampling, which uses a back-off mechanism and reduces the sampling rate if the program’s behavior remains constant. Therefore, when considering a program that remains memory intensive, the phase detection system reduces the number of polls during the execution. However, if a change is detected, the system resumes the frequent polls to determine whether other changes are also occurring. Empirically, the best settings on the considered systems use a base sleep time of one millisecond and a maximum sleep time of seventy-six milliseconds.

4.3 Energy Consumption

Before trying to reduce energy consumption during the program’s execution, it is important to understand how a machine utilizes energy depending on the running application. Certain parameters such as data size, algorithms, and importance of a timely result, vary greatly between different programs and, therefore, require different levels of processing, memory use, and file storage. In order to accommodate the trend of varying demands on a system, general purpose architectures now are isolating the components and creating voltage and clock islands. For example, the 3.30 GHZ Intel Xeon E3-1240 architecture features CPU, GPU, and uncore voltage islands allowing different configurations of performance in each component.

Currently, Linux relies on the cpufreq governors, most commonly the OnDemand governor, a driver that still views a program’s consumption of resources as slices of processing time. Because of this, the OnDemand governor will increase CPU frequency if there is a time slot requested, unlike REST which observes how to balance the system. As the view of resource management has evolved, several projects have attempted to solve these problems for Linux [108] [6]. Regardless of its flaws, OnDemand is the standard driver for managing frequencies in Linux and to which the chapter compares REST.
Figure 4.4: Wall energy consumption and time execution for the SPEC program *Libquantum* depending on frequencies.
The next important question is: what is the best frequency to use for a given program? If the program is highly compute-bound, such as the SPEC program Gromacs, the best frequency to choose is the highest frequency. The highest frequency allows the program to finish faster, minimizes static energy losses, and therefore consume less energy. However, for a memory-bound program such as Libquantum, in Figure 4.4, the optimal frequency is the lowest frequency because it does not modify execution time but considerably reduces the energy consumed by 20.56% without any performance degradation. Observing a fixed execution time, regardless of the frequency, may seem counter-intuitive. However, the simultaneous execution of Libquantum on each core completely saturates the memory system, making frequency of the processor unimportant. Finally, for neither compute nor memory bound programs, the selection of frequencies will vary, and usually the best static frequency for minimal energy consumption is neither highest or lowest frequencies. In the case of these codes, the frequency to choose depends on user goals: either the lowest possible energy consumption or low energy consumption while maintaining high performance. One instance, shown in Figure 4.5, is the case of the RTM application. In these figures, the points in the figure were derived using a static run of the program at that frequency and the total energy consumed. This gave a means of validation for our decision making. For RTM, REST uses a linear function with the hardware performance counters presented in Section 4.2 to select between the three frequencies in the base of the cup and provides an energy savings of 19.35% with a performance degradation of 9% compared to a Turbo Boost execution. For programs Gromacs and Libquantum, REST selects directly the best frequency to reduce energy consumption.

As shown, the “Race to Finish” technique, by increasing the core frequency, does not automatically reduce the energy consumption. The rest of the chapter presents REST, its implementation and evaluation to demonstrate how automatic energy reduction is still possible.
Figure 4.5: Wall energy consumption and time execution for the Reverse Time Migration program depending on frequencies
4.4 REST

The previous sections presented how to determine a program’s current behavior and how it corresponds to the energy consumption when modifying the frequencies. The sections showed reducing frequency when a program is memory-bound does not degrade performance and, to reduce energy consumption in a compute-bound program, raising the frequency is the best option. The following section presents how REST utilizes the information to accurately reduce the energy consumption.

The REST runtime system is modularly composed in order to provide both high portability and configurability. Three main components form REST: the profiler, decision maker, and frequency change driver. In its current version, REST implements an interrupt-based sampling of the three different performance counters described in Section 4.2. The current profiling uses a new thread inside the user process and relies on \texttt{usleep} calls to perform necessary timer functions and context switches. At the end of the program, the thread is woken up and destroyed after writing out any profiler data for post-processing.

REST includes three different kinds of decision makers, which are responsible for receiving data from profilers, interpreting if the data signifies a phase shift, and determining if a frequency change is beneficial. The most naïve decision maker simply accepts all data as correct and always changes frequencies if necessary. The drawback of such a solution is the aliasing problem, where the program cycles between a compute-bound phase and a memory-bound phase. If the sampling rate is incorrectly set or if the phases are too short, REST switches frequencies constantly, raising the overhead on performance and energy consumption.

In order to correct the aliasing issue, most branch predictors contain a confidence level. The second decision maker implements the confidence level by verifying if the profile data is not simply transient noise. It maintains a simple history table of past profiler data samples, and once a phase shift is determined to be a true shift in compute or memory behavior, the decision maker changes frequencies. However, the disadvantage of a prediction system is two-fold. First, the overhead of calculating and
maintaining a history is not free and forces REST to be overly conservative because at a real phase change, REST waits a few samples to confirm the change and finally to modify the frequency. Second, the slow change in frequencies reduces the size of the phase to achieve the best energy savings in phase-shifting programs.

Lastly, the third implemented decision maker resolves both aliasing and best energy savings by studying past phase changes and predicting future changes. It feeds the phase shift information into a Markov predictor based on Beyler et. al [21] much in the style of [116]. The idea behind the third decision maker is to predict changes and, when confidence levels achieve a threshold, the profiler layer can turn itself off until needed for verification purposes. However, the overhead of the Markovian system is high, and it fails to handle unpredictable behaviors.

Figure 4.6: Comparison between the naïve and branch-predict decision makers
When the implementation of the three decision makers was finished, tests were performed and the naïve decision maker sufficed. Figure 4.6 presents a percentage point comparison in energy savings and performance degradation between the naïve and the predictor decision makers. Any bar above the zero line represents the naïve predictor’s better performance, whether energy or execution time. As is seen, the difference between the two is limited; though the predictor may achieve better results in certain cases, the effort required is not automatically worth the complexity. The third decision maker, using a prediction system with an underlying Markovian model, has a huge overhead and is not able to achieve confidence in the patterns it observes. Therefore, it almost never starts predictions and therefore the whole system degrades the execution time.

The final component of REST is a thin frequency changing driver. In the current implementation, the experimentation platform uses the Linux `cpufreq` module to manage P-state transitions within the kernel. To provide a method for assembling these three components, REST transparently initializes itself by the use of the `LD_PRELOAD` environment variable at the start of a program and configures all necessary choices through REST specific environment variables. The previous architectural choices allow for a flexible and configurable runtime system.

<table>
<thead>
<tr>
<th>Model Number</th>
<th>X5650</th>
<th>E3-1240</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores</td>
<td>2 × 6</td>
<td>4</td>
</tr>
<tr>
<td>Memory</td>
<td>8 GB</td>
<td>4 GB</td>
</tr>
<tr>
<td>PowerMeter</td>
<td>Yokogawa WT210</td>
<td>Hardware counters</td>
</tr>
<tr>
<td>Operating System</td>
<td>Linux 2.6.38</td>
<td>Linux 2.6.38</td>
</tr>
<tr>
<td>Compilers</td>
<td>GCC 4.6 - Ifort 12.1</td>
<td>GCC 4.6</td>
</tr>
</tbody>
</table>

**Table 4.1:** Experimental Testbed
4.5 Experimental Results

For the experimental testbeds, Table 4.1 shows the hardware and software used in the REST experiments. As previously presented, the REST system was tested with synthetic benchmarks. Two benchmark suites were used to show REST’s energy saving capability. First the sequential SPEC2006 were used. The applications, though sequential, were executed on each core simultaneously to simulate a full workload. An internally developed tool, MicroLauncher [7], ensured all processes were pinned, synchronized, and uninterrupted to attain the most stable results. The SPEC programs are relatively complex and complete programs. Combined with our parallel launcher, the setup ensures a more realistic emulation of future programs and represents a more general study compared to smaller parallel benchmark suites. For some of the SPEC2006, the simulation required more memory than was physically available and forced paging to occur. Such cases were dropped from the study because correctly tuned programs always fit into the available physical memory. Finally, with the goal of completeness, the parallel NAS benchmark suite shows REST’s abilities since its programs are parallel. In all, two real applications, RTM and QMCC, as well as two benchmark suites, SPEC and NAS, were used to prove REST’s ability to reduce energy consumption without degrading the program’s performance too much.

Having explained the phase detection and REST architecture, it is important to see if energy consumption can be reduced. In regards to efficacy, Figure 4.7 shows how REST is able to reduce the energy consumption for the SPEC 2006 benchmark suite. Notice, in some cases like Libquantum, throttling the frequencies actually increases performance, likely due to reduced conflicts in buffers and coherence buses. One feature, which has not been mentioned, is REST only selects fixed frequencies, i.e. it does not select the Turbo Boost frequency. Had the system allowed Turbo Boost frequencies, the results would have varied slightly. The reason REST restricted itself to static frequencies was to provide insight on the cost effectiveness of the Turbo Boost mode. If a user considers energy savings though, it is moot to select Turbo Boost since, from the results, Turbo Boost gains in performance but always reduces the power/performance...
Figure 4.7: REST energy savings and performance degradation on the SPEC 2006 benchmark suite, with the most naïve decision maker.
In addition to SPEC results, Figure 4.8 presents the results obtained on the parallel NAS benchmark suite. The runs were performed using Class C benchmark sizes. As BT, CG, MG, and SP in Figure 4.8 show, there is opportunity for large energy savings at minimal performance degradation when MPI communications overlap the slower processing. However, LU, which is highly coupled to its messaging and scheduling, suffers from skew introduced by the REST runtime.

4.6 Related Work

The current chapter contains multiple related work domains, such as DVFS techniques and profiling program behavior at runtime. DVFS systems have been presented in the past for many different types of systems [57] [66] [75] [131] [91], occasionally focusing solely on multi-process systems such as MPI in the case of [57] [83] [109]. Though useful, the techniques considered latency induced by message passing or memory sharing as energy saving targets, without considering the general trend of compute bound versus memory bound as a decision maker.

REST considers each core separately to determine the correct frequency to use.
Though modern processors actually handle the frequency of the cores in the form of packages, initial experimental work showed REST providing better results when considering each core independent whether true or not. Therefore, the hardware’s limitation, though real, is ignored by the software stack level in order to provide better results. The frequency is used depending on the program’s current computation load, ignoring whether it is an OpenMP program, a MPI program, or even a sequential program. REST handles the three types of programs in a unified manner, only the startup of the system differs. The differentiation enables REST to uniformly handle any type of program and any parallel paradigm on a single given node.

Isci et. al provide a similar hardware counter-based system to REST [66]. The authors propose a phase prediction system using a Global Phase History Table, which produces next-phase behavior deductions based on previous samples. Once a prediction is performed, the framework is linked to a DVFS method, which reduces the frequency if memory-bound and raises it if cpu-bound. REST is similar in the concepts of frequency decision making but differs from their approach by handling multi-process applications. Also showing, in the DVFS scenario, predictions are not required; a simple naïve decision maker suffices and reduces the incurred decision making overhead. REST was evaluated using the energy consumption from the wall and not from the CPU, a methodology more widely used.

Another related work domain regards determining whether the application is compute-bound or memory-bound. As in the DVFS section, there have been prior works in the phase detection domain [16] [66] [93] [43] [133] [65] [116]. Static and dynamic systems differ in approaches. Static techniques rely on offline profiling or instrumentation to learn about the execution behavior, whereas the dynamic methodologies rely on low overheads, since they are accounted in overall execution, to determine solutions during the same execution.

Static systems rely on code instrumentation [50] [131] or profiling mechanisms [109] and require a first run to provide a profile of the program. Compared to REST, the systems assume the compiler or a first run is able to provide such information,
which is true for programs that have a predictable behavior. However, REST is not only able to handle any type of complex code, which has various behaviors between runs, but it is also able to consider codes contained in already compiled libraries or in pre-compiled object files.

Dynamic systems profile the code at runtime using low overhead techniques \[57\] \[66\]. Some require modifications to the code base, to the hardware \[91\] \[75\], use tools such as VTune \[16\], or use a simulator \[90\]. REST, implemented on modern systems, provides a software layer that utilizes hardware performance counters and gives users a plausible energy consumption improvement with their current hardware set-up.

4.7 Conclusions

REST presents a technique to study program behavior in order to correlate selected frequencies with energy consumption. The solution reduces the obtained energy consumption, while minimizing the execution time degradation. REST studies how much the application accesses memory and reduces the frequency accordingly. Doing so at runtime without modifying the base source code allows REST to be transparent and user-friendly for HPC-type applications.

The system characterizes each core’s behavior separately and determines the best frequency for each independently. With the implementation of three different decision makers, on the two considered architectures the simplest solution based on a sampling mechanism is the most efficient technique. The system reduces on average the energy consumption by 15.01% on the SPEC 2006 and 10.45% on the parallel NAS benchmark suites while only degrading performance by 5.95% and 3.74%, respectively. The benchmark suites and two real-world applications show REST’s effectiveness.

The chapter also includes an example of the energy related hardware counters on the 3.30 GHZ Intel Xeon E3-1240 for the QMCC benchmark suite and a comparison between CPU-only energy consumption and from the wall. The study shows, even on modern systems, the processor still consumes more than 48.8% of the total energy consumption and should still be the general focus of any energy saving techniques.
Lastly, the shortcomings of a general OS have been exposed such that new methods and metrics for efficiently controlling the performance/energy tradeoff are needed. In future chapters, we hope to give an idea of new methods which take into consideration process variance and near threshold voltage practices.
Chapter 5

VARIANCE MODELING

The previous chapters have so far given a motivation for increased awareness in energy efficiency and attempted a runtime on current OS and commercial platforms to aid in transitioning software to this methodology. Further chapters will now look to the future of what is possible without the constraints of backwards compatibility and the demands of commercial product design and will thus be based on simulations. Each chapter will describe a novel approximation or analysis motivated by physical phenomenon and founded on previous fields of work. Combining all these approximations will provide a complete methodology that both functional simulators and the runtime systems developed on top of those simulators need to be designed with energy efficiency as a possible optimization goal. Having armed ourselves with large numbers of tools for both modeling transistors and performing analysis, the dissertation will propose several novel approximations. This chapter in particular will begin by looking at the base source of the nonlinear aspects of a processor which we covered in Section 3.1: threshold voltage variation. We call this model the Pelgrid model since it combines the Pelgrom model and grid based variation mapping.

5.1 Background

The primary nonlinear relation is the threshold voltage of a transistor. This nonlinearly determines both the leakage of a transistor when it is off and determines the on current which directly relates to the frequency of a processor. In the case of standard design, the threshold voltage affects the frequency of a processor quadratically, while at near threshold, this goes back to a near exponential effect. Due to this relation, threshold variance (simply variance for the rest of this chapter) will have a larger
impact on near threshold designs in a future chip. Variance has two sources: random and systemic.

Random variance comes from manufacturing processes that rely on diffusion or other methods that follow brownian motion. Examples found in the literature would include Random Dopant Fluctuations (RDF) or Metal Gate Grainulairity (MGG). These have no spatial relation and instead only rely on the size of the transistor being created to determine the variance of its threshold voltage. Furthermore, these variances are truly random, making it difficult to design around.

The systemic variance is related to the lithographic and environmental processes that have slight spatial errors. Errors caused by systemic variances would include Gate Edge Roughness (GER), Line Edge Roughness (LER), or Fin Edge Roughness (FER) in the case of finfets and trigate transistors. When modeling both types of these variances together, the standard model in industry follows the seminal work of Marcel Pelgrom [105, 86] which describes the mismatch between two transistors as:

\[ \sigma^2(\Delta P) = \frac{A_P^2}{WL} + S_P^2D^2 \] (5.1)

where \( A_P \), the random variance coefficient, and \( S_P \), the systemic variance coefficient, are fabrication constants and \( W \) and \( L \) are the width and length of each transistor and \( D \) is the distance between the two transistors. These factors will generate a statistical variance \( (\sigma^2) \) or likelihood of a certain threshold delta between transistors \( (\Delta P) \). This variance will then manifest itself in chip designs in three aspects measured by industry: Wafer to Wafer (W2W), Die to Die (D2D), and Within Die (WID). To model the variance in our simulator, we must do two things: make a prediction of the magnitude of the systemic and random components and to find a way to generate the spatial correlation of the systemic variance for a simulator to use as input. Section 5.2 will provide a survey of predictions for several types of transistor geometries and their simulated variances. This survey will form the basis for a reasonable estimate of the magnitudes of the random threshold variance that will be required for the timing and leakage models. Section 5.3 will use the trends from Figure 7 of [98] to assume a D2D
and WID Variance that our new approximation method must match to simulate the threshold spatial variance of the chip which will create difference processor frequency capabilities and leakage currents. This will also couple with the temperature model so that leakage and temperature prediction is more accurate.

5.2 Random Variance

For transistor variance, there are many academic estimates and simulations based on many assumptions of the underlying lithographic process. This is further exacerbated by the fact that there are multiple type of transistors with which circuits can be built. Some of the promising technologies in the 2018 timeframe are FD-SOI [54], trigate [121, 100], and finfet transistor [32, 126]. There are many tradeoffs between these types in terms of performance, manufacturability, price, and energy consumption. Because of the complexity of these tradeoffs, we can only make an estimated guess at what variance will be in 2018-2020. The most detailed analysis for transistor variance is Wang et al [126]. They show the RDF, MGG, FER, and others for 14, 10, and 7nm finfets with correlations so that a best estimate for the 2018 timeframe will have a random threshold variance of 30-40σmV while spatial variance will be in the 15-20σmV like it is today. However, if a new technology like EUV (Extreme UltraViolet) lithography [70] or alternatives [106] were to be available, this could potentially change these assumptions and thus predictions.

5.3 Systemic Variance

5.3.1 Weighting for Spatial Correlation

Next, our model must provide a realistic and general purpose means to model the systemic variation and provide the spatially correlated data for different processors as input into a simulator. For this, we use the same technique as many other well known studies [13, 92, 27] which provides the spatial correlation through the use of a quadtree grid structure to assign random variables to different locations as shown in Figure 5.1.
In this model, the concept is to use random variables at each level of the quadtree to create spatial correlation since many nearby processors will share the same assignment at the upper levels of the quadtree. Because we are using Normally Distributed Independent Random Variables ($RV_{NDI}$) which we will denote as $X$ in this section, there are special properties that will make the quadtree easy to use when variance is given as a normal distribution.

\[
\begin{align*}
Var(X) &= \sigma(X)^2 \\
E(\sum X_i) &= \sum E(X_i) \\
Var(\sum X_i) &= \sum Var(X_i)
\end{align*}
\]  

(5.2)

$\sigma$ is the standard deviation of the normal distribution for the $RV_{NDI}$, $E()$ is the expected value of the expression, and $Var()$ is the variance of the expression, all of which we described and derived in Section 3.2.
5.3.2 Using Pelgrom to Calculate Weights between Levels

Using Equation 5.2, we can scale the $\sigma$ of $X$ from the Unit Normal value of 1 by using a weight we will denote as $\omega$. We generate the weight, $\omega_l$, for each level $l$ in the quadtree such that the final variance of the sum is a specified target. However, it is unclear what should the weighting of the different levels be and foundries are extremely reluctant to share specific space correlation data since it is considered trade secret information. Indeed, in [13], where we improved upon their algorithm, they make no mention of the weights used for test results and is assumed to be provided by Motorola. If more weight is allowed at the lower levels, lower spatial correlation will exist at the upper levels and make the chip look more like white noise. However, if the higher levels are allowed more weight, large amounts of spatial correlation will exist and create areas of the chip where the threshold is generally higher or lower than the expected value. Without any empirically derived coefficients for the weights, we propose a general method that uses Pelgrom’s model as a guide for determining both weighting at each level and weights for each specific processor.

![Intuition for Quadtree Weighting for Proper Spatial Correlation](image)

**Figure 5.2:** Intuition for Quadtree Weighting for Proper Spatial Correlation
Initially the expected value for a region is $\mu$ (we assign $\mu$ to 0 for the purposes of displaying variance although a true map would have a target threshold voltage), however once the RV is assigned, the expected value for that region shifts from 0 to the weighted RV with the centroid being the expected center. Since the centroid of a child in the quadtree is half the distance from the centroid of the parent as compared to the parent and its parent, it stands to reason the variance going down a level should be $\frac{1^2}{2^2} = \frac{1}{4}$ as much since Pelgrom states the variance is proportional to $D^2$. This creates a geometric series for an N-level quadtree where a weight down the next level is a quarter of the previous level. The identity for a geometric series is listed in Equation 5.3 for reference.

$$\text{Geometric Series} \quad \sum_{i=0}^{n} z^i = \frac{1 - z^{n+1}}{1 - z} \quad (5.3)$$

Using that identity, we weight each level appropriately and normalize for a target variation for a 256 block chip (quadtree depth of four). This is formalized in Equation 5.4

$$\sum_{l=0}^{3} \omega \cdot .25^l = \omega \cdot \frac{1 - .25^4}{1 - .25} = \text{Var}(\text{Target})$$

$$\omega = \text{Var}(\text{Target}) \times \frac{1}{1.328125}$$

$$\omega_{\text{level}} = \omega \cdot .25^{\text{level}} \quad (5.4)$$

$$\text{Block}_0 = \sqrt{\omega_0} \cdot X_{0,0} + \sqrt{\omega_1} \cdot X_{1,0} + \sqrt{\omega_2} \cdot X_{2,0} + \sqrt{\omega_3} \cdot X_{3,0}$$

where $X_{\text{level,region}}$ and $\omega_{\text{level}}$

As the equation describes, each block in a 256 processor simulation is a function of four RVs with different weights at each level. Note that one RV is unique for each processor while the other three are shared with 4,16, and 64 different processors. And while the RVs may be unique in different regions of a level, every RV in that level receives the same weight. By using Equation 5.2, the addition of these four RVs, weighted using the square root of their level variances, will also have a variance of the
total target variance. Figure 5.3 shows the resulting processor map that represents
the deviation of the systemic threshold voltage as a heatmap.

![Spatial Variation of Chip Threshold Voltage](image)

**Figure 5.3:** A Blocky Quadtree Result

Here you see that the majority of the chip is near zero deviation and some
regions are above and some below in much a normal distribution of frequency. It is
also very spatially correlated since the top level weight is much greater than the lower
level weights of the quadtree.
5.3.3 Using Pelgrom to Interpolate within a Level

As one can see, the current method has noticeable edge effects as stated in the literature [22]. This is highly unrealistic compared to actual physics because large discrete gradients would not exist. This is because the high level assignments of the quadtree break the Pelgrom rule by homogeneously assigning the RV across the region. In order to remove that gradient, we again use the notion of weighting according to the distance to interpolate every RV at a given level to every leaf node at the lowest level. Unlike the multilevel normalization, there is no easy geometric series formula so instead the weights are totaled inversely proportional to the squared distance (in keeping with the Pelgrom model) to every RV source and then for that RV are normalized using that total to the proper $\omega_{\text{level}}$ calculated back in Equation 5.4. Thus the closer you are to an RV, the larger your weight from that source while still having the same variance at every level. Now each processor is composed of weighted RVs from every region in every level except the last one where it has an RV assigned solely to itself. This derivation is expressed in Equation 5.5 and graphically represented for the first level weights of a single RV (ie $\sqrt{\omega_{0,0,i}}$ for all i when target variance is 1) in Figure 5.4.

\[
TotalWeight_{l,i} = \frac{1}{\sum_{j=0}^{4^{l+1}-1} \frac{1}{\text{Distance}(\text{Block}_i, X_{l,j})^2}}
\]

\[
\omega_{l,j,i} = \omega_{\text{level}} \cdot \frac{1}{\text{Distance}(\text{Block}_i, X_{l,j})^2 \text{TotalWeight}_{l,i}}
\]

\[\text{thus } \sum_{j=0}^{4^{l+1}-1} \omega_{l,j,i} = \omega_{\text{level}} \quad (5.5)\]

\[
\text{Block}_i = \sum_{l=0}^{N-2} \left( \sum_{j=0}^{4^{l+1}-1} \sqrt{\omega_{l,j,i}} \cdot X_{l,j} \right) + \sqrt{\omega_{N-1}} \cdot X_{N-1,i}
\]

where $N = \text{Levels}$, $X_{\text{level,region}}$ and $\omega_{\text{level,region,block}}$

Thus, for a 256 block chip, each processor is a uniquely weighted combination of 84 RVs and a single unique RV. This gives a smooth weighting factor that still respects Pelgrom’s model and gives the proper variance probability.
Figure 5.4: Spatial Smoothing Algorithm

The resulting maps look like Figure 5.5 and corresponds very well to the frequency mapping shown in Dighe et. al [44]

Higher fidelity quadtrees can also produce very fine systemic variance that could then be filtered to provide more accurate results like in Figure 5.6.
Figure 5.5: Final Approximating Algorithm

Figure 5.6: High Resolution Variance Map
5.3.4 Verifying the Variance

As already mentioned, verifying against chip foundry data would be very difficult to obtain but we can make assurances that D2D and WID variance is correct and can be adjusted to reported WID and D2D measurements provided by foundries. Additionally, we need to verify that the spatial correlation matches what is seen in real world chip tests.

Figure 5.7: Histogram to Verify WID Variance Portion

Figure 5.7 shows a histogram of 10,000 experimental chips generated in Python and categorized in order to verify what amount of the quadtree variance manifests
as WID variation or D2D variation. Additionally, we use the same methodology as Friedberg et al. [51] to perform the distance-based autocorrelation and statistically show the spatial correlation within our chips. For determining the portion of variance that is WID, it appears that a third of overall variance is WID with a lognormal distribution around the expected variance. Additionally, we see that compared to a real chip, the spatial correlation is proportional to distance however it appears that the overall correlation is much higher and the slope is shallower than what we see in Friedberg et al. due to the over estimated D2D variation the algorithm naturally produces. To account for this and also to calibrate the D2D variation to a future technology target, we can offset all chip values by a scale factor of that particular chip’s average calculated in Equation 5.6.

\[
\sigma_{\text{Sim}} = \sqrt{\sigma_{\text{WID}}^2 \cdot 0.3^{-1}} \\
\text{ScaleFactor} = \frac{\sqrt{\sigma_{\text{Sim}}^2 - \sigma_{\text{WID}}^2 - \sigma_{\text{D2D}}^2}}{\sqrt{\sigma_{\text{Sim}}^2 - \sigma_{\text{WID}}^2}} \\
\text{ScaleFactor} = \frac{1.5\sigma_{\text{WID}} - \sigma_{\text{D2D}}}{1.5\sigma_{\text{WID}}} \\
\text{Map} = \text{Map} - \text{ScaleFactor} \cdot \text{Map}_{\text{AVG}}
\]

For the different \( \sigma \), the \( \sigma_{\text{Sim}} \) designates the variance fed into the interpolation function to produce WID variation. The \( \sigma_{\text{WID}} \) and \( \sigma_{\text{D2D}} \) designates the desired target variations. Since we know 30% of the interpolation scheme manifests as WID variation, we can set the simulation input in line 1. Then we can create a scale factor since we know the remainder of the variance manifests as D2D. We do this by preserving the desired variance and then normalizing against the remainder. Then this scale factor will shift the chip average to satisfy the desired sigma. Having adjusted the D2D variation down to 15 \( \sigma \)mV as estimated from [102] as a reasonable variance, we see the characterization for these sets of inputs very closely matches the studies performed by Friedberg et al. including a mild increase in correlation at large distances that is neglected by other models.
Characterization for Target of WID 15 mV and D2D of 15 mV

![Graph](image)

**Figure 5.8:** Statistical Analysis of Chip Variance

### 5.4 Related Work

This work is built off the quadtree technique of Agarwal et al. [13]. However, they don’t provide any insight into the weighting levels where we give a reasoned model for proper weighting in the absence of specific foundry data. We also are similar in technique to Khandelwal and Srivastava [74] in that we perform spatial interpolation between RV sources. However, we give an intuitive reason to use a squared distance interpolation of the variance rather than a purely distance based interpolation of the RV directly. This work most closely compares to the Varius project [112, 72]. Their
work similarly creates a map using statistical models so that an example chip can be used within a simulation framework. Varius is inspired by the same studies utilized for verification in this chapter and use a distance-based correlation function of threshold voltage amongst transistors [51]. While Varius has created a reasonable relation for variance on the particular process, it has no intuition as to how the spatial correlation would change across processes and so the configuration of a certain process will take some level of calibration. Furthermore, our model is based off of the Pelgrom model and has been extensively evaluated across many different process nodes and foundries. This makes our framework slightly more powerful and straightforward to use since we can project possible chip variation for any WID and D2D metrics in a straightforward manner. While the distribution of WID variation wasn’t verified due to a lack of a public data source, we believe this distribution is correct due to the use of Pelgrom’s model and also seems to be corroborated by the frequency ratio prediction distributions in Varius NTV [72]. In addition, we can specify the granularity of the map so that fidelity\time tradeoffs can be made, for example, when doing Monte Carlo simulations. More importantly, this work aspires to join the methods of quadtree grid based mapping with spherical mapping and help to bridge both research communities. This model and Varius are not for transistor level modeling since neither captures the difference in horizontal and vertical correlations but instead only work in radial terms. Nonetheless, it is sufficient for core level predictions and chip level modeling for the purpose of runtime system development.

5.5 Conclusion

We have presented a new methodology that generates realistic systemic variance patterns that can be utilized by a functional simulator. Additionally, we’ve made estimates about expected random variance that will be a base assumption in future chapter models. Now that these assumptions and inputs have been generated, we can present the timing and energy models that will depend on these threshold voltage variations.
In Chapter 3, we discussed the basic models of a transistor but without application. In this chapter, we will start to find general approximations that are appropriate for NTV using these transistor models paired with established high level approximations. The ultimate goal of the approximations is to use them to fit empirical observations or predictions to our model using this methodology so that we can model the behavior of processors for runtime development. Keeping this in mind, we are not expecting to validate this model against circuit simulations but instead only wish to match these models to observations. Before diving into novel work, we wish to examine some verified high level models that have been used in industry for predicting timing performance of processors.

6.1 Background

6.1.1 Alpha Power Law

In 1990, Sakurai and Newton [110] published a mathematical model for predicting the change in timing with the change in voltage. Their work extended the first order model of Shockley that states a long channel transistor in saturation can be described as:

\[ I = \frac{\beta}{2}(V_{gs} - V_{Th})^2 \]  

(6.1)

This was modified by Sakurai so that a short channel transistor in saturation would operate in a relation slightly smaller than quadratic by a term he calls alpha. This work was later extended [26] with a physical explanation as to why alpha made sense,
namely that the carrier velocity saturated in the channel. This effect led researchers to add to the short channel models to include saturation voltages to account for these effects. Sakurai also had a timing model that looked at the critical path of transistors as a series of cascading charging capacitors that would follow a standard RC relation with time.

$$I = \frac{\beta}{2} (V_{gs} - V_{Th})^\alpha$$

$$\tau_{gate} \propto \frac{C_{gate} \cdot V_{DD}}{I} \rightarrow \frac{C_{gate} V_{DD}}{(V_{DD} - V_{Th})^\alpha}$$

$$f_{req_{gate}} \propto \frac{C_{gate} V_{DD}}{C_{gate} V_{DD}}$$

$$\tau_{proc} = \sum_{n=0}^{\#\text{critical gates}} \tau_{gate(n)}$$

$$f_{req_{proc}} = \frac{1}{\tau_{proc}}$$

As Equation 6.2 states, the gate delay time, $\tau_{gate}$, will be a function of input capacitance, $C_{gate}$; gate voltage (for CMOS this will be proportional to $V_{DD}$), and the current, $I$. And the time delay for the whole processor will be the summation of all the individual gate delays of the critical path. Following this and the Shockley model, it made sense that frequency would be linearly proportional to the drive voltage. However, it should be noted that a clear assumption of the alpha-power model is that the gate voltage was well above the threshold voltage and that temperatures were reasonable. These assumptions will surely be false with an NTV processor where compute capability is expected to be grossly overprovisioned for the thermal power envelope, and threshold voltages will nearly be comparable to driving voltages. But our model, similar to Varius-NTV [72], will instead replace the alpha-power current function with a near threshold compatible model in order to predict processor frequency capabilities.

6.1.2 Statistical Static Timing Analysis

Static timing analysis [23] is a computationally cheaper way to perform timing analysis than a full circuit simulation for the purpose of finding optimal transistor sizing and critical path reduction. In regular deterministic static timing analysis, each
gate is a node in a graph and arrival times into each node go into a max operation, then
the time for the gate is added and this becomes the output time that then has some
interconnect delay added to become the input for the next node in the graph. When
a deterministic method was used, that is to say the gate delay is a static fixed time,
the method was simple for finding critical paths and sizing other transistors down to
reduce leakage and fan in or fan out capacitances. However, as process variance has
affected the overdrive voltage more, the method of statistical static timing analysis
(SSTA) has developed. In this model, arrival times now are presented as a Cumula-
tive Distribution Functions (CDF) with a probability of arrival, and the gate delay is
represented as a Probability Distribution Function (PDF). The field has divided into
main methods: block based analysis [42, 134] and path based analysis [125]. For block
based, the computation is done from the perspective of the logic operation. First,
MAX operations are performed for all inputs using the CDFs, converted to PDF, then
the ADD operation is performed with gate delay, and finally tranformed back to CDF
for propogation. However, there were added complications which is that two inputs
could have correlation because of a shared interaction from a predecessor node. This
problem, known as reconvergent paths, has led to computationally intense solutions.
On the other hand, path based models assume they are the critical path and thus
do not require the MAX operation, since they assume they are always the last input.
However, the number of paths in a processor can be extremely high. A 64-bit barrel
shifter alone has 4032 datapaths (64 × 63) through the multiplexers let alone the con-
trol logic. Thus, it is necessary to use a method for rejecting paths quickly so that the
problem can be tractable. For a more in depth state of the art, we direct the reader
to [23, 42].

6.2 Intraprocessor Timing

Our goal in this derivation is to provide a back of the envelope approximation
for the impact that variation will have on pipeline timing which can be utilized in a
processor timing model. For the timing between gates inside a processor, it was not
possible to find a closed form analytical solution and so we provide a best guess hybrid
SSTA inspired approach to estimate the timing penalty due to variation. We can use
a hybrid approach only because we are not actually calculating a real RTL design, but
instead making reasonable approximations to determine the affect random variance
will have in a wide pipeline with many logic paths. We only address random variation
since the spatial variation of two transistors within a millimeter should be less than
$3 \sigma mV$ where as the random variation will be 30-40 $\sigma mV$.

6.2.1 Methodology

We assume that the same target threshold voltage is set for all transistors, that
standard CMOS logic rather than Complementary Pass Logic or a dynamic logic is
used, and that nearly minimal sizing is used for the gates such that variances cited in
Chapter 5 will be applicable. Given those assumptions, we will build up the approxi-
mation from the transistor to processor.

6.2.2 Transistor Level

In Chapter 5, we looked at the threshold voltage which nonlinearly controls the
current of a FET transistor and in near threshold operation has the largest impact on
variability, and in Chapter 3 we looked at Equation 3.11 for how to generate PDFs
from other PDFs and monotonic functions. For our transistor level approximation,
we take the EKV model from Equation 3.1 and generate a normalized drive current
approximation using the general interpolation function. Recalling all these things,
Equation 6.3 shows the derivation.

\[ f(V_{Th}) \rightarrow I = I_0 \cdot \ln^2\left(e^{\frac{V_{DD}-V_{Th}}{2nV_T}} + 1\right) \]

\[ f^{-1}(I) = -2nV_T\ln(e\sqrt{I_0} - 1) + V_{DD} \]

\[ \frac{dI}{dI} = \frac{-e^{\sqrt{I_0}} \cdot nV_T}{(e\sqrt{I_0} - 1)\sqrt{I_0}} \]

\[ I_\mu = I_0 \cdot \ln^2(e^{\frac{V_{DD}-V_{Th}}{2nV_T}} + 1) = 1 \]

\[ I_0 = \frac{1}{\ln^2(e^{\frac{V_{DD}-V_{Th}}{2nV_T}} + 1)} \]

\[ f_X(I) = \frac{e^{\sqrt{I_0}} \cdot nV_T}{(e\sqrt{I_0} - 1)\sqrt{\frac{I_0}{I_0}2\pi V_{Th,\sigma}}} e^{-\frac{5(-2nV_T\ln(e\sqrt{I_0} - 1) + V_{DD}-V_{Th})^2}{(V_{Th,\sigma})^2}} \]

Here we see the transistor’s ability to switch will be most impacted by the drive current that is represented by the EKV model in line 1. Next we see the inverse function that can translate a current to a given threshold voltage and it’s differentiation with respect to current. We then wish to analyze performance compared to the nominal threshold voltage so we define the nominal current as \( I_\mu \) and make it 1. In line 5, we redefine \( I_0 \) as a function of nominal values. Applying change of variables, we arrive at a final PDF that can predict the probability of the drive current given a \( \mu \) and \( \sigma \) of the threshold voltage. Recalling from the Pelgrom model in Equation 5.1, the \( \sigma \) of a transistor scales with \( \frac{1}{\sqrt{W}} \). In Figure 6.1, we make predictions for different transistor sizes using the PDF and this relation.

### 6.2.3 Gate Level

Taking these PDFs for each transistor width, we then will construct a PDF for the gate delay. We will first construct the PDF for an inverter and different input NAND gates as proxies for more complex logic. We will also use the traditional 2:1 width ratio for P versus N type transistor although this may not be optimal in the near threshold \[85\] but this will change the variance for each transistor, which is important for this analysis. And we always use a 2 wide PMOS transistor even though it could be
Figure 6.1: Probability of Transistor Speed from Nominal

... reduced, since this is a reasonable compromise to remove the most variable transistors. 
As to calculating gate times, for any logic there is a rise time and a fall time for a 
gate which relates to the pull up or pull down network of the gate. We will make 
the assumption that the gate delay is the maximum of these two times, or to say this 
conversely using Equation 6.2 the frequency of the gate is the minimum current from 
the pull up or pull down network. In the case of parallel transistors, we must consider 
each one as a possible drive current source. And in the case of transistor stacks, we 
use the approximation of the single transistor equivalent without dealing with DIBL or 
body effects, but because we are dealing with random variables (RVs), we sum up the 
RVs of every transistor PDF and divide by the number in the stack squared rather than 
the standard approximation method of simply taking a single transistor and dividing 
by the number of equally sized transistors. Finally, we take the minimum of all current 
sources to be the gate delay of the circuit. To help with the statistical operations...
involved, see Equation 6.4 where $X_i$ is an RV, $f_X()$ is a PDF and $F_X()$ is a CDF.

$$f_{X_1+X_2}(x) = f_{X_1}(x) * f_{X_2}(x)$$

$$min : (X_1, X_2, ..., X_N) = 1 - \prod_{i=0}^{N} \bar{F}_X(i)$$

(6.4)

In the case of summation of RVs, this is a convolution operation of the PDFs involved and a division can be accomplished by downsampling the result and adjusting for area under the curve. To find the minimum amongst a set of random variables, it is the remaining probability from the case where every transistor is that value or greater which from Equation 3.7 is the Complementary CDF (CCDF). Thus the procedure in a discretized calculation for creating the gate delay from transistor PDF involves convolving the transistor stack if there is one, converting all current source’s PDFs to CCDFs by performing cumulative sums, performing a piecewise multiply of all CCDFs and subtracting from 1, then taking the gradient of the result to form the new PDF. The results are presented in Figure 6.2.

![Figure 6.2: Probability of Gate Speed from Nominal Transistor Value](image)

Figure 6.2: Probability of Gate Speed from Nominal Transistor Value
To interpret the y-axis of the figure, recall that the area under the curve of a PDF must be 1 from Equation 3.7, and since we have normalized the speed, the probability normalizes itself in order to maintain this property. Also recall that the y-axis is the instantaneous probability which is why it can be above 1. Therefore as the range narrows especially below 1, the higher the instantaneous probability will go.

6.2.4 Processor Level

For the processor level, we now need to construct a proxy path that will stand for a typical critical path. We decide to simply use 2 input NAND gates for every level of logic since it has the highest expected value of the logic gates. So now we must estimate timing for the processor and again following the alpha-power law, we simply sum each gate delay which for RVs is the convolution of their PDFs. After convolving and normalizing, the last thing to do is to estimate the number of potential critical proxy paths a prospective architecture would have. From those paths, all having the PDF of the convolved gates, we find the minimum in the same method as the gate current source. Figure 6.3 shows the results, assuming different depths of logic and using 10,000 critical proxy paths similar to Varius [112].

As one can see, the increase in logic depth has a substantial impact on timing capability lost to random variance. Likewise, we also see a substantial spread in timing variability with decreased logic depth, both observations being easily explained by the central limit theorem. However, this analysis should not be mistaken that we are discussing loss of timing capability compared to the nominal capability. Indeed a 10 deep logic design would operate 3 times faster than a 30 deep logic design in the nominal case, but variance will have a much larger impact on both reaching nominal frequency and also the spread of possible frequency capabilities of the 10 deep logic design which can be very detrimental in the case of making homogeneous systems even for standard processor design. This observation has been made by Varius [112] as well, including a similar analysis in superthreshold operation only.
6.2.5 Bad Case Scenario

For completeness, we also include the following figures for a 45 $\sigma mV$ scenario.

Figure 6.3: Probability of Processor Speed from Nominal
Figure 6.4: Bad Case Probability of Transistor Speed from Nominal

Figure 6.5: Bad Case Probability of Gate Speed from Nominal
Figure 6.6: Bad Case Probability of Processor Speed from Nominal

The sensitivity to random variance is extremely high in the NTV case with a 16% loss in performance between the 30 and 45 $\sigma mV$ cases for 10 depth logic. While noone would rightly design that shallow a depth, it shows a clear point: Random variance is one of the most critical parameters for lithography nodes to control if 7nm and beyond technologies are to be reasonable for manufacturing energy efficient processors especially at near threshold.

### 6.3 Interprocessor Timing

So far we have looked at the timing within the pipelines of a processor, but now we must address the systemic variation that will impact between processors on the same chip. This model will similarly use the alpha-power law and transregional transistor models to provide analytical solutions to predict processor frequency spread in NTV.
6.3.1 Developing the Model

Similar to the intraprocessor timing model, we model the timing of the processor as a critical path of gates where the timing of the processor is proportional to the sum of the gate delays. Again we apply the EKV model from Equation 3.1 to both gate and processor timing assuming the same target threshold voltages for all transistors and now look at all the factors that affect threshold voltage found in Equation 3.6.

\[
\tau_{\text{gate}} \propto \frac{C_{\text{gate}} V_{DD}}{V_T^2 \ln^2 \left( e^{\frac{V_{DD} - V_{Th}}{2mV_T}} + 1 \right)}
\]

\[
\tau_{\text{proc}} \propto \sum_{i=0}^{\text{logic depth}} C_{\text{gate}}(i) \cdot \frac{V_{DD}}{V_T^2 \ln^2 \left( e^{\frac{V_{DD} - V_{Th}}{2mV_T}} + 1 \right)}
\]

\[
\tau_{\text{proc}} = \frac{C_{\text{Rand}}(V_{DD}) \cdot C_{\text{Eff}} \cdot V_{DD}}{V_T^2 \ln^2 \left( e^{\frac{V_{DD} - V_{Th}}{2mV_T}} + 1 \right)}
\]

\[
f_{\text{freq}}(X) = \frac{V_T^2 \ln^2 \left( e^{\frac{V_{DD} - V_{Th}}{2mV_T}} + 1 \right)}{C_{\text{Rand}}(V_{DD}) \cdot C_{\text{Eff}} \cdot V_{DD}}
\]

where \( V_{Th} = X + -\delta V_{DD} + k(T - T_0) + V_{Th_0} \)

and \( X = \text{RV of Threshold Voltage} \)

Line 1 applies the alpha-power law using the EKV interpolation. Line 2 describes the critical path as the summation of all the gate delays along the path. Line 3 instantiates a variable, \( C_{\text{Eff}} \), which is the effective capacitance equivalent for all the gates on that critical path. Lastly, we define the frequency of the processor as a function of X. While we’ve swept many additional variables under the rug, because these variables are not dependent on voltage or temperature, we can place all variables into the effective capacitance variable \( C_{\text{Eff}} \) that serves to capture all the capacitances of the gates in the critical path. Likewise, \( C_{\text{Rand}} \) is the variable that accounts for the loss in performance due to random variance which we just calculated and would need to be estimated numerically for any voltage the simulator would wish to run. To utilize this approximation across a chip, we simply utilize the threshold voltage mapping in
Chapter 5 to generate a specific threshold voltage for each processor, thus giving each processor the ability to determine the maximum frequency it could attain.

### 6.3.2 Alternative Transistor Model for NTV

The EKV continuous timing model is very robust but if we are only interested in the NTV operating region, we can leverage the transregional model presented in Equation 3.5. In particular, we will use this model for an analytical derivation so that we can get the expected frequency if every processor were allowed to operate at its fastest possible speed rather than be clocked to a common frequency. We will model the threshold voltage as a normal random variable, use the transregional model as our function, and find the expected variable using the method in Equation 3.12. This derivation is shown in Equation 6.6.

\[
\text{freq}_{\text{proc}}(V_{Th}) = \frac{mV_T^2 \cdot e^{k_1 \frac{V_{DD} - V_{Th}}{mV_T} + k_2 \left( \frac{V_{DD} - V_{Th}}{mV_T} \right)^2}}{C_{\text{Rand}}(V_{DD}) \cdot C_{Eff} \cdot V_{DD}}
\]

\[
E[X] = \int_{-\infty}^{\infty} \frac{mV_T^2 \cdot e^{k_1 \frac{V_{DD} - x}{mV_T} + k_2 \left( \frac{V_{DD} - x}{mV_T} \right)^2}}{C_{\text{Rand}}(V_{DD}) \cdot C_{Eff} \cdot V_{DD}} \cdot \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \, dx
\]

\[
E[X] = \frac{mV_T^2}{C_{\text{Rand}}(V_{DD}) \cdot C_{Eff} \sigma\sqrt{2\pi V_{DD}}} \int_{-\infty}^{\infty} e^{k_1 \frac{V_{DD} - x}{mV_T} + k_2 \left( \frac{V_{DD} - x}{mV_T} \right)^2} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \, dx
\]

First, we replace the transregional transistor model from Section 3.1.3 into the previously derived timing model. We then evaluate the expected value of the processor frequency given a normal variation of the threshold voltage. In line 3, we move invariance terms outside of the intergral and define a new variable, \(Z\), for convenience.

\[
E[X] = \frac{Z}{\sigma\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{k_1 \frac{V_{DD} - x}{mV_T} + k_2 \left( \frac{V_{DD} - x}{mV_T} \right)^2} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \, dx
\]

\[
E[X] = \frac{Z}{\sigma\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{e^{-2\sigma^2} \frac{k_2 \left( \frac{V_{DD} - x}{mV_T} \right)^2}{2 \sigma^2} \cdot e^{\frac{-2k_2 \sigma - mV_T k_1 \sigma + \mu(mV_T)^2}{\sigma^2 (mV_T)^2} x}} \, dx
\]

\[
E[X] = \frac{Z}{\sigma\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{e^{-2\sigma^2} \frac{k_2 \left( \frac{V_{DD} - x}{mV_T} \right)^2}{2 \sigma^2} \cdot e^{\frac{-2k_2 \sigma - mV_T k_1 \sigma + \mu(mV_T)^2}{\sigma^2 (mV_T)^2} x}} \, dx
\]
Next, we integrate the probability exponential with the transregional model function. We expand all terms and then rearrange them into the coefficients for the polynomials of x.

\[
\int_{-\infty}^{\infty} e^{ax^2+bx+c} \, dx \to \frac{\sqrt{\pi}}{\sqrt{-a}} e^{c-\frac{b^2}{4a}}
\]

\[
a = \frac{-(mV_T)^2 + 2k_2\sigma^2}{2\sigma^2(mV_T)^2}
\]

\[
b = \frac{-2k_2\sigma - mV_Tk_1\sigma + \mu(mV_T)^2}{\sigma(mV_T)^2}
\]

\[
c = \frac{2k_1\sigma^2V_{DD}mV_T + 2k_2\sigma^2V_{DD}^2 - \mu^2(mV_T)^2}{2(mV_T)^2\sigma^2}
\]

\[E[X] = \frac{Z}{\sigma\sqrt{2\pi}} \frac{\sigma\sqrt{\frac{2\pi}{\sigma}(mV_T)}}{\sqrt{(mV_T)^2 - 2k_2\sigma^2}} e^{c-\frac{b^2}{4a}}
\]

\[E[X] = \frac{m^2V_T^3}{C_{Rand}(V_{DD}) \cdot C_{Eff} \cdot V_{DD} \cdot \sqrt{(mV_T)^2 - 2k_2\sigma^2}} e^{c-\frac{b^2}{4a}}
\]

We then reference the general solution found in Appendix B and reprinted in the derivation. We then resubstitute Z to find the final approximation for the expected frequency of the processor. While it’s possible to simplify the expected value by removing certain terms which are negligible, we keep the entire expression since it will be utilized within a script for the purpose of approximating the maximum throughput and energy consumption across a simulated chip.

6.3.3 Determining \(V_{Th}\) and \(C_{Eff}\) through Observation

The timing model is a function of 8 variables: \(C_{Rand}, C_{Eff}, V_{DD}, V_T, m, \delta, k, V_{Th0}\) of which 2 can be modified by the processor \((V_{DD}, V_T)\) through voltage regulation or temperature control. \(C_{Rand}\) can only be determined through an RTL based SSTA analysis so can really only be estimated. We will show a method to determine \(\delta\) in Chapter 7 which leaves 4 more variables that need to be either assumed or determined through empirical means. In every case, we will need to use a nonlinear solver to determine the variables, but if we can find observations that will not affect two of the 4 remaining variables, we can use a system of equations to derive them. For
example, leaving the temperature constant removes $k$ from modifying the threshold voltage leaving only $m$, $C_{Eff}$, and $V_{Th0}$ to be determined using 3 $V_{DD}$\Frequency operation points if the DIBL is known. Likewise, we can use only 2 operating points if the subthreshold slope, $m$, is assumed to determine $V_{Th}$ and $C_{Eff}$. Finally, leaving $V_{DD}$ constant and modifying the temperatures can give the $k$ constant once previous variables have been discovered.

6.3.4 Predicting Frequency Capabilities

Unlike superthreshold where capable frequencies are diminished with an increase in temperature, near threshold processors can increase frequency when temperatures increase [44]. This is due to the drop in threshold voltage caused by the negative temperature dependence of the threshold voltage. Likewise, some processors may not be able to meet a frequency if they are too cold. This makes the proposed EKV model very beneficial in determining if the processor would have timing errors due to loss of heat. This can be built into a simulator such that the runtime can develop proper methods for ensuring legal frequencies are set at all times. Likewise, we can determine the number of processors that will be able to reach that point by determining what the sigma of operation is and using the $\Phi$ function.

6.3.5 Validation

This work utilizes well proven high level approximation like the Alpha Power law model and modifies it for Near Threshold. This technique was similarly done in Varius-NTV [72]. In using the transregional model, this was validated by the author in [73]. Otherwise, it is very difficult to validate this model due to the lack of public data on processor timing variance in the near threshold.

6.4 Conclusion

We present a robust timing model that accounts for both random variance and spatial variance. This model can be used with varied temperatures and voltages and determine the maximum legal frequency a processor can attain in both near threshold
and super threshold. Additionally, we provide an analytical approximation for determining the expected frequency of all near threshold processors if they are operating in the near threshold. Using this model coupled with our energy model, it will be possible to do a chip level analysis.
Chapter 7

LEAKAGE MODEL

So far our models have addressed how timing is affected by near threshold voltage (NTV) designs, but recall that our entire motivation for utilizing NTV is as a technique to reduce energy consumption in the effort to maintain high performance levels of computing in a reasonable power envelope. While the upside of NTV is quadratic reduction in dynamic power, the downside is lower frequencies and sub-quadratic reduction in static energy, making leakage the dominant source of energy consumption at NTV. This means an accurate leakage model is a valuable tool at NTV. As we have reiterated, leakage is nonlinearly affected most by temperature and is highly sensitive to perturbations of $V_{th}$ making accurate estimates that much harder. In the past, architects and system designers could utilize simpler physical equations with nominal values due to higher $V_{th}$ and less variance however, as we will show in later chapters, doing so for NTV designs would grossly underestimate the leakage. To state this based on our mathematical model, the assumption that

$$E[P_{Leak}] \propto V_{DD} \ast I_{Leak}(E[V_{Th}])$$

fails with large amounts of variance and as such will produce very inaccurate processor and chip level leakage predictions. Just as in Chapter 6, the use of transistor models can help to develop high level approximations. For our leakage model, we will again do this but instead apply the subthreshold model from Section 3.1.2.

7.1 Intraprocessor Leakage

We first need to account for the different leakage transistor within a processor. From Equation 3.4 in our primer, we can see the current of a single transistor in the
subthreshold region. For convenience, we rewrite the subthreshold model in Equation 7.1.

\[ I_{D-S} = \mu \frac{W_{eff}}{L_{eff}} C_{ox} \cdot (m - 1) V_T^2 \cdot e^{\frac{|V_{G-S} - V_{th}|}{m \cdot V_T}} \cdot (1 - e^{-\frac{-|V_{D-S}|}{V_T}}) \]  

(7.1)

where \( V_{th} = \delta V_{D-S} + k_1 (T - T_0) + V_{Th0} \)

In Equation 7.1, \( \mu \) is the electron mobility, \( W_{eff} \) is the effective channel width, \( L_{eff} \) is the effective channel length, \( C_{ox} \) is the capacitance formed by the oxide layer of the gate, \( V_{G-S} \) is the voltage applied from gate to source, \( V_{th} \) is the threshold voltage or the voltage which switches the transistor between the “ON” and “OFF” modes, and \( V_{D-S} \) is the voltage applied across from Drain to Source. Now we wish to simplify this model, making reasonable assumptions about what state the leakage transistor will be in. First, we assume the gate is driven completely to ground so that \( |V_{G-S}| \) is 0. Likewise, the transistor is holding back the output which is at \( V_{DD} \) potential, and ignoring the possibility of stacked transistors, \( |V_{D-S}| \) will be \( V_{DD} \) which will easily be above \( 4V_T \) making the last term effectively 1. This reduces our leakage model as shown in Equation 7.2.

\[ I_{\text{Leak}} = \mu \frac{W_{eff}}{L_{eff}} C_{ox} \cdot (m - 1) V_T^2 \cdot e^{-\frac{(\delta V_{DD} + V_{Th0} + k_1 (T - T_0))}{m V_T}} \]

\[ I_{\text{Leak}} = \mu \frac{W_{eff}}{L_{eff}} C_{ox} \cdot (m - 1) V_T^2 \cdot e^{\frac{-k_1 T}{m}} \cdot e^{-\frac{-|V_{D-S}|}{V_T}} \cdot e^{\frac{-k_1 T}{m T}} \]  

(7.2)

Now that we’ve made the assumptions, we transform the subthreshold transistor model to a generalized leakage model for the transistor and develop our model from there.

7.1.1 Developing the Model

Now we wish to include variation into our methodology. For the leakage of the processor, recall from our timing model in Chapter 6 that random variance will be the
dominant form of variation within the processor. This means that we need to account for the impact random variance will have as temperature and voltage change for that specific processor. If we vary the threshold voltage to be a normal distribution, this means the resulting leakage for the transistor will be a lognormal random variable which we derived in Chapter 3 and for our model, we will make the leakage of the whole processor a lognormal random variable as well. It is a reasonable practice to approximate the sum of exponentials as would be the case in aggregating all the leakage of individual transistors in a processor as another exponential with different parameters. A common approximation method first published by L Fenton [47] for communication equally applies in this scenario. In the approximation, the expectation and variance terms are matched to the sum of variance and expectations. Thus, the variance of a transistor, \( \sigma_t^2 \), influences the aggregated lognormal model of a processor \( p \) using this equation:

\[
\sigma_p^2 = \log\left( \frac{\sum e^{2\mu_t+\sigma_t^2}(e^{\sigma_t^2} - 1)}{(\sum e^{\mu_t+\sigma_t^2/2})^2} + 1 \right)
\]

\[
\mu_p = \log\left( \sum e^{\mu_t+\sigma_t^2/2} \right) - \frac{\sigma_p^2}{2}
\]

(7.3)

In the case that all \( x_t \) transistors have the same variance parameter \( \sigma_t = \sigma \), these formulas simplify to

\[
\sigma_p^2 = \log\left( (e^{\sigma^2} - 1) \frac{\sum e^{2\mu_t}}{(\sum e^{\mu_t})^2} + 1 \right)
\]

\[
\mu_p = \log\left( \sum e^{\mu_t} \right) + \frac{\sigma^2}{2} - \frac{\sigma_p^2}{2}
\]

(7.4)

Given that the number of transistors will be on the order of 1 million transistors, \( \sigma_p \to 0 \) due to the law of large numbers while \( \mu_p \propto \mu_t + \frac{\sigma^2}{2} \). This means for our model, that we simply add the random variance of threshold voltage developed in Chapter 5 divided by the subthreshold slope and thermal voltage defined as \( \frac{\sigma_{mV}}{mV_T} \) as a square term to the original term that will represent our varied processor’s behavior. While we may not know all the variables for every transistor, leakage currents of all transistors do
no depend on each other (ignoring stacking) like in timing but are independent, and because of superposition, we can simply form a single variable that subsumes all these parameters that we can then find through observation or estimation. This variable will be \( \alpha \). Thus our equation for the leakage power for a single processor is:

\[
P_{\text{Leak}} = V_{DD} \cdot \alpha \cdot T^2 \cdot e^{\delta \cdot \frac{V_{DD} - \lambda}{2 \cdot T^2} + \frac{(Var_{Rand})^2}{2 \cdot T^2}}
\]

where \( \beta = \frac{\delta m k}{q} \), \( \lambda = \frac{V_{Th0} + k_1 \cdot T_0}{m k} \)

and \( Var_{Rand} = \frac{\sigma_{Rand} mV}{m k} \)

(7.5)

7.2 Interprocessor Leakage

Now we wish to develop the variation between processors which will mean we need to add systemic variances across a chip. In this case, we wish to approximate the expected leakage mostly in an effort to calibrate our fitting constants in our model, and then will instantiate each processor with a random variable that is derived from the variance mapping algorithm presented in Chapter 5.

7.2.1 Developing the Model

In the same way as we model the intraprocessor variance as a normal random variable of the threshold voltage resulting in a lognormal random variable, we perform the same method with the resulting model given in Equation 7.6.

\[
\text{ChipLeak} = V_{DD} \cdot \alpha \cdot T^2 \cdot e^{\beta \cdot \frac{V_{DD} - \lambda + X}{2 \cdot T^2} + \frac{(Var_{Rand})^2 + (Var_{Sys})^2}{2 \cdot T^2}}
\]

where \( \beta = \frac{\delta m k}{q} \), \( \lambda = \frac{V_{Th0} + k_1 \cdot T_0}{m k} \), \( Var_{Sys} = \frac{\sigma_{Sys} mV}{m k} \)

\( Var_{Rand} = \frac{\sigma_{Rand} mV}{m k} \) and \( X = N(0, \sigma_{Sys} mV \frac{q}{m \cdot k}) \)

(7.6)

Just as before, the chip leakage will be the expected value of a lognormal variable, but now \( \sigma^2 = \frac{(Var_{Rand})^2 + (Var_{Sys})^2}{T^2} \). Likewise, the processor leakage has an expected
value the same as before but now we will vary the $\lambda$ variable using a normal random variable, $X$ whose sigma is that of the systemic variance. For the purposes of probability, we can use this to graph PDFs, but in simulation we would feed each processor its correlating variance from the previous mapping work in Chapter 5.

### 7.2.2 Acquiring the Coefficients through Observation

In Chapter 6, we assumed that leakage information would be able to provide the $\beta$ and thus the effective DIBL once the subthreshold slope is determined. Similarly, once the effective DIBL is derived, then $V_{Th}$ and the thermal dependence constant can be derived from the timing model. For operating points with the same temperature but differing $V_{DD}$, we can use the ratios of the two point’s leakage current derived using Equation 7.2 to solve for $\beta$

\[
\frac{I_1}{I_2} = \frac{\alpha T^2 e^{\beta V_{DD1} - \lambda}}{\alpha T^2 e^{\beta V_{DD2} - \lambda}}
\]

\[
\ln\left(\frac{I_1}{I_2}\right) = \frac{(\beta V_{DD1} - \lambda) - (\beta V_{DD2} - \lambda)}{T}
\]

(7.7)

\[
\frac{T \ln\left(\frac{I_1}{I_2}\right)}{V_{DD1} - V_{DD2}} = \beta
\]

### 7.3 Validation

The use of the subthreshold current equation is not particularly novel as it has been used extensively in superthreshold design and analysis [60, 88, 58]. Furthermore, these works have validated both timing and leakage models on SPICE simulations and empirical experiments. However, the accounting of the systemic and random variance is novel at the system level, but has been proposed for macroscopic effects of transistor length variability in circuit design [97]. It would be very difficult to validate this model without specific foundry information since it relies heavily on a variance model.
7.4 Conclusion

We’ve provided a method to estimate leakage very accurately as a function of voltage and temperature that takes into account all forms of variance. The solution is straightforward and statistically sound, being very accurate for functional simulation and runtime development when paired with the proper variance, temperature, and timing information.
Chapter 8
TEMPERATURE MODELING

Temperature Modeling is usually considered the low level simulation of chip design so that RTL level simulations can be accurately made. This dissertation wishes to make temperature modeling a necessary component of even runtime system development, since it can have a huge impact on the leakage and energy efficiency which would thus impact the runtime’s effectiveness at making proper decisions.

8.1 Laws of Physics

Thermal conduction is the macroscopic affect of many small particles in excitation and the diffusion of this excitation to surrounding molecules. The true conduction of heat and its relation to temperature can be described using the partial differential equation of the Heat Equation:

\[
\frac{\partial u}{\partial t} - \alpha \nabla^2 u = 0 \tag{8.1}
\]

where \( u \) is the temperature and \( \alpha \) is the thermal diffusivity of the material. While closed form solutions have been derived for specific geometries such as an infinitely thin metal rod famously analyzed by Fourier\cite{49}, typical computational methods use Finite Element Methods (FEM) where geometry is subdivided usually along material boundaries or in uniform voxels depending on time fidelity requirements and the conductivity of the material. Programs like ANSYS \cite{80} are employed by chip and circuit designers but would require tremendous memory and computation to very accurately calculate temperatures across a chip. While this accuracy is critical for low level circuits, approximate temperature will work for the purposes of general processor
behavior such that a runtime can be developed. Because of this, the Newton model of heat transfer is performed.

8.2 Lumped Capacitance Model

As an approximation, the heat equation is modeled instead as lumps of material with interfaces in between. From a computational point of view, this is exactly how a Finite Difference or Finite Element calculation would work, however the coarseness of the material volume is much more than could accurately calculate the full heat equation. Instead, the thermal model is lumped into elements and made into an electrical circuit with correlating properties in Table 8.1. See Figure 8.1 for a 4 node example of a thermal simulation.

8.3 Transient Solution

After linearizing the circuit so that only resistors and current sources remain, one can apply Kirchoff’s law to establish a linear system of equations. Once the system is backsolved, transient temperatures are then calculated using numerical methods for initial value problems like backward Euler or Runge-Kutta [77]. For state of the art programs like Hotspot[118], this is the exact procedure. After a predetermined number of simulation cycles, all circuit elements report their energy consumption for that period and their previous temperature. The equation is backsolved and then new temperatures are distributed to elements for continued simulation. This creates a fork-join style of program control-flow which can be very taxing for parallel discrete event simulators.

<table>
<thead>
<tr>
<th>Electrical</th>
<th>Thermal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voltage (V)</td>
<td>Temperature (T)</td>
</tr>
<tr>
<td>Current (A)</td>
<td>Heat Flow (W)</td>
</tr>
<tr>
<td>Resistance (Ω)</td>
<td>Thermal Resistance (W/K)</td>
</tr>
<tr>
<td>Capacitance (F)</td>
<td>Thermal Capacitance (J/K)</td>
</tr>
</tbody>
</table>

Table 8.1: The Duality of Conductions
Figure 8.1: Example of a 4 Lump Thermal Circuit
8.4 Proposed Method

To remove this fork-join synchronization, we propose an asynchronous method of transient calculation whereby the temperature of surrounding neighbors is static as can be seen in Figure 8.2. While it’s not entirely the case, temperatures in bulk silicon are expected to move very slowly so this assumption is not unreasonable. Just as is the method for deriving the heat equation, the proposed method will obey the law of conservation of energy by determining heat transfers and then atomically transferring the heat to a remote process. Because of the atomicity requirement and the fact that heat transfer is calculated from two different points of view, there must be a predetermined actor to do the accounting. For our convention, we will only ever “push“ heat energy from a warm processor to a cooler one, and anytime heat is pushed to a remote processor, the new temperature of that processor will be returned by the packet handler. Additionally, temperatures will be broadcasted to neighbors periodically to ensure correct data is present when transient calculations are made.

8.5 Known Limitations

As mentioned previously, heat transfer is highly dependent on material properties. It has been well studied that Silicon on Insulator (SOI) lithography breaks the assumption of lumping areas of silicon together since SOI uses silicon dioxide layers within transistor construction[136]. It is also worth mentioning the work of PowerBlurring[137] which uses a high fidelity simulation like ANSYS to determine the impulse responses or green function for a hardware configuration. Similar to image blurring, PowerBlurring will convolve the heat propagation and create very efficient thermal results at much lower costs, however, we chose not to implement this because an asynchronous method was non-obvious, the use of high precision software to calibrate the system was required, and temperature calculations were not critical to be perfectly accurate for our runtime system simulation.
Figure 8.2: Proposed Method for Thermal Simulation
8.6 Conclusion

The dissertation has proposed a model that breaks the synchronous and thus barrier laden approximations used to currently measure temperature at chip level simulations. It has been implemented in the SAFE framework and is capable of providing the needed functionality for other models to interact with it.
Chapter 9

LEAKAGE ANALYSIS FOR AN NTV PROCESSOR

Having gone through quite a bit of explanation about the model and how one can attain the parameters through observation, it would be very useful to see a concrete example. In this case we will examine the proposed architecture of Traleika Glacier (TG), a NTV proposal first envisioned during the DARPA UHPC [4] program and continued in the DOE Xstack program [5]. In particular, we look at the TG strawman specified by Intel for 7nm with these performance characteristics and an architecture sketched out in Figure 9.1:

<table>
<thead>
<tr>
<th>Technology</th>
<th>7nm, 2018</th>
</tr>
</thead>
<tbody>
<tr>
<td>Die Area</td>
<td>500 $mm^2$</td>
</tr>
<tr>
<td>Processors/Die</td>
<td>2048</td>
</tr>
<tr>
<td>Power</td>
<td>600 Watts @ $V_{DD}$</td>
</tr>
<tr>
<td></td>
<td>37 Watts @ $\frac{1}{2}V_{DD}$</td>
</tr>
<tr>
<td>Frequency</td>
<td>4.2 GHz @ $V_{DD}$</td>
</tr>
<tr>
<td></td>
<td>600Mhz @ $\frac{1}{2}V_{DD}$</td>
</tr>
<tr>
<td>Energy</td>
<td>34 pJ/F @ $V_{DD}$</td>
</tr>
<tr>
<td>Efficiency</td>
<td>15 pJ/F @ $\frac{1}{2}V_{DD}$</td>
</tr>
</tbody>
</table>

Table 9.1: TG Strawman Specifications

From here we make a series of assumptions in order to reverse engineer the parameters of a TG chip. The first assumption is that these operating points use the same temperature and that the temperature is within expected power dissipation for the full $V_{DD}$ operating point. For temperature, we assume steady state chip temperature of 125°C at 600W. We also assume $V_{DD}$ is 1 which is a reasonable estimate for expected scaling. Now we must determine reasonable leakage power for each operating point by
Figure 9.1: Organization of the Traleika Glacier Architecture
removing the dynamic power from the total using equation 9.1 from page 98. We look at possible values of dynamic capacitance and the impact that has on the leakage.

\[ P_{\text{Dyn}} = C_{\text{Dyn}} \times V_{\text{DD}}^2 \times F \]  

(9.1)

**Figure 9.2:** Finding a Reasonable Dynamic Capacitance

In figure 9.2, the values centered around 50 pF per processor show reasonable leakage values of 60% for NTV operation and 30% for STV operation. We choose 48pF for the rest of the analysis knowing it is an approximation. Using Equation 9.1, this results in leakage operating conditions indicated in Table 9.2.

<table>
<thead>
<tr>
<th>Voltage (V)</th>
<th>Temperature (C)</th>
<th>Expected Power (mW)</th>
<th>Clock Period (ps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>.5</td>
<td>125</td>
<td>10.8</td>
<td>1667</td>
</tr>
<tr>
<td>1</td>
<td>125</td>
<td>91.4</td>
<td>238</td>
</tr>
</tbody>
</table>

**Table 9.2:** Assumed TG Operating Conditions
9.1 Variance Analysis for Traleika Glacier Strawman

Having established rough operating conditions, we can now look at the effect variance has on the distribution of processor leakage. Figure 9.3 below shows multiple solutions to the operating conditions in Table 9.2 for a range of threshold variations with the probability of of a processor state on the vertical axis.

![Graph showing 2 Sigma Variance of a NTV Processor](image)

**Figure 9.3:** Solutions with Different Variances

Here we see the large error produced between the median processor leakage and the average processor leakage as variation increases. Actual chip level leakage per processor can be as much as 2 times higher than the median processor leakage when systemic threshold variation is 15%. We can also see the nonlinear effect that threshold variance has on processor leakage variance. While the graph is cut off, using the 2 sigma limit which would sample 95% of the chip, the processor with the most leakage can have 6x, 30x, and 115x more leakage than the processor with the least leakage for threshold variances of 5%, 10%, , and 15%, respectively.
9.2 Temperature Analysis for Traleika Glacier Strawman

We now hold variance at 10% and look at the temperature variances and ranges for our model. The case of \( Var = 10\% \) produces the coefficients in Table 9.3

| \( \alpha \) | 85.97863731 |
| \( \beta \) | 1148         |
| \( \lambda \) | -6181       |

\[
\alpha = 85.97863731, \quad \beta = 1148, \quad \lambda = -6181
\]

\[
99.5 \frac{mV}{V_{BSW}} DIBL, \quad 535 \frac{mV}{V_{th}} BodySwing
\]

Table 9.3: Resulting Coefficients for Conditions in Table 9.2 with 10% Variance

![Figure 9.4](image)

**Figure 9.4:** Leakage per Processor as a Function of Temperature

The points which were used as operating conditions are marked, showing that expected chip level leakage is satisfied by the operating condition, but that a wide range of leakage curves exist as the threshold voltage varies. This implies that if large spatial correlation exists, processors could vary energy consumption by more than 100%. The wide divergence in leakage also means that thermal control of any NTV processor will be difficult at best as hot spots will form around leakage prone areas.
9.3 Isothermal Energy Analysis

The last piece of initial data analysis will be based around bringing the variance back to the initial goals of the proposal: to schedule an NTV processor in the most efficient way possible. Here we look at a scenario where a single frequency is the only choice available for each VDD, and the impact that has on energy efficiency per operation. Here we use leakage distribution information similarly shown in section 9.1 from the 10% variance scenario, divide by the frequency, and add back the dynamic energy. Figure 9.5 shows the distribution of energy per instruction and impact temperature has on these distributions. The solid line indicates the expected value of the energy per operation and the dashed line indicates the median processor’s energy efficiency.

Figure 9.5: Energy per Operation for Different Temperatures in a 10% Variance Scenario
From Figure 9.5, we can make many observations. The first observation is one that is quite intuitive: processors run much better when cool. The variance range is much higher as probabilities smear out over a large range of possible efficiencies. This means that the importance of proper scheduling is much paramount at high temperature, leading to the conclusion that scheduling algorithms should be temperature dependent. There should be a quick and low overhead scheduler used when the chip is cool, and a thorough scheduling decision made when temperatures are high. Next, it is clear to see at higher temperatures that there are energy efficient processors that would have a lower energy per instruction than some of the less efficient processors at a lower VDD. This advocates for multiple frequencies being available even if this complicates frequency domain translation.

9.4 Conclusion

The dissertation has leveraged the leakage variance model to give significant insight into behaviors we expect to see in future manycores and the impact variance may have depending on how severe it is. We have derived the basic coefficients necessary to perform the analysis using a series of reasonable assumptions and methodologies laid out in previous chapters. The analysis also makes clear the paramount need for an intelligent scheduler that can both manage temperature both locally and throughout a chip.
Chapter 10

ENERGY AVOIDING MATRIX MULTIPLY: DATA MOVEMENT ALGORITHMS FOR NTV

Having provided an extensive set of tools for solving the optimization of computing resources, our final chapter will seek to provide initial insight into new methods for reducing the data movement energy in near threshold voltage designs. Starting with GPUs and continuing with future many-core architectures, the trend of placing large amounts of compute capacity on a single chip has created a phenomenon this paper describes as memory hierarchy capacity per capita inversion. To overcome this inversion, we present a dynamic tiling scheme which we apply to solve the classic Matrix Multiply algorithm. The scheme quickly determines a near-optimal tiling with minimal data movement energy while still allowing for slack and variance within the computation and memory of a chip. Using Hilbert curves for inspiration, this tiling scheme is a highly dynamic and adaptive aggregation algorithm. Unlike many programming paradigms, it doesn’t require memory that is symmetric in size or addressing nor does it require the same compute capability or throughput. This is intentional in the algorithm design to make it very robust to chip variance and allows all possible resources to be utilized which is necessary for future near threshold voltage designs. We show initial results on an future many-core simulator, FSim, and give initial estimates of memory reads and writes to all parts of the chip.

10.1 Traditional Tiling on Traditional Architectures Is Not Tenable for Exascale Architectures

Matrix Multiply (MM) has been studied for decades. Early works presented algorithmic improvements for asymptotic reduction of operations of MM to $O(N^{\log_2(7)})$ by
trading multiplications for simpler addition and applying recursively [120]. More recent work has looked at communication avoidance by seeking to minimize bytes read per floating point operation and attempting to reach the known lower bound which can provide more locality and less communication [39, 69, 18, 64]. Other previous works have taken the traditional algorithm and looked within the context of architecture and memory subsystems.

Projects like ATLAS [127] looked to empirically determine capacity, cache line size, and alignment so that optimal tiling is created for each level and this produced excellent results. As multicore solutions evolved, these solutions and others [15, 37] evolved to better leverage parallelism and solve problems that arise from shared cache structures. Traditionally, lower level data in a cache required replication to higher levels of caches. While we see efforts to advance the efficiency of complex cache hierarchies to loosen this constraint [67] and thus fit larger working sets in smaller equivalent chip space, the principle of having larger cache capacity at levels farther from the processor is still true today. However, we see a shift for future architectures starting with GPUs.

We are targeting the Traleika Glacier (TG) architecture, a prototype design chip for exploring Near Threshold Voltage (NTV) computing and an extension of the Runnemede many-core processor architecture [30]. TG features highly hierarchical architectures: execution engines are grouped into blocks; blocks are grouped into units; and units are grouped under a single chip as shown in Figure 10.1.
The sizes of memory are very unconventional as well. Figure 10.2 compares the memory hierarchy of a CPU, the 10 core Intel Xeon Processor E5-2470 v2 with a GPU, the NVidia Tesla K80, and a prototype design [30]. As the figure illustrates, for chip designs with dense amounts of compute, the higher level memory would occupy far too much area on die and thus is reduced. This reduction creates a memory capacity per capita inversion (CPCI) for the levels of the memory hierarchy. This reduction places a heavy burden on cache replacement policies or shared memory use in GPUs or cache thrashing may occur at these higher levels. This trend is expected to continue for many-core designs including TG. To provide an interesting experimental platform, the architecture is considering support for configuring all levels of memory as scratchpad...
or potentially as incoherent cache [62]. While these memory capacities might vary in the final design of TG, it is certain CPCI will exist. Our solution for TG similarly follows how GPUs, utilizing programmer managed shared memory, perform MM by exclusively storing the results in lower levels of memory permanently thus ensuring the higher level cache is only utilized for read-only accesses of $A$ and $B$ [71, 123, 82]. Additionally, as seen in the TG architecture, this trend can extend to not just a single shared memory, but instead every level of programmer controlled shared memory in a CPCI hierarchy. This opens many possibilities for unique and interesting techniques for utilizing this space including tiling which this paper will leverage.

In Section 10.2, the paper will explain the basic algorithm of Matrix Multiply and give an intuition to the trade offs in tiling techniques. The paper further extends tiling, specifically looking at tiling for energy efficiency in Section 10.3. Section 10.4 introduces a novel method for dynamically generating tile shapes using a hilbert inspired ordering. Section 10.5 combines these two techniques to provide a methodology for creating a tiling scheme for any memory layout and explain how to use asynchronous tasks to build a robust MM algorithm. Section 10.6 provides specific details about our experimental testbed using the FSIM simulator and the results.
10.2 Background

10.2.0.0.1 Matrix Multiply

MM is a common low level algorithm called in numerical packages for computing more complex algorithms such as singular value decomposition or LU factorization. Additionally, it is also a common benchmark or the core routine of benchmarks used to test hardware due to its large reuse of data which can test memory and caching subsystems. It can be computed with a triple nested loop, making the asymptotic computational complexity $O(N^3)$. In this paper, MM is defined as $C_{M,N} = A_{M,K} \times B_{K,N}$, $A, B, C \in \mathbb{R}^2$, $M, N, K \in \mathbb{N}^*$. 

10.2.1 Tiling Principles

Looking at the typical MM loop nest and seeing no interloop dependencies, standard classical compilation techniques can be applied. However, let’s first look at the loop nests from a mathematical operation point of view. In general, matrix multiply can be viewed in 2 ways:

![Illustration of Different Tiling Products](image)

Figure 10.3: Illustration of Different Tiling Products

A matrix multiply can be viewed as either a dot product of cross products or a cross product of dot products or any nested combination therein as shown in Fig. 10.3. In terms of data reuse and locality, inner product ordering can provide $K$ reduction of memory operations for the $C$ matrix if 1 extra temporary space is given and is
synchronization free on $A$ and $B$ since any pair of $A$ and $B$ values will not be reused in that dot product, but requires exclusive ownership of the $C$ value on which it reduces. However, $A$ and $B$ matrices are still accessed $MNK$ times. On the other hand, outer product ordering can provide $M$ and $N$ reduction of memory operations for the $A$ and $B$ matrices respectively but requires $M \cdot N + N + M$ extra temporary space to hold the cross product results and 2 input vectors and there is inherent synchronization requirements after each cross product since the temporary input space is reused $N$ times and cannot be overwritten until all products are finished. $A$ tiling scheme can be composed of any decomposition of inner products and outer products which will determine both reuse rate of data for each matrix as well as memory consumption for that tile. Traditionally, a new tile static in both size and shape will be used for each level of memory since more temporary space is available at farther memory levels and thus can provide more reuse of $A$, $B$, and $C$. In Section 10.5, this paper will introduce a novel hybrid method of distributing a tile amongst multiple levels of a CPCI hierarchy with a dynamic shape that can better utilize memory and reduce data movement.

10.3 Energy Efficient Tiling

As previously mentioned, outer product tiling is the only way to provide reuse of the $A$ and $B$ matrix at the expense of more temporary storage and strict synchronization, but what about energy efficiency? Energy consumed during computing can be divided up into energy to do compute and energy to move data. As we shrink lithography processes more, data movement and leakage will begin to dominate energy consumption [25]. Since leakage occurs regardless of executing tasks, an algorithm must keep all processors busy with little scheduling downtime. Thus, we also rely on asynchronous fine-grained scheduling in order to keep processors busy where synchronization is occurring, and double buffering to create slack in the synchronization. This is similar in style to Garcia et al. [53]. For reducing data movement, we propose a method to model the energy consumed by a tiling scheme to quickly determine a near-optimal tile size for a given amount of memory. This method creates a machine model
using a few assumptions:

1. Accessing data (read or write) from any kind of memory can be approximated as a particular static cost composed of dynamic access energy, leakage energy, and communication energy for both a farther memory and a closer memory.

2. This static cost is the average for all the values of that memory level regardless of variances in location, temperature, or circuit performance.

3. Accessing memory levels of neighbor processors will be on the order of the static cost of the closest common shared memory structure (i.e., shared memory structure is physically near all neighbors and the distance travelled dominates the static energy cost function).

Making these assumptions, we can express the total energy consumed for a subtiling in Eq. 10.1. Within the equation, we define $E$ as the static energy cost per access to either a higher memory (HM) or the lower memory (LM) in which we are tiling. For tiling dimensions, we define $A$ as having $M \times K$ and $B$ as $K \times N$ sizes in HM with a sub-tiling scheme in LM of outer product $m \times n$ and inner product $k$.

$$
HM_{Total} = 2MN \cdot E_{HM} + \left(\frac{NMK}{n} + \frac{NMK}{m}\right) \cdot E_{HM}
$$

$$
LM_{Total} = MN \cdot \frac{K}{k} \cdot (2 + 2k) \cdot mn \cdot E_{LM}
$$

$$
E_{Total} = HM_{Total} + LM_{Total}
$$

In the HM energy consumption, we see that every C result is read and written once because of the inner product ordering of the tile, and we see the reduction of $m$ and $n$ access for the $A$ and $B$ tiles by using outer product ordering of the smaller subtile. These reductions require increases in access to LM. First, a subtile must read in a partial sum from the LM subtile, then read $k$ values from $A$ input buffer and $k$ values from the $B$ input buffer, perform $k$ computes, and finally write back the partial sum to the result subtile. This operation is performed for the $mn$ values for each result tile every $\frac{K}{k}$ synchronization points at the energy cost of the lower memory (LM). Then the final results are written back out to HM, and the procedure will be repeated for the $\frac{MN}{mn}$ number of result tiles needed to complete the matrix in HM. Now to optimize
the energy consumed by data movement we make several changes of variables and a memory constraint. Let \( R = \frac{E_{HM}}{E_{LM}} \) define the ratio of energy consumption from higher to lower memory, and let \( S = \frac{m}{n} \) define the ratio of the longest side to the shortest side of the subtile (for this derivation, we assume \( m \) is longer). When \( S = 1 \), the tile is square, and as the tile becomes more rectangular, the squareness factor increases. Eq. 10.1 can then be simplified to Eq. 10.2.

\[
HM_{Total} = 2MNK \cdot E_{LM} + \frac{(1 + S)MNK}{Sn} R \cdot E_{LM}
\]

\[
LM_{Total} = \frac{MN}{Sn^2} \cdot \frac{K}{k} (2 + 2k) Sn^2 \cdot E_{LM} = \frac{MNK}{k} (2 + 2k) \cdot E_{LM}
\]

\[
E_{Total} = \left( MNK \cdot \left( \frac{2}{k^2} + 2 \right) + \frac{(1 + S)MNK}{Sn} \cdot R + 2MNK \cdot E_{LM} \right)
\]

Next, we make a memory constraint and thus define \( Q \) as the quantity of memory available for tiling in LM. We also will constrain our equation to a tiling scheme which will double buffer the \( A \) and \( B \) input vectors in order to loosen synchronization requirements which results in a memory constraint definition in Eq. 10.3.

\[
Q = Sn^2 + (1 + S) \cdot 2kn \quad \rightarrow \quad k = \frac{Q - Sn^2}{(1 + S) \cdot 2n}
\]

Substituting \( k \) in our original expression and again simplifying, we derive the total energy consumed as a function of higher tile dimensions, ratios, quantity of memory, and a single variable \( n \) to define the subtiling in Eq. 10.4.

\[
E_{Total} = \left( MNK \left( \frac{2(1 + S)2n}{Q - Sn^2} + 2 \right) + \frac{(1 + S)MNK}{Sn} (R) + 2MNK \cdot E_{LM} \right)
\]

\[
E_{Total} = (1 + S) \cdot \frac{4Sn^2 + (Q - Sn^2) \cdot R}{(Q - Sn^2) \cdot Sn} \cdot MNK \cdot E_{LM} + (2MNK + 2MNK) \cdot E_{LM}
\]
And lastly to find the minimum energy, we differentiate and set to 0 in Eq. 10.5. Solving the quadratic for \( n^2 \) and simplifying we obtain our final equation, Eq. 10.6.

\[
\frac{dE_{\text{Total}}}{dn} = -\frac{(1 + S) \cdot (Q^2 R - 2QS \cdot (R + 2)n^2 + n^4 \cdot (R - 4) \cdot S^2)}{Sn^2 \cdot (Q - Sn^2)^2} \cdot MNK \cdot E_{LM} = 0
\]

(10.5)

\[
Sn^2 = Q \left( \frac{(R + 2) - \sqrt{8R + 4}}{R - 4} \right)
\]

\[
FF = \begin{cases} 
\frac{(R+2)-\sqrt{8R+4}}{R-4} & R \neq 4 \\
\frac{1}{3} & R = 4 
\end{cases}
\]

(10.6)

As one can see, the proper amount of memory that should be dedicated to the outer product result tile is a function of the energy access ratio between higher and lower memories regardless of the shape of the tile. This function we call the fill factor and is designated as \( FF \) in Eq. 10.6. It is important to understand that this model is based on the three assumptions where the energy is static, which is not necessarily true. Where the inner product length is extremely short, there will be potential startup overheads that are not amortized such that the energy factor does not properly relate to the real energy cost. Similarly, in the case where the inner product is very long due to a low fill factor, the bandwidth requirements will increase to the high memory which typically require more energy per operation when accessed at higher bandwidths. Thus, this model should only be utilized as a first order approximation strategy for an overall tiling scheme.

Other limits and checks should be imposed as well to ensure this is the optimal tiling. One such requirement is that \( k \geq 1 \) and \( n \geq 1 \) can be violated by the non-discrete fill factor calculation and by using Eq. 10.3 and Eq. 10.6, additional constraints to the tiling shown in Eq. 10.7 can be added to make sure enough memory is available.

\[
1 \leq \frac{Q - Sn^2}{(1 + S) \cdot 2n} \quad \Rightarrow \quad Q \geq \frac{4(S + 1)^2 FF}{S(FF - 1)^2}
\]

(10.7)
Lastly, in our previous derivations we defined \( S = \frac{m}{n} \) where \( m \) and \( n \) were sides of a full rectangle tile. Now we define \( S' \) as the \( S \) equivalent of work to inputs for a full tile but \( S' \) is not perfectly filled. To do this, we match the outer product work of the partial tile and the input requirements of the partial tile to determine what the full tile equivalent would be and simplify using the quadratic equation to derive Eq. 10.8.

\[
W_{\text{ork}} = Sn^2 \\
(1 + S)n = Inputs \tag{10.8}
\]

\[
S' = \frac{Inputs^2 - 2W_{\text{ork}} + Inputs\sqrt{Inputs^2 - 4W_{\text{ork}}}}{2W_{\text{ork}}}
\]

We will see in future sections how these constraints and \( S' \) can be applied to coarsen tiles and thus lower runtime overhead, and still ensure that sufficient memory is left for input buffers.

### 10.4 Hilbert Inspired Global Layout

![Figure 10.4: Order 1 through 3 Hilbert and Morton Curves](image)

So far this paper has discussed hierarchical tiling and mathematical formulas for assigning the proper amounts of \( A, B, \) and \( C \) tiles in each memory level in the abstract
sense. However, we need an automatic method for aggregating tiles which creates a tile shape that has the least projected surface area for both dimensions (thus a low $S'$), can adapt to any memory layout, and is robust for any problem dimension. This makes a space filling curve an excellent candidate since these curves map a higher order space into a one dimensional space perfect for linearly enumerating as asynchronous tasks while also ensuring a good amount of locality. While some space-filling curves like Morton [96] curves are computationally very inexpensive, they are not contiguous and thus if used recursively could lead to large jumps within the matrix. A much better candidate would be a Peano [104] or Hilbert [55] curve which is what we propose to use. Fig. 10.4 provides examples of Hilbert and Morton curves.

Once the requirement to replicate data down a hierarchy in the manner of a cache is removed, the freedom to pin tiles anywhere in the hierarchy is possible. However, a strategy for the best layout is not obvious. We present a top down and then bottom up approach which utilizes as much memory as possible, is adaptive to different memory sizes, preserves locality even during dynamic throughput changes in processors, and is based on energy optimal tiling principles.
This produces tiles in certain memory locations in the method shown in Fig. 10.5. In order to achieve the properties described, the aggregations are not perfectly square or perfectly filled, which will incur some performance penalty that must be quantified.

### 10.4.1 Measuring $S'$ Empirically

![Figure 10.6: $S'$: Equivalent Full Tile Aspect Ratio matching Work:Inputs of a Partial Tile](image)

$^1$ These memory capacities are for illustrative purposes only.
Hungershofer and Wierum [61] show that for all sections of a Morton and a Hilbert curve, Hilbert curves have slightly lower average surface area to volume but also contain a higher worst case surface area to volume ratio.

Fig. 10.6 shows our calculations for worst case, average, and minimum $S'$ values for every possible aggregation for each tile size using a $1024 \times 1024$ Morton and Hilbert curves. For $S'$, the Hilbert curve outperforms Morton by a factor of four on average and has a bounded maximum below 8 whereas the Morton curve produces large maximum aspect ratios. This is because $S'$ is more related to projected surface areas than standard surface areas, giving an even larger penalty to Morton curves and making the choice of Hilbert inspired curves as the only reasonable choice.

10.4.2 Decomposition Rules for Layout

Figure 10.7: HIC Curve: an example $50 \times 80$ Hilbert Inspired Curve
Fig. 10.7 gives an example curve for any arbitrarily dimensioned problem which provides good locality for tiling which we will call HIC meaning Hilbert Inspired Curve. To accomplish this, we implement a pseudo-Hilbert curve algorithm very similar in style to Zhang et al. [135] and somewhat in the style of Chung et al. [34], which will be close to a Hilbert curve in $S'$ performance. Unlike Zhang et al. where divisions create splits with sections having power of 2 dimensions on the outer portions of the matrix, our algorithm makes simple division by 2 splits in each dimension until we reach a base case. While the Zhang technique generates more regular patterns at the expense of different aspect ratios throughout the matrix, our technique ensures a Hilbert order with as close to the overall aspect ratio of the matrix at the expense of a more complex base case ordering. In the base case where either tile dimension goes below 7, HIC terminates recursion and specifies every possible scanline order in a look up table similar to Zhang. We ensure that a split in the base case cannot result in two odd tiles by shifting the split as necessary. This reduces our look up table to 4 cases for each curve type, resulting in 64 total scanline orders.

![Figure 10.8: A Portion of the Base Case Scanline Rules Look Up Table](image)

Fig. 10.8 illustrates the scanline order for all cases of base tiles for a curve traversing from lower left to lower right. The other three curve orientations are not presented. Dots indicate start points for each scanline and the highlighted case does not have a contiguous end/start connection between the two upper subtiles as previously mentioned.

Lastly, as a precondition to create lower aspect ratio tiles, HIC will make scanlines of tiles in the style of Chung et al. rather than using Hilbert recursion. The
condition for this comes from dividing a rectangular tile into a more square tile which
occurs only until the longer length switches axes and is no longer smaller than the
current $S$ value. This creates the condition shown in Eq. 10.9.

\[ S > \frac{1}{\sqrt{2}} \rightarrow S > \sqrt{2} \quad (10.9) \]

Notice this has the equivalent effect of always dividing the longest dimension
of the tile similar to many cache oblivious algorithms except we only consider two
dimensions, and we only perform this single dimension split when the curvetypes allow
a chain of hilbert curves to connect like in the style of Chung. This is exactly what
we see in our example tile from Fig. 10.7 where the ratio of 8:5 is higher than $\sqrt{2}$ and
so a single dimension cut on the X axis is made in the middle shown in blue dashes.
This produces two tiles with an $S$ of 5:4 and so Hilbert recursion begins with the first
2 cuts shown in red dashes. To see the impact of these changes to allow arbitrary
matrix dimension instead of just the traditional Hilbert curve, several aspect ratios
were tested to ensure $S'$ values were still reasonable with results presented in Fig. 10.9.

![Figure 10.9: $S'$ for 3 Different Aspect Ratios](image)

While the maximum value has reached as high as 16 for smaller tile sizes, the
overall $S'$ values remain nearly the same as the original Hilbert curve, which will bode
well in Section 10.5 when utilizing this curve to aggregate tiles in a CIPC hierarchy
and coarsen the base tile dimensions.
10.5 Tiling Up and Down a Hierarchy Efficiently

The first step to the implementation of our algorithm is to query the runtime for all free memory at every level including memory that resides with processors that will not be computing for us. With this, we build a tree of all memory in the system where the smallest memory closest to the processor is a leaf and the memory that is shared between different groups of processor are inner nodes. The next task, which can be performed during the previous step, is to determine the base tile size and inner product length, $n$ and $k$ from Section 10.2.1. Of course, a tile size of 1 could work but the overhead of runtime queues and synchronization would be cost prohibitive. If we make the base tile size too coarse, then we will be unable to utilize smaller regions of memory, fragment pieces of memory during tiling, create excessive work stealing, and for small problem sizes, expose too little concurrency. Thus, determining the proper base tile size is important. We do this by applying the bounding constraint of the outer product for $Q$ from Eq. 10.7 with an $S$ value of slightly more than the expected $S'$ value we measured in Section 10.4.

10.5.1 Aggregating Tiles

Bottom up tile formation starts by attempting to aggregate base blocks together into larger tiles that can form outer products while still having enough memory available for the input buffers. All aggregations must follow a global layout so the leaf node may not change the shape of the base blocks to make a more square outer product. It simply calculates how many blocks it can legally fit according to the FF from Eq. 10.6. Once the children of a subtree have finished, the subtree will act as a leaf with one exception: it will attempt to aggregate all its children’s result tiles as well as its own memory into a super block. The method for checking if additional base blocks can be added is the same as the children, but the fill factor calculation that determines the optimal number of base blocks to incorporate is determined by the subtree’s local memory size added with all children’s outer product result tiles due to assumption 3 of our machine model from Section 10.2.1. The rest of the memory is divided and utilized for the A and B
tiles according to the dimensions created by the HIC. Additionally, we insure that an upper level input buffer can hold a large enough tile to support lower level input buffer reads. This assignment continues sequentially all the way through the memory tree until the root finishes by initiating the first data movement of matrices from DRAM. While this sequential section could delay initial startup of the chip, all calculations are either constant time or logarithmic operations and loading of the initial $C$ matrix (in the case of MM) can begin for pinned result tiles as soon as they are decided. In the case of GPUs where all memory is reserved per Streaming Multiprocessor, this setup could be precalculated in parallel making for shorter startup. After all nodes of the memory tree are initialized, the spawning of tasks for computation can begin.

10.5.2 Creating Tasks

As previously mentioned, outer product operations have synchronization requirements if multiple operations are occurring in parallel. In order to perform these operations yet still maintain high performance, we implement a hierarchically double buffered and load balanced asynchronous computation. Again, this operation and the tasks follow recursively in the same style as Garcia et al. [53]. This ensures coherence through the broadcast of input buffers down the memory hierarchy in a structured manner as each memory level copies varying sections of $A$ and $B$ from the higher memory level. This will be energy optimal since each memory region uses a $k$ value determined by our energy model. For partial result tiles in higher levels of memory, a load balancer will assign tiles and each execution engine will perform direct DMA transfers, bypassing all other memory structures. While we could have provided additional data reuse by recruiting groups of XEs in the same block or unit to perform a similar input broadcast into the lower level buffer just as the lower level tiles did, this would add more synchronization and potentially affect performance. However, we could consider investigating this tradeoff in future work.
10.6 Experimental Results

10.6.1 Testbed

We experiment using a functional simulator, FSim, which is a heavily multi-threaded and multi-process simulator created by Intel that models the TG architecture. Each execution or control engine is implemented through an individual thread, and so are the various load-store queues, and memory controllers and memory banks at each level of the TG hierarchy. Using a 32 node cluster, we simulate up to $\frac{1}{8}$ of the targeted 2048-core TG chip, i.e., up to 4 units of 8 blocks each, with 8 execution engines and one control engine in each block ($\approx 256$ cores). Because we are only simulating part of the chip, we reduce the chip area to 64 $mm^2$ for performing On-Chip network energy calculations, and modify the amount of memory in the Unit and Chip shared memories in order to maintain a hierarchy inversion ratio of 2:1 as seen in Table 10.1.

<table>
<thead>
<tr>
<th>Chip Shared Memory</th>
<th>16 MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Units/Chip</td>
<td>4</td>
</tr>
<tr>
<td>Unit Shared Memory</td>
<td>8 MB</td>
</tr>
<tr>
<td>Blocks/Unit</td>
<td>8</td>
</tr>
<tr>
<td>Block Shared Memory</td>
<td>2 MB</td>
</tr>
<tr>
<td>Base Tile Size $\rightarrow n$</td>
<td>30</td>
</tr>
<tr>
<td>Base Tile Size $\rightarrow k$</td>
<td>30</td>
</tr>
</tbody>
</table>

Table 10.1: Simulation Parameters

We trace and count all matrix data movements from any memory module in the hierarchy using our runtime, and in addition we report relative energy consumption provided by FSim that includes dynamic tiling computations and runtime overheads. However, FSim is not cycle accurate so we are unable to estimate static power consumption, but all dynamic energy consumption is measured using approximations developed from architectural designs. For this paper, since we are more interested in
data movement, we fix the voltage to be in superthreshold operation so all dynamic energy consumption is on that order.

10.6.2 Tiling Related Results

![Memory Access to Differing Levels of Memory](image)

Figure 10.10: Memory Accesses

Figure 10.10 shows the number of memory accesses to all shared memories on the chip. Our energy aware algorithm gives a clear preference for the closer memory operations, preferring to access BSM 20 times more than DRAM. In fact, the algorithm favors the local operations so strongly that the number of DRAM operations is exactly the lower bound on the number of accesses to do the MM operation. This is in spite of the C matrix being 83% of the size of the CSM showing that our algorithm can easily operate on working sets larger than the highest capacity of memory in the hierarchy.
10.6.3 Machine Related Results

The preference for on chip memory operations over DRAM accesses also is very helpful for bandwidth utilization as well. Given that a $1050 \times 1590 \times 1590$ requires 5.3 GFLOPs and our tiling scheme is able to only require 60 MB of loads or stores to DRAM, if the TG architecture were computing at the superthreshold throughput levels of 1.75 TFLOPS we would expect that $\frac{1}{5}$ of a chip could perform, it would still only require a DRAM bandwidth of 20 GB/s. This could potentially be an even larger reduction in off chip bandwidth requirements if the full memory capacity were simulated.

In Figure 10.11, we show the relative energy consumption (without static energy) to the 4 different shared memory regions of the chip as well as the dynamic energy consumption of the processors for three different MM sizes. Here we notice that even though the BSM, USM, and CSM are read and written orders of magnitude more than the DRAM, the energy consumed by the DRAM is still much more than the more local memory operations.

10.7 Related Work

Space filling curves have long been known to provide locality even with successful applications in non-obvious areas such as database optimization [79]. Specifically for Matrix Multiply, Chatterjee et. al [31] studied recursive data layouts for multiple kinds of Morton curves as well as Hilbert curves in the context of Matrix Multiply, while Bader and Zenger [19] created an algorithm using Peano curves. More recently, Ballard et al. [20] used a Morton inspired ordering in which they divide by the largest dimension which in a square matrix resolves to Morton order. Compared to this work, these previous works solely looked at the locality properties of space filling curves in order to provide cache friendly ordering. We expand the use of space filling curves to incorporate a hierarchy of scratchpad memories with additional constraints to ensure the tiling scheme provides energy optimal data movement. Furthermore, this technique
is not solely a data layout algorithm but also leverages the curve in a scheduler for more choreographed data movement to increase locality.

In regards to algorithms, Frigo et. al [52] define an algorithm as being cache oblivious when the algorithm is cache optimal (producing a minimal number of cache misses) without requiring any parameters defining the cache. They do this by using the inclusion property of caches to simplify the problem into a 2 memory space problem: fast cache memory and slow system memory similar to our formulation. They then can infer cache optimality for any algorithm that provably minimizes communication between these two memories so long as the algorithm is not a function of the sizes. However, this means exclusive caches or noncoherent caches or scratchpads like the

\[ \text{Figure 10.11: Relative Energy Consumption} \]
CPCI hierarchies we target can not apply to a cache oblivious algorithm or if so a
complex analysis of the coherence algorithm is necessary to determine what the maxi-
mum working set the cache can hold and under what conditions of memory operations
that maximum working set can exist. However, we only require the energy cost to be
inclusive and let the capacity be a variable we define in our model. We then man-
ually allocate buffers and coordinate coherence through the use of dependencies in a
fine-grained asynchronous runtime. The downside to our algorithm is that it naturally
operates using a machine model where all data movement is explicit and formulating
an algorithm within a traditional cache hierarchy would be difficult if not impossible
for some caches. Since our target is a prototype architecture that fits our machine
model, this weakness is not relevant.

More recent work related to ours would also include the communication avoid-
ing (CA) classes of algorithms [39, 69, 18, 64]. These build upon the cache oblivi-
ous work but also extend to network obliviousness as well. CARMA [40] utilizes a
breadth-first/depth-first hybrid algorithm that leverages additional available memory
to reduce communication across distributed-memory and NUMA shared memory ma-
chines. However, we assume that energy consumption will be a dominating and limiting
factor within a chip in future architectures rather than bandwidth. Because CA algo-
rithms are cache oblivious, they place equal weight on memory accesses regardless of
the energy liabilities they generate which could limit overall performance when ther-
mal constraints are considered. However, future interconnects will certainly have high
correlations between bandwidth and energy consumption [33] which begs the question:
how does this algorithm compare to current CA algorithms and could it compliment
them?

10.8 Conclusion and Future Work

Having presented a property we call memory capacity inversion, formulated an
energy aware tiling method, and applied a custom space filling curve to implement
the method, there are a few promising lines of research to pursue. In the area of
memory capacity inversion and hardware analysis, we can quantify the scheduling overheads of this algorithm in a strictly cycle accurate simulator, so that not just data movement energy can be obtained, but a total estimate of energy per operation could help guide computer architects in how effective inverted memory hierarchies can be utilized. Likewise, our machine model with static energy cost could be extended to model bandwidth consumption on chip using data like in [33]. We could also develop a cache coherence methodology that could bridge traditional cache techniques with explicit techniques like our own. In the field of compilers, we could extend the dynamic space filling curves and energy model to a more general framework and leverage the analytical power of Polyhedral models. This could perhaps provide a more automated method of performing techniques like jagged tiling [117]. In the field of runtimes, there is the potential for using runtime information to guide custom schedulers for optimal locality using our framework.
Chapter 11

CONCLUSION

This dissertation has presented an array of different statistically sound models that when integrated into a standard functional simulator can provide accurate processor behavior. Equipped with this behavior, the development of a exascale class runtime can begin in earnest.

11.1 Challenges of Future Software

Tasks like power and temperature management are classicly decoupled from the scheduler and resource allocators of a runtime. Unlike superthreshold design, near threshold processors will greatly vary in performance and energy efficiency. This means that the defined line between program and hardware must be blurred. Techniques must be implemented into runtimes that have a concept of the interactions at play between these two sides if the necessary energy efficiency and performance is to be reached. While this is a heavy burden placed on software, it is the best viable option moving forward if we are to reach exascale in an economically feasible manner.

11.2 Introspection

We began the dissertation looking at very basic feedback control-based runtimes which were introspective of the hardware. However, REST was not introspective with the program. We expect a big challenge of a future runtime will need to have introspection on both hardware and software. This dissertation provides the tools to ensure that hardware properties will be accurate and available for introspection in high throughput functionally accurate simulators. However this leaves the programmatic analysis an open research topic.
11.3 Programming Models

We introduce a codelet-based algorithm that has an energy optimizing data movement scheme. However, there are many problems much less formulaic than matrix multiply. While large efforts are being made in the field of fine-grained asynchronous runtimes, much more research in new algorithms and adapting an extensive codebase into that framework is necessary to fully solve the programming problem that is only exacerbated by utilizing NTV.

11.4 Everything will be Heterogeneous

It is clear that chip variance will make homogenous scheduling to all processors an untenable strategy. We have provided realistic models of variance to simulate future potential chips as well as the timing, leakage, and temperature models to accompany such a framework. While we have not solved all the problems necessary to reach exascale, there is still great promise and hopefully the work presented has made a small contribution in that regard.
BIBLIOGRAPHY


Symposium on Microarchitecture*, MICRO 36, pages 217–, Washington, DC, USA,

48-core ia processor with variation-aware scheduling and optimal core mapping.

mos transistor model valid in all regions of operation and dedicated to low-voltage
114, July 1995.

[46] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam,
and Doug Burger. Dark silicon and the end of multicore scaling. In *Proceedings
of the 38th Annual International Symposium on Computer Architecture*, ISCA
’11, pages 365–376, New York, NY, USA, 2011. ACM.

[47] L. Fenton. The sum of log-normal probability distributions in scatter transmission
1960.


Fourier ; translated, with notes, by Alexander Freeman ; edited for the

[50] Vincent W. Freeh and David K. Lowenthal. Using multiple energy gears in mpi
programs on a power-scalable cluster. In *Proceedings of the tenth ACM SIGPLAN
symposium on Principles and practice of parallel programming*, PPoPP ’05, pages
164–173, New York, NY, USA, 2005. ACM.

within-die spatial correlation effects for process-design co-optimization. In *Quality

Cache-oblivious algorithms. In *Proceedings of the 40th Annual Symposium on
Foundations of Computer Science*, FOCS ’99, pages 285–, Washington, DC, USA,


[86] B. Linares-Barranco and T. Serrano-Gotarredona. A physical interpretation of
the distance term in pelgrom’s mismatch model results in very efficient cad. In

LITTLE/mp.git. [Online].

[88] Yongpan Liu, R.P. Dick, Li Shang, and Huazhong Yang. Accurate temperature-
dependent integrated circuit leakage power estimation is easy. In Design, Au-
tomation Test in Europe Conference Exhibition, 2007. DATE ’07, pages 1–6,
April 2007.

[89] David Luebke, Mark Harris, Jens Krüger, Tim Purcell, Naga Govindaraju, Ian
Buck, Cliff Woolley, and Aaron Lefohn. Gpgpu: General purpose computation
on graphics hardware. In ACM SIGGRAPH 2004 Course Notes, SIGGRAPH
’04, New York, NY, USA, 2004. ACM.

[90] Kai Ma, Xue Li, Ming Chen, and Xiaorui Wang. Scalable power control for
many-core architectures running multi-threaded applications. In Proceedings of
the 38th annual international symposium on Computer architecture, ISCA ’11,
pages 449–460, New York, NY, USA, 2011. ACM.

based dynamic voltage and frequency scaling for a multiple clock domain micro-

circuits considering process variations. In Quality Electronic Design (ISQED),

[93] Akihiko Miyoshi, Charles Lefurgy, Eric Van Hensbergen, Ram Rajamony, and
Raj Rajkumar. Critical power slope: understanding the runtime effects of fre-
quency scaling. In Proceedings of the 16th international conference on Supercom-
puting, ICS ’02, pages 35–44, New York, NY, USA, 2002. ACM.

[94] Antonio Monari, Anthony Scemama, and Michel Caffarel. Large-scale quantum
monte carlo electronic structure calculation on the EGEE grid. In Proceedings
of the 5th International Workshop on Distributed Cooperative Laboratories : In-
strumenting the Grid (2010), 2010.


[96] Morton. A computer oriented geodetic data base and a new technique in file


Appendix A

DERIVING GAUSSIAN MOMENTS

Now we must show that the first 2 moments are correct.

\[
E[X] = \int_{-\infty}^{+\infty} x \cdot \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx
\]

\[
\int_{-\infty}^{+\infty} ((x - \mu) + \mu) \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx
\]

\[
\int_{-\infty}^{+\infty} (x - \mu) \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx + \mu \cdot \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx
\]

\[
y = x - \mu \quad dy = dx
\]

\[
\int_{-\infty}^{+\infty} (y) \frac{1}{\sqrt{2\pi}} e^{-\frac{(y-\mu)^2}{2\sigma^2}} dy + \mu
\]

\[
\int_{-\infty}^{0} (y) \frac{1}{\sqrt{2\pi}} e^{-\frac{(y-\mu)^2}{2\sigma^2}} dy + \int_{0}^{+\infty} (y) \frac{1}{\sqrt{2\pi}} e^{-\frac{(y-\mu)^2}{2\sigma^2}} dy + \mu
\]

\[
\int_{-\infty}^{0} (y) \frac{1}{\sqrt{2\pi}} e^{-\frac{(y-\mu)^2}{2\sigma^2}} dy - \int_{0}^{+\infty} (y) \frac{1}{\sqrt{2\pi}} e^{-\frac{(y-\mu)^2}{2\sigma^2}} dy + \mu
\]

\[
\int_{-\infty}^{0} (y) \frac{1}{\sqrt{2\pi}} e^{-\frac{(y-\mu)^2}{2\sigma^2}} dy - \int_{0}^{0} (-y) \frac{1}{\sqrt{2\pi}} e^{-\frac{(-y-\mu)^2}{2\sigma^2}} \cdot -dy + \mu
\]

\[
\int_{-\infty}^{0} (y) \frac{1}{\sqrt{2\pi}} e^{-\frac{(y-\mu)^2}{2\sigma^2}} dy - \int_{0}^{0} (y) \frac{1}{\sqrt{2\pi}} e^{-\frac{(y-\mu)^2}{2\sigma^2}} dy + \mu
\]

\[
E[X] = \mu
\]

First, we evaluate the expected value using the PDF from Equation 3.14 and apply that to the definition in Equation 3.9. Next, we add and subtract \(\mu\) and separate the integral into two separate evaluations, one with \(\mu\) and one with \((x - \mu)\). The latter evaluation, by definition, will be 1 since it is simply the PDF scaled by \(\mu\) and the first evaluation has another change of variables shown on line 4. Next we break up the definite integral into two pieces and take advantage of the property of odd/evenness to
cancel each evaluation out upon itself, leaving $\mu$ as the expected value. Next, we must verify the variance.

$$Var[X] = \int_{-\infty}^{+\infty} (x - \mu)^2 \cdot \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx$$

$$t = \frac{(x - \mu)}{\sqrt{2\sigma}} \quad dt = \frac{1}{\sqrt{2\sigma}} dx$$

$$\int_{-\infty}^{+\infty} 2\sigma^2 (t)^2 \cdot \frac{1}{\sigma \sqrt{2\pi}} e^{-(t)^2} \sqrt{2\sigma} dt$$

$$\frac{2\sigma^2}{\sqrt{\pi}} \int_{-\infty}^{+\infty} (t)^2 e^{-(t)^2} dt \quad (A.2)$$

Now we evaluate the $2^{nd}$ central moment of the Normal distribution and again we make a change of variables in line 2. This reduces the integral back to $2^{nd}$ order polynomial multiplied by a exponentiated $2^{nd}$ order polynomial. Here we must use Integration by Parts which is provided for reference in Equation A.3 and the portions of our expression are assigned accordingly on line 5 of Equation A.2. On line 6, we rearrange $u = \frac{1}{t}$ so that we can apply L’Hôpital’s and see the evaluation is 0. Finally, the last portion of the integration by parts has resolved to the Gaussian Integral from Equation 3.13, thus confirming the Variance of a Normal Distribution is $\sigma^2$.

\[ d(u \cdot v) = u \cdot dv + v \cdot du \quad (A.3) \]

\[ \int_{a}^{b} u \cdot dv = [uv]_{a}^{b} - \int_{a}^{b} v \cdot du \]
DERIVING EXPECTED VALUES OF LOGNORMAL DISTRIBUTIONS

First, we wish to solve the integral in the form listed in Equation B.1. While this may not be the most direct form of deriving the expected values of the lognormal distribution, it will have great application in Chapters 6. We then give the generalized solution similar to completing the square for a lognormal distribution.

\[
\begin{align*}
\int_{-\infty}^{\infty} e^{ax^2 + bx + c} dx &= \int_{-\infty}^{\infty} e^{-(ax^2 - bx) + c} dx \\
\int_{-\infty}^{\infty} e^{-(ax^2 - bx - \frac{b^2}{4a}) + c} dx &= e^{\frac{-b^2}{4a}} \int_{-\infty}^{\infty} e^{-(ax^2 - bx - \frac{b^2}{4a})} dx \\
e^{-\frac{b^2}{4a}} \int_{-\infty}^{\infty} e^{-(ax^2 - bx - \frac{b^2}{4a})} dt &= e^{-\frac{b^2}{4a}} \int_{-\infty}^{\infty} e^{-t^2} dt \\
\end{align*}
\]

(B.1)

The first 3 lines “complete the square” of the polynomial but while still allowing the square to be negative. Since the extra portions needed to complete the square are invariant, they are brought outside the integral. Then we perform change of variable so that we can create the Gaussian Integral which evaluates to \(\sqrt{\pi}\). Having solved this generalized equation, we will now use it to solve the lognormal distribution. Recall

\[
t = \sqrt{-ax} - \frac{b}{2\sqrt{-a}} \rightarrow dt = \sqrt{-a} \cdot dx
\]

\[
e^{-\frac{b^2}{4a}} \int_{-\infty}^{\infty} e^{-t^2} dt = \frac{\sqrt{\pi}}{\sqrt{-a}} e^{-\frac{b^2}{4a}}
\]
from Equation 3.12 that expected values can be determined using the original PDF rather than the formulated PDF. This means we can derive the expected value for a lognormal distribution using a normal PDF and the exponentiated variable as our function.

\[
E[L] = \int_{-\infty}^{\infty} e^x \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx \\
= \frac{1}{\sigma \sqrt{2\pi}} \int_{-\infty}^{\infty} e^{x+\frac{-(x-\mu)^2}{2\sigma^2}} dx \\
= \frac{1}{\sigma \sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-x^2+2\mu x+2\sigma^2 x-\mu^2} dx \\
a = \frac{1}{-2\sigma^2} b = \frac{\mu + \sigma^2}{\sigma^2} c = -\frac{\mu^2}{2\sigma^2} \\
= \frac{1}{\sigma \sqrt{2\pi}} \int_{-\infty}^{\infty} e^{ax^2+bx+c} dx \\
= \frac{1}{\sigma \sqrt{2\pi}} \cdot \sqrt{\pi} e^{-\frac{\mu^2}{2\sigma^2}} \left( \frac{\left(\frac{\mu+\sigma^2}{\sigma^2}\right)^2}{1-2\sigma^2} \right) \\
E[L] = e^{\mu + \frac{1}{2} \sigma^2}
\]

We first get the function integrated into the normal exponential, then since we are integrating over x, we work to make it into the polynomial form solved in Equation B.1. After that, algebraic manipulations bring us to the final value. It will also be very useful for this dissertation to have a partial expectation derived below in the traditional
manner.

\[
E[L|a < L < b] = \int_a^b e^{x} \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx
\]

\[
\frac{1}{\sigma \sqrt{2\pi}} \int_a^b e^{-(x^2-2\sigma x+2\sigma^2)/2\sigma^2} dx
\]

\[
\frac{1}{\sigma \sqrt{2\pi}} \int_a^b e^{-(x-(\mu+\sigma^2))2/2\sigma^2-(\mu+\frac{1}{2}\sigma^2)} dx
\]

\[
e^{\mu+\frac{1}{2}\sigma^2} \frac{1}{\sigma \sqrt{2\pi}} \int_{\ln(a)}^{\ln(b)} e^{-(x-\mu)^2/2\sigma^2} dx
\]

\[
E[L|a < L < b] = e^{\mu+\frac{1}{2}\sigma^2} \left( \Phi\left(\frac{\ln(b) - \mu - \sigma^2}{\sigma}\right) - \Phi\left(\frac{\ln(a) - \mu - \sigma^2}{\sigma}\right) \right)
\]

\[
E[L|a < X < b] = e^{\mu+\frac{1}{2}\sigma^2} \left( \Phi\left(\frac{b - \mu - \sigma^2}{\sigma}\right) - \Phi\left(\frac{a - \mu - \sigma^2}{\sigma}\right) \right)
\]

A partial expectation is the expected value given some knowledge that limits the bounds of the lognormal RV for that particular instance. This is different from a truncated distribution where it is never possible for the RV to instantiate outside of some bound. In Equation B.3, we see the traditional derivation for a lognormal expected value but having applied the change of variables to the bounds of integration in line 9 so that it is not infinitely bounded. This creates a partial expectation in a simple closed form solution using the \( \Phi \) function.
Appendix C

PERMISSION LETTERS

SPRINGER LICENSE
TERMS AND CONDITIONS
Nov 17, 2015

This is a License Agreement between Kelly Livingston ("You") and Springer ("Springer") provided by Copyright Clearance Center ("CCC"). The license consists of your order details, the terms and conditions provided by Springer, and the payment terms and conditions.

All payments must be made in full to CCC. For payment instructions, please see information listed at the bottom of this form.

License Number 3751590499950
License date Nov 17, 2015
Licensed content publisher Springer
Licensed content publication Computer Science - Research and Development
Licensed content title Computer using too much power? Give it a REST (Runtime Energy Saving Technology)
Licensed content author Kelly Livingston
Licensed content date Jan 1, 2012
Volume number 29
Issue number 2
Type of Use Thesis/Dissertation
Portion Full text
Number of copies 10
Author of this Springer article Yes and you are the sole author of the new work
Order reference number None
Title of your thesis / dissertation TOWARDS CO-DESIGNED ENERGY EFFICIENT COMPUTING AND RUNTIMES: SIMULATION-FRAMEWORK AND EXPERIMENTS
Expected completion date Jan 2016
Estimated size(pages) 138
Total 0.00 USD
Terms and Conditions

Introduction
The publisher for this copyrighted material is Springer Science + Business Media. By clicking "accept" in connection with completing this licensing transaction, you agree that the following terms and conditions

147
apply to this transaction (along with the Billing and Payment terms and conditions established by Copyright Clearance Center, Inc. ("CCC"), at the time that you opened your Rightslink account and that are available at any time at http://myaccount.copyright.com).

Limited License
With reference to your request to reprint in your thesis material on which Springer Science and Business Media control the copyright, permission is granted, free of charge, for the use indicated in your enquiry.

Licenses are for one-time use only with a maximum distribution equal to the number that you identified in the licensing process.

This License includes use in an electronic form, provided its password protected or on the university's intranet or repository, including UMI (according to the definition at the Sherpa website: http://www.sherpa.ac.uk/romeo/). For any other electronic use, please contact Springer at (permissions.dordrecht@springer.com or permissions.heidelberg@springer.com).

The material can only be used for the purpose of defending your thesis limited to university-use only. If the thesis is going to be published, permission needs to be re-obtained (selecting "book/textbook" as the type of use).

Although Springer holds copyright to the material and is entitled to negotiate on rights, this license is only valid, subject to a courtesy information to the author (address is given with the article/chapter) and provided it concerns original material which does not carry references to other sources (if material in question appears with credit to another source, authorization from that source is required as well).

Permission free of charge on this occasion does not prejudice any rights we might have to charge for reproduction of our copyrighted material in the future.

Altering/Modifying Material: Not Permitted
You may not alter or modify the material in any manner. Abbreviations, additions, deletions and/or any other alterations shall be made only with prior written authorization of the author(s) and/or Springer Science + Business Media. (Please contact Springer at (permissions.dordrecht@springer.com or permissions.heidelberg@springer.com)

Reservation of Rights
Springer Science + Business Media reserves all rights not specifically granted in the combination of (i) the license details provided by you and accepted in the course of this licensing transaction, (ii) these terms and conditions and (iii) CCC's Billing and Payment terms and conditions.

Copyright Notice: Disclaimer
You must include the following copyright and permission notice in connection with any reproduction of the licensed material: “Springer and the original publisher /journal title, volume, year of publication, page, chapter/article title, name(s) of author(s), figure number(s), original copyright notice) is given to the publication in which the material was originally published, by adding; with kind permission from Springer Science and Business Media”

Warranties: None

Example 1: Springer Science + Business Media makes no representations or warranties with respect to the licensed material.

Example 2: Springer Science + Business Media makes no representations or warranties with respect to the licensed material and adopts on its own behalf the limitations and disclaimers established by CCC on its behalf in its Billing and Payment terms and conditions for this licensing transaction.

Indemnity

You hereby indemnify and agree to hold harmless Springer Science + Business Media and CCC, and their respective officers, directors, employees and agents, from and against any and all claims arising out of your use of the licensed material other than as specifically authorized pursuant to this license.

No Transfer of License

This license is personal to you and may not be sublicensed, assigned, or transferred by you to any other person without Springer Science + Business Media’s written permission.

No Amendment Except in Writing

This license may not be amended except in a writing signed by both parties (or, in the case of Springer Science + Business Media, by CCC on Springer Science + Business Media’s behalf).

Objection to Contrary Terms

Springer Science + Business Media hereby objects to any terms contained in any purchase order, acknowledgment, check endorsement or other writing prepared by you, which terms are inconsistent with these terms and conditions or CCC’s Billing and Payment terms and conditions. These terms and conditions, together with CCC’s Billing and Payment terms and conditions (which are incorporated herein), comprise the entire agreement between you and Springer Science + Business Media (and CCC) concerning this licensing transaction. In the event of any conflict between your obligations established by these terms and conditions and those established by CCC’s Billing and Payment terms and conditions, these terms and conditions shall control.

Jurisdiction

All disputes that may arise in connection with this present License, or
the breach thereof, shall be settled exclusively by arbitration, to be held
in The Netherlands, in accordance with Dutch law, and to be conducted
under the Rules of the 'Netherlands Arbitrage Instituut' (Netherlands
Institute of Arbitration). **OR:**

**All disputes that may arise in connection with this present**
License, or the breach thereof, shall be settled exclusively by
arbitration, to be held in the Federal Republic of Germany, in
accordance with German law.

**Other terms and conditions:**
v1.3

Questions? customer@copyright.com or +1-855-239-3415 (toll free in the US) or
+1-978-646-2777.