MULTILAYER TECHNIQUES TO ADDRESS PARAMETER VARIATION

BY

MIRCEA-RADU TEODORESCU

Dipl. Eng., Technical University of Cluj-Napoca, 2002
M.S., University of Illinois at Urbana-Champaign, 2005

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2008

Urbana, Illinois

Doctoral Committee:

Professor Josep Torrellas, Chair
Professor Marc Snir
Professor Sarita Adve
Assistant Professor Deming Chen
Professor Todd Mowry, Carnegie Mellon University
Chris Wilkerson, Intel Corp.

ABSTRACT

As integrated-circuit technology continues to scale, process variation is becoming an issue that cannot be ignored at the microarchitecture and system levels. Process variation is particularly detrimental to a processor's frequency and leakage power. To solve this growing problem, solutions at different levels of the computing stack are needed. This thesis presents a couple of such solutions.

The first solution, is a circuits technique that has important implications on the microarchitecture. It is based on the previously-proposed Fine-Grain Body Biasing (FGBB), where different parts of the processor chip are given a voltage bias that changes the speed and leakage properties of their transistors. Previous work proposed determining the optimal body bias voltages at manufacturing time and setting them permanently for the lifetime of the chip. In this thesis, I propose a new technique (called Dynamic FGBB - D-FGBB), which allows the continuous re-evaluation of the bias voltages to adapt to dynamic conditions.

Within-die process variation causes individual cores in a Chip Multiprocessor (CMP) to differ substantially in both static power consumed and maximum frequency supported. In this environment, ignoring variation effects when scheduling applications or when managing power with Dynamic Voltage and Frequency Scaling (DVFS) is suboptimal. This thesis presents a set of variation-aware algorithms for application scheduling and power management. One such power management algorithm, uses linear programming to find the best voltage and frequency levels for each of the cores in the CMP — maximizing throughput at a given power budget.

To Magdalena, for her love.

ACKNOWLEDGMENTS

First and foremost I would like to thank my adviser, Josep Torrellas for his moral and financial support during my Ph.D, for teaching me how to do quality research and inspiring me to pursue an academic career. I am particularly grateful to him for believing in me from the beginning and for teaching me the value of perseverance in the face of rejection.

I would like to thank the members of my Ph.D. committee Prof. Marc Snir, Prof. Sarita Adve, Prof. Deming Chen, Prof. Todd Mowry and Chris Wilkerson for their feedback on my research and my thesis.

I was very fortunate to be part of the outstanding IACOMA research group and I believe that every one of its past and present members has contributed in some way to the successful completion of my Ph.D. With some of them I worked directly and we co-authored papers. I would like to thank Jun Nakano who pioneered the research on variation in our group — a path that many of us followed after his graduation. I was impressed my his analytical skills, his whiteboard-art skills and his unique humor. I would also like to thank Brian Greskamp for the great discussions and invaluable feedback on my research. I also owe him a lot for his willingness to proofread my papers and job application and for always providing great feedback. I also enjoyed working with Abhishek Tiwari and Smruti Sarangi and have benefited a lot from their insights on my research.

Even though I did not work directly with other members of our research group, I benefited a lot from interactions will all of them. I would like to thank Pablo Montesinos for his friendship and for orchestrating my successful conversion to world of Apple computers. I would like to thank Luis Ceze and Karin Strauss for their help and advice on my job application process, for the outstanding meals I had at their house and for their friendship. I owe my addiction to great coffee to Paul Sack with whom I shared many enjoyable lunches and discussions about data races and Vespa scooters. He has also patiently read some of my papers with a critical look, for which I am very grateful. I would also like to thank James Tuck for his help and encouragement with my academic application and Wonsun Ahn for proofreading my application package. I am also very grateful to the younger generation of iacomers: Ulya Karpuzcu, Abdullah Muzahid, Sandy Qi and Xuehai Qian for putting up with my endless practice talks and providing feedback on my work. Special thanks to Sandy for helping make my latest trip to China very enjoyable and memorable. I would also like to thank our former postdoctoral researcher Wei Liu for his help and feedback.

Other IACOMA alumni have left a significant mark on my career. In particular, I would like to thank José Renau for writing the world's best multiprocessor simulator and José Martínez and Milos Prvulovic for helpful career advice.

At University of Illinois I benefited from working in a great environment and with a lot of incredibly talented faculty and students. I would like to thank Yuanyuan Zhou for working with me on a publication. Other faculty that have shaped my career at UIUC include Deming Chen, Sarita Adve, Craig Zilles, Vikram Adve, Sanjay Patel and many others. I also worked with students from other groups at UIUC. In particular, I would like to thank Pin Zhou for our successful collaboration on a research project that led to us co-authoring a great paper. Other students I have interacted with include James Roberts, Naveen Neelakantam, Pierre Salverda, Man-Lap Li, Pradeep Ramachandran and so many more.

I am very grateful to Intel Corporation for their support during my Ph.D. I was very fortunate to do an internship at Intel Pittsburgh in 2006 and I discovered the most amazing group of people there. In particular I would like to thank Mike Kozuch and Phil Gibbons, my mentors at Intel Pittsburgh, for a great collaboration. I would also like to thank Todd Mowry who was the lab director at the time and Shimin Chen. Intel also grated me a fellowship for the final year of my Ph.D, which in addition to the generous financial support, allowed me to present my work in front of Intel researchers and to make new professional contacts there. In particular, I would like to thank Chris Wilkerson, my fellowship mentor, for his feedback on my research and his support of my job application.

I would not have made it to this point without the support of friends and family. I would like to thank my dear friends Nancy and Gerald Yaxley for their support and for being my second family away from home. I would also like to thank my long time friend Sergiu Nedevschi for his continued help and moral support.

I would like to thank my parents, Iuliana and Emil Teodorescu for everything they have done for me. First of all they set a great example by always emphasizing education in our family's life. They did their best to make sure that my brother Gabi and I always made learning a priority and always found ways to make it easier for us, even in the face of hardship. My parents also bought my first computer — back in the day when they used to cost a small fortune — because they strongly believed it would further my education and they were certain that the sacrifice was worth it. They have always supported my choices even when those choices meant I would have to move thousands of kilometers away from them. For all they have done I can never hope to thank them enough. I can only hope to have learned from them enough to do the same for my children.

Finally, I would like to thank my wonderful wife Magdalena. Without her I wouldn't have made it to this point. She has been more supportive of me than any husband can ask. She has a contagious optimism and positive outlook that got us through the most difficult situations. She stood by me when I decided to leave my country and pursue a Ph.D, even though that meant we would live apart for three long years. Her love gives me strength and balance every day and for that I will forever be grateful.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

Introduction and Motivation

Parameter variation — the divergence of process parameters from their nominal specifications — is a challenge that has been recognized in the circuits community for many years. Its detrimental effects on microprocessor frequency and power consumption have been well documented. However, as integrated-circuit technology continues to scale and variation worsens, it is becoming an issue that cannot be ignored at the microarchitecture and system levels. To solve this growing problem, solutions at different levels of the computing stack are needed.

The work presented in this thesis makes several contributions in this space. First, we developed a parametrized model of process variation (presented in Chapter 2). The model was used throughout this work to estimate the effects of variation on current and future chip multiprocessors and to test the effectiveness of our solutions for mitigating and tolerating the effects of parameter variation. This thesis presents two such techniques.

The first one, presented in Chapter 3, is a circuits technique that has important implications on the microarchitecture. It is based on the previously-proposed Fine-Grain Body Biasing (FGBB), where different parts of the processor chip are given a voltage bias that changes the speed and leakage properties of their transistors. Previous work proposed determining the optimal body bias voltages at manufacturing time and setting them permanently for the lifetime of the chip. In this thesis, a new technique is presented (called Dynamic FGBB - D-FGBB), which allows the continuous re-evaluation of the bias voltages to adapt to dynamic conditions. Our results show that D-FGBB is very versatile and effective. Specifically, with the processor working in normal mode at fixed frequency, D-

FGBB reduces the leakage power of the chip by an average of 28-42% compared to static FGBB. Alternatively, with the processor working in a high-performance mode, D-FGBB increases the processor frequency by an average of 7-9% compared to static FGBB, or 7-16% compared to no body biasing.

Within-die process variation causes individual cores in a Chip Multiprocessor (CMP) to differ substantially in both static power consumed and maximum frequency supported. In this environment, ignoring variation effects when scheduling applications or when managing power with Dynamic Voltage and Frequency Scaling (DVFS) is suboptimal. In the second part of this thesis (Chapter 4), I present a set of variation-aware algorithms for application scheduling and power management. One such power management algorithm, called *LinOpt*, uses linear programming to find the best voltage and frequency levels for each of the cores in the CMP — maximizing throughput at a given power budget. In a 20-core CMP, the combination of variation-aware application scheduling and *LinOpt* increases the average throughput by 12–17% and reduces the average ED^2 by 30–38% — all relative to using variation-aware scheduling together with a simple extension to Intel's Foxton power management algorithm.

The final chapters of this thesis present some relevant related work (Chapter 5) and conclusions (Chapter 6).

CHAPTER 2

A Model for Parameter Variation

2.1 Introduction and Background

Parameter variation encompasses several effects including process, voltage and temperature variation. Process variation is caused by the inability to precisely control the fabrication process at small-feature technologies. It is a combination of systematic effects [25, 60, 75] (e.g., lithographic lens aberrations) and random effects [5] (e.g., dopant density fluctuations). Voltage variations can be caused by *IR* drops in the supply distribution network or by *L dI/dt* noise under changing load. Temperature variation is caused by spatially- and temporally-varying factors. All of these variations are becoming more severe and harder to tolerate as technology scales to minute feature sizes.

Two key process parameters subject to variation are the transistor threshold voltage, V_{th}, and the effective length, L_{eff}. V_{th} is especially important because its variation has a substantial impact on two major properties of the processor, namely the frequency it attains and the leakage power it dissipates. Moreover, V_{th} is also a strong function of temperature, which increases its variability [78].

One of the most harmful effects of variation is that some sections of the chip are slower than others — either because their transistors are intrinsically slower or because high temperature or low supply voltage renders them so. As a result, circuits in these sections may be unable to propagate signals fast enough and may suffer timing errors. To avoid these errors, designers in upcoming technology generations may slow down the frequency of the processor or create overly conservative designs. It has been suggested that parameter variation may wipe out most of the potential gains provided by one technology generation [7].

To examine the impact of variation, we need an efficient way to accurately model its effects on microprocessor chips. Existing models of parameter variation are generally too low-level to be suited for microarchitectural studies. This is the main reason we developed our own model that is at the right level of abstraction to model variation impact on microarchitectural units. Our model is parametrized allowing a range of variation levels to be modeled, it considers both random and systematic effects and also models the spatial correlation of the systematic component.

2.2 Modeling Variation

Parameter variation can be broken down into two major components, namely die-to-die (D2D) and within-die (WID). Furthermore, WID variation can be divided into random and systematic components. Thus, variation in any parameter P, like V_{th} or L_{eff}, can be represented as follows:

$$\Delta P = \Delta P_{D2D} + \Delta P_{WID} = \Delta P_{D2D} + \Delta P_{rand} + \Delta P_{sys}$$

In this work, we focus on WID variation, but D2D variation is easily modeled: One needs only add a chip-wide offset to the V_{th} and L_{eff} parameters of every transistor on the die. For simplicity, we model the two components of WID process variation with normal distributions. This is an accepted approach that has been used elsewhere [73].

From a microarchitectural perspective, V_{th} and L_{eff} variation are of key importance: they directly affect a chip's leakage and frequency. The WID variation of these parameters is impacted by both systematic and random effects [5]. Limitations of the lithography and other manufacturing processes introduce systematic variations. Typically, such variations exhibit a spatial structure with a certain scale of parameter changes over the two-dimensional space [25, 60, 75]. On the other hand, a variety of materials effects, such as changes in the dopant density of the channel [5] and lithographic phenomena like line edge roughness [85], introduce random variations. Such random variations have a different profile for each transistor and are in effect noise superimposed on the systematic variation.

We treat random and systematic variation separately, since they arise from different physical phenomena. As described in [73], we assume that their effects are additive.

2.2.1 Systematic Variation

Systematic variation is characterized by a spatial correlation, meaning that adjacent areas on a chip have roughly the same systematic components. Such correlation can be characterized using different models. For example, [52, 73] use a quad tree model that recursively partitions the die into four parts. In this work, we use a different method that models systematic variation using a multivariate [62] normal distribution with a specific correlation structure.

We divide a chip into N small rectangular cells. The value of the systematic component of V_{th} is assumed to be constant within one small cell. This is consistent with other work [73]. We also assume that the value of V_{th} for all the cells has a normal distribution with mean μ and standard deviation σ. Along with this, the values of V_{th} are spatially correlated.

To determine the spatial correlation, we make the following assumptions. First, we treat the distribution of V_{th} as isotropic and position-independent. This means that given two points $\vec{x}$ and $\vec{y}$ in the grid, the correlation between them depends only on the distance between $\vec{x}$ and $\vec{y}$, and not on the direction of the segment that goes from $\vec{x}$ to $\vec{y}$, or the position of $\vec{x}$ and $\vec{y}$ in the grid. We verify these assumptions by analyzing the empirical data obtained by Friedberg *et al.* [25] and using results from [73]. Nevertheless, we acknowledge the fact that in reality there are some anisotropic effects — for example in defects due to misalignment of the masks.

Given the assumptions of position independence and isotropy, the correlation function of $V_{\text{th}}(\vec{x})$ and $V_{\text{th}}(\vec{y})$ is expressible as $\rho(r)$, where $r = |\vec{x} - \vec{y}|$. By definition, $\rho(0) = 1$ (i.e., totally correlated). We also set $\rho(\infty) = 0$ (i.e., totally uncorrelated) because two infinitely separated points have independent V_{th} when we only consider WID variation.

To determine how $\rho(r)$ changes from $\rho(0) = 1$ to $\rho(\infty) = 0$ as r increases, we use the Spherical model [16, 38], which has the following form:

$$\rho(r) = \begin{cases} 1 - (3r/2\phi) + (r/\phi)^3/2 & \text{if (r} \leq \phi) \\ 0 & \text{if (r} > \phi) \end{cases} \tag{2.1}$$

This model is very similar to the correlation function experimentally measured by Friedberg *et al.* [25] for the WID variation of gate length. Our rationale for using this model is that gate length variation is the main determinant of systematic V_{th} variation.

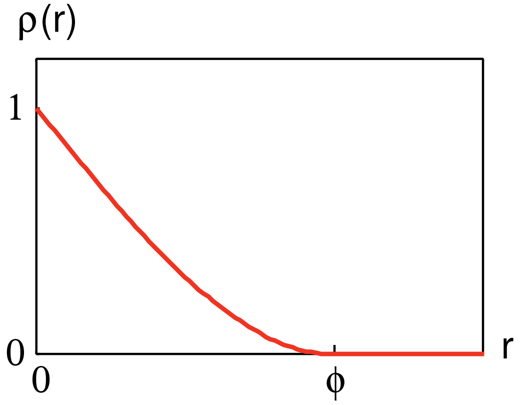


Figure 2.1: Spherical function.

Figure 2.1 shows the function $\rho(r)$. At a finite distance ϕ that we call *range*, the function converges to zero. Intuitively, this assumption implies that the V_{th} of a transistor is highly correlated to the V_{th} of those in its immediate vicinity. The correlation decreases linearly with distance at small distances. Then, it decreases more slowly. At distance ϕ, there is no longer any correlation between two transistors' V_{th}.

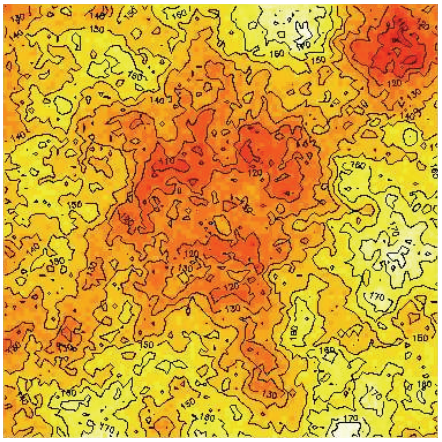


Figure 2.2: Systematic V_{th} variation map for a chip with $\phi = 0.5$.

In this work, we express ϕ as a fraction of the chip's width. A large ϕ implies that large sections of the chip are correlated with each other; the opposite is true for small ϕ. As an illustration, Figures 2.2 and 2.3 show example systematic V_{th} variation maps for chips with $\phi = 0.1$ and $\phi = 0.5$. Both maps were generated by the geoR statistical package [66] of R [64]. In the $\phi = 0.5$ case, we discern large spatial features, whereas in the $\phi = 0.1$ one, the features are small. A distribution without any correlation ($\phi = 0$) appears as white noise.

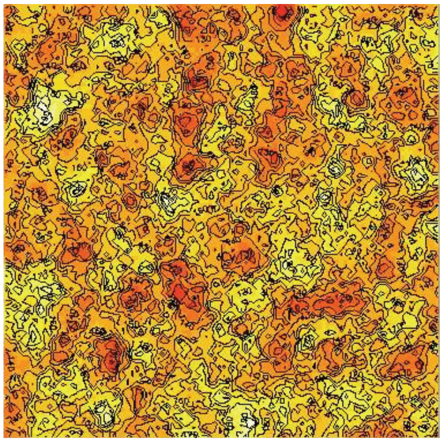


Figure 2.3: Systematic V_{th} variation map for a chip with $\phi = 0.1$.

Finally, to estimate the systematic component of L_{eff}, we proceed as follows. The ITRS report [36] tells us that the total σ/μ of L_{eff} is roughly half of that of V_{th}. Moreover, according to [9], the systematic component of L_{eff} is strongly correlated with the systematic component of V_{th}. Hence, we use the following equation to generate a value of the systematic component of L_{eff} given the value of the systematic component of V_{th}. Let L_{eff0} be the nominal value of the effective length and let V_{th0} be the nominal value of the threshold voltage. We use:

$$L_{\text{eff}} = L_{\text{eff0}} \left(1 + \frac{1}{2}(V_{\text{th}} - V_{\text{th0}})/V_{\text{th0}}\right) \tag{2.2}$$

2.2.2 Random Variation

The random variation occurs at a much finer granularity than the systematic variation; it occurs at the level of individual transistors, rather than at the level of millions of transistors. Hence, it is not possible to model random variation in the same explicit way as systematic variation — by simulating a grid where each cell has its own parameter values. Instead, random variation appears in the model analytically. Random components of V_{th} and L_{eff} are normally distributed with a σ_{rand} and a zero mean.

2.2.3 Combining Variations

Finally, since the random and systematic components of V_{th} and L_{eff} are normally distributed and independent, the total variation is normal with zero mean and a standard deviation of:

$$\sigma = \sqrt{\sigma_{rand}^2 + \sigma_{sys}^2} \tag{2.3}$$

where V_{th} and L_{eff} have a different σ.

2.2.4 Values for μ, σ and ϕ

For V_{th}, we set $\sigma/\mu = 9\%$. This is consistent with near-future technologies and includes both the systematic and random components. Moreover, according to empirical data gathered by [43], these two components are approximately equal for 32 nm technology. Hence, we assume that they have equal variances. Since both components are modeled as normal distributions, their standard deviations σ_{rand} and σ_{sys} are equal to $9\%/\sqrt{2} = 6.4\%$ of the mean. This value for the random component matches the empirical data of Keshavarzi *et al.* [44].

As explained before, we take the total σ/μ of L_{eff} to be half of that of V_{th}. Consequently, L_{eff}'s σ/μ is 4.5%. Furthermore, assuming again that the two components of variation are more or less equal, we have that σ_{rand} and σ_{sys} for L_{eff} are equal to $4.5\%/\sqrt{2} = 3.2\%$ of the mean.

To estimate ϕ, we note that Friedberg *et al.* [25] experimentally measured the correlation of gate length to be around half of the chip length. The rest of this work also adopts $\phi = 0.5$, but depending on how ϕ scales with die size, larger values may be appropriate for smaller dies.

2.3 Impact on Chip-Level Behavior

For an initial analytical evaluation of the impact of variation on a chip's behavior, we look at two key characteristics: chip leakage power and frequency.

2.3.1 Leakage Power

Subthreshold leakage is the main source of leakage in current and future technologies, especially now that the accelerated adoption of high-k gate dielectric is set to reduce gate leakage 100-fold [11]. The following subthreshold leakage model is based on that of

HotLeakage [86], itself a simplification of the full BSIM3 SPICE model:

$$I_{\text{leak}} \propto (kT/q)^2 e^{\frac{q(V_{off}-V_{th})}{(\eta kT)}} \tag{2.4}$$

where $V_{\text{th}} = Tc_1 + c_2$, k is Boltzmann's constant, and q the electron charge, while c_1, c_2, η and V_{off} are empirically determined parameters. We find the value for these parameters by fitting the leakage Equation 2.4 to experimental data for the 32 nm technology node obtained from SPICE simulations using the Predictive Technology Model [87].

The values for these parameters are given in Table 3.1 for both PMOS and NMOS transistors.

Transistor Type	Parameter Values for 32 nm
NMOS	$c_1 = -1.23mV/K$, $c_2 = 613mV$, $V_{off} = 134mV$, $\eta = 3.42$
PMOS	$c_1 = -1.14mV/K$, $c_2 = 544mV$, $V_{off} = 183mV$, $\eta = 3.43$

Table 2.1: Parameter values for Equation 2.4.

In order to estimate the impact of different levels of V_{th} variance on the chip's leakage power, we take our V_{th} distribution and integrate Equation 2.4 over all the transistors in the chip. The result is the total leakage current in the chip. Let P_{leak} and I_{leak} be the chip leakage power and current under V_{th} variation, and P^0_{leak} and I^0_{leak} be the same parameters when there is no variation. The expected value of the ratio of post-variation and pre-variation leakage is:

$$P_{\text{leak}}/P^0_{\text{leak}} = I_{\text{leak}}/I^0_{\text{leak}} = e^{(q\sigma/\eta kT)^2/2} \tag{2.5}$$

which implies that the increase in the chip's leakage power and current due to V_{th} variation depends on the standard deviation σ of V_{th}. Figure 2.4 plots the relative power as a function of σ. It increases rapidly as σ goes up.

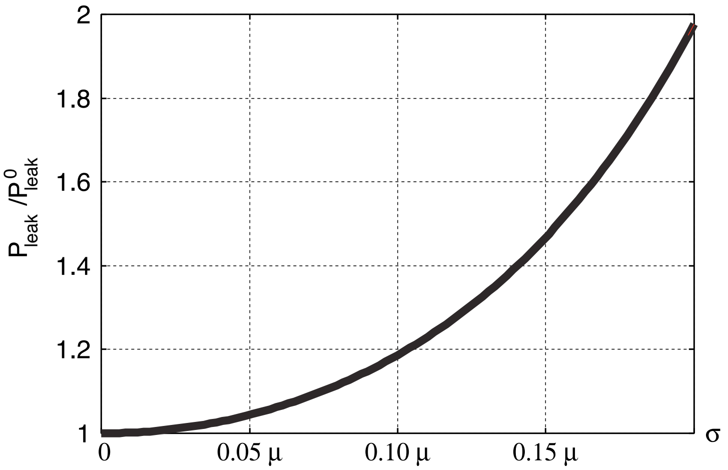


Figure 2.4: Relative leakage power in the chip as a function of V_{th}'s σ. V_{th0} is 0.150V at 100 °C.

Another important factor affecting leakage power is temperature. Figure 2.5 shows how the relative leakage power changes as a function of temperature, for different threshold voltages at 100 °C. Leakage power increases dramatically with temperature (3X from 50 °C to 100 °C). In addition, we observe that the leakage dependence on the threshold voltage is significant. For different V_{th} (different lines in Figure 2.5), the leakage changes significantly. For instance, for a 20% variation in the reference V_{th} at 100 °C, we see a 130% increase in leakage power.

2.3.2 Chip Frequency

The delay of an inverter gate is given by the alpha-power model [67] as:

$$T_g \propto \frac{L_{\text{eff}} V}{\mu (V - V_{\text{th}})^{\alpha}} \tag{2.6}$$

where α is typically 1.3 and μ is the mobility of carriers ($\mu(T) \propto T^{-1.5}$). As V_{th} decreases, $V - V_{\text{th}}$ increases and the gate becomes faster. As T increases, $V - V_{\text{th}}(T)$ increases, but $\mu(T)$ decreases [42]. The second factor dominates and, with higher T, the gate becomes

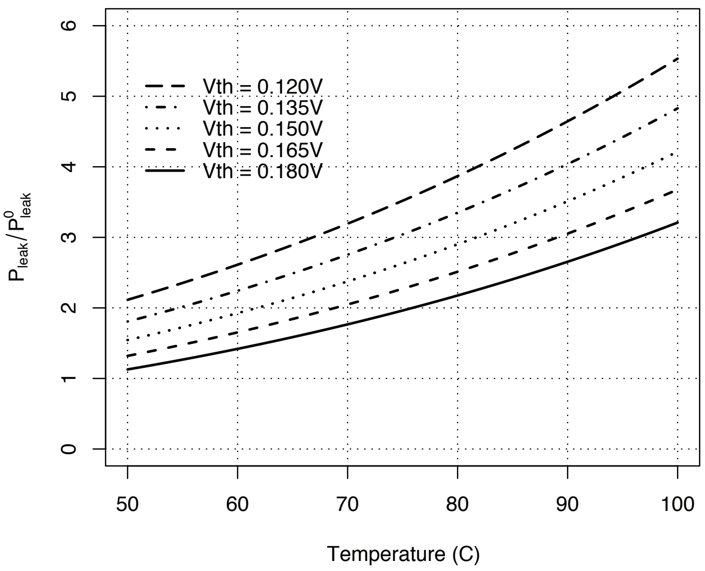


Figure 2.5: Relative leakage power versus temperature for different threshold voltages at 100 °C. We use V_{th0}=0.150V at 100 °C.

slower. Figure 2.6 plots the dependence between relative switching frequency and temperature as dictated by Equation 2.6. We can see that the dependence is not very strong.

Consider now a fixed temperature. Substituting Equation 2.2 into Equation 2.6 and factoring out constants with respect to V_{th} produces:

$$T_g \propto \frac{1 + V_{\text{th}}/V_{\text{th0}}}{(V - V_{\text{th}})^{\alpha}} \tag{2.7}$$

Empirically, we find that Equation 2.7 is nearly linear with respect to V_{th} for the parameter range of interest. Because V_{th} is normally distributed and a linear function of a normal variable is itself normal, T_g is approximately normal.

Assuming that every critical path in a processor consists of n_{cp} gates, and that a modern processor chip has thousands of critical paths, Bowman *et al.* [7] compute the probability distribution of the longest critical path delay in the chip ($\max\{T_{\text{cp}}\}$). Such path determines the processor frequency ($1/\max\{T_{\text{cp}}\}$). Using this approach, we find that the value of V_{th}'s σ affects the chip frequency.

Figure 2.7 shows the probability distribution of the chip frequency for different values of V_{th}'s σ. The frequency is given relative to a processor without V_{th} variation (F/F_0). The figure shows that, as σ increases, (i) the mean chip frequency decreases and (ii) the chip frequency distribution gets more spread out. In other words, given a batch of chips, as V_{th}'s σ increases, the mean frequency of the batch decreases and, at the same time, an individual chip's frequency deviates more from the mean.

We saw that V_{th}'s σ directly affects chip leakage and frequency. As σ increases, chip leakage increases rapidly, and chip frequency decreases in mean value and varies more. Therefore, V_{th} (and L_{eff}) variation is very detrimental.

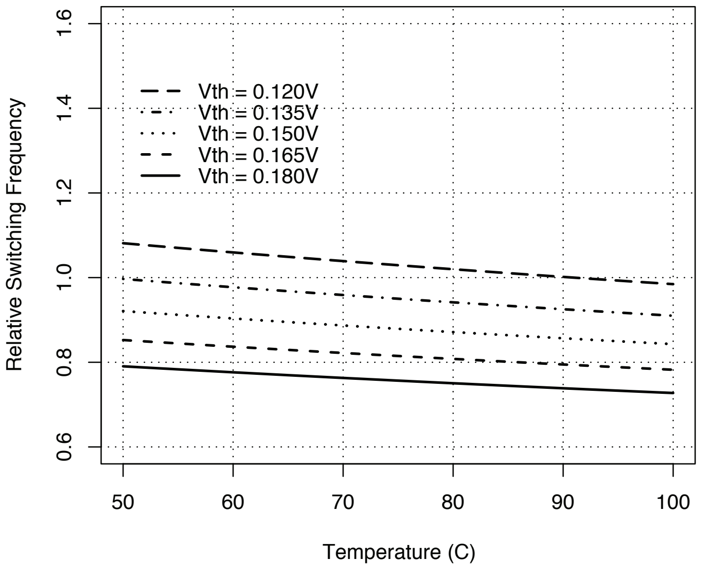


Figure 2.6: Relative switching frequency versus temperature for different threshold voltages at 100 °C. We use V_{th0}=0.150V at 100 °C.

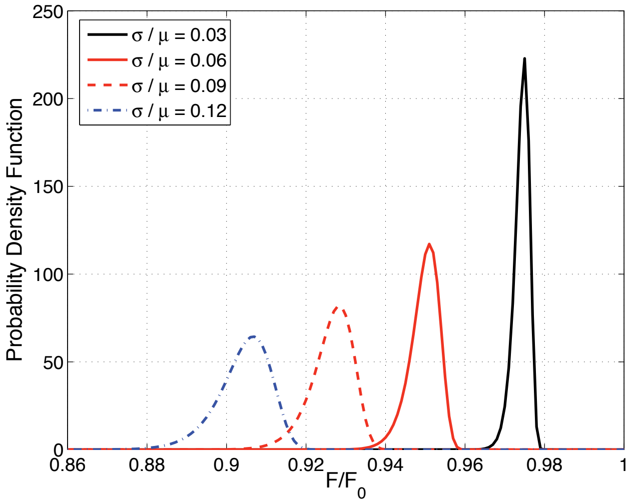


Figure 2.7: Probability distribution of the relative chip frequency as a function of V_{th}'s σ. We use V_{th0}=0.150V at 100 °C, 12 FO4s in the critical path, and 10,000 critical paths.

CHAPTER 3

Dynamic Fine-Grain Body Biasing

3.1 Introduction

As we have seen in Chapter 2, variation in V_{th} directly impacts two major properties of the processor, namely the frequency it attains and the leakage power it dissipates. Moreover, V_{th} is also a function of temperature, which increases its variability [78].

A recently-proposed technique to mitigate V_{th} variation within a chip is Fine-Grain Body Biasing (FGBB) [82]. FGBB applies different body biases to different sections of the chip, which we call *Cells*. A body bias is a voltage applied between the source or drain of a transistor and its substrate, effectively changing the transistor's V_{th} [78]. Depending on the polarity of the voltage applied, V_{th} increases or decreases. If it increases, the transistor becomes less leaky and slower; if it decreases, the transistor becomes leakier and faster. By reducing the V_{th} in cells with slow transistors and increasing the V_{th} in cells of leaky transistors, we reduce the variation within the die and attain a better frequency-leakage operation for the chip.

Previous work has proposed determining the body bias voltages at manufacturing time and setting them permanently for the lifetime of the chip. This means that the optimal values for the bias voltages have to be selected considering worst-case temperature and, therefore, delay conditions. This results in an overly-conservative configuration. In practice, the processor does not normally run at worst-case temperature and delay conditions. To take advantage of this, we propose to continuously adjust the body biases dynamically, adapting to changes in operating conditions. We call the scheme *Dynamic FGBB (D-FGBB)*.

The main contribution of this work is to introduce and evaluate D-FGBB. We show that D-FGBB is very versatile and significantly more effective than S-FGBB. Specifically, with the processor working in normal mode at fixed frequency, D-FGBB reduces the leakage power of the chip by an average of 28–42% compared to static FGBB — the higher savings corresponding to the cases with more body bias cells per chip. Alternatively, with the processor working in a high-performance mode, D-FGBB increases the processor frequency by an average of 7–9% compared to static FGBB — or 7–16% compared to no body biasing. We also show that D-FGBB can complement Dynamic Voltage Scaling (DVS) and that it scales well when combined with Dynamic Voltage and Frequency Scaling (DVFS).

3.2 Body Biasing

Body Biasing (BB) a transistor involves applying a voltage between its source or drain and substrate to alter its V_{th} [50]. In Forward BB (FBB), the voltage polarity is such that V_{th} decreases, creating a faster and leakier transistor. In Reverse BB (RBB), V_{th} increases, creating a slower, less leaky transistor. BB can be applied in a way such that the chip receives a single bias voltage or that it receives different bias voltages in different regions of the chip [82] — we call the latter Fine-Grain Body Biasing (FGBB). We call each of the regions with a different bias voltage a *Cell*.

3.2.1 Uses of BB and FGBB

At least two commercial processors use BB, namely Intel's Xscale [15] and Transmeta's Efficeon [19]. Both apply a single, chip-wide BB. Xscale uses RBB in standby mode to reduce leakage. There are fewer details on Efficeon, but it appears that the chip uses BB either to reduce leakage or to boost frequency. In addition, an experimental 80-core network-on-chip from Intel [84] uses FGBB. Specifically, it uses FBB to increase frequency in active mode and RBB to save leakage power in idle mode.

Another proposed use of BB is to reduce D2D process variation [6, 82]. After fabrication, different dies from the same batch run at different frequencies and leak different amounts. Applying different levels of chip-wide BB to different chips — RBB to high-leaking chips and FBB to slow ones — pushes the chips into a more homogeneous region of operation with acceptable frequency and leakage.

Other work has focused on using FGBB to mitigate WID variation [2, 12, 82]. Specifically, Tschanz *et al.* [82] implement FGBB on a test chip with 21 cells, each containing one critical path and circuitry to determine the optimal BB for the cell. Cells with a slow critical path are made faster with FBB, while cells with a fast (and leaky) critical path are made less leaky with RBB. The result is that WID variation in speed and leakage decreases.

3.2.2 Overhead of BB and FGBB

Implementing BB in a chip requires adding power lines for the BB voltage and including circuitry to determine and generate the optimal BB voltage [50]. In addition, to apply BB to NMOS, the manufacturing process has to be enhanced with a triple-well process [82]. There are three overheads to consider, namely area, power, and time.

The area overhead of BB is examined by Kuroda and Sakurai [50], who discuss various circuits to apply BB. The circuitry that controls the BB is simple, and its area overhead is estimated to be 1% of the chip area. On top of that, there is the area overhead of routing the power lines for BB. However, Narendra *et al.* [58] implement a router chip with BB for PMOS that contains a central bias generator, 24 local bias generators distributed in the chip with their own control circuits, and the needed global routing, and report a full-chip area overhead of 2%. Similarly, in an experimental 150nm FGBB chip, with 21 cells that contain one critical path each, Tschanz *et al.* [82] report an overall chip area overhead due to FGBB of 2-3%. Furthermore, an optimized design of BB circuits using recently-proposed approaches such as Chen and Gregg's [12] or Azizi and Najm's [2] may further reduce the area overhead.

Applying and controlling BB consumes some static and dynamic power. The static power dissipated is proportional to the area and, therefore, is small. The dynamic power consumed charging and discharging the substrate capacitance when BB levels change is small because the currents are small. Overall, according to Kuroda and Sakurai [50], the power overhead of BB and the circuitry that controls it is 1% of the chip's power.

The timing overhead of BB is also negligible. Kuroda and Sakurai [50] present designs that allow large changes in BB voltage to occur in the order of 1μs or 10μs. In this work, we only change the BB voltage in small increments when T changes, which is in the order of ms. Moreover, the processor does not stop while the BB voltage is being adjusted. Finally, Narendra *et al.* [58] report that the presence of the BB circuitry does not hurt the frequency of their router chip. Consequently, we assume there is no timing overhead.

Finally, determining the optimal amount of BB to apply can be done using a circuit that is representative of the critical paths in the cell. Using a phase detector similar to the one used in Razor [23] on that representative circuit, Tschanz *et al.* [82] determine the frequency that the transistors in that cell can support. Based on it, they set the BB to apply to the cell.

3.3 Dynamic Fine-Grain Body Biasing

Judicious application of FGBB can redress the problem of WID V_{th} variation. As suggested by Tschanz *et al.* [82], RBB can be applied to cells with low V_{th} and FBB to cells with high V_{th}. The net effect is to lower V_{th}'s σ. As a result, the chip may increase its frequency, reduce its leakage, or a combination of both.

While Tschanz *et al.* proposed to use FGBB statically, we propose to use FGBB dynamically. Moreover, our approach and goal are different than Intel's 80-core network-on-chip [84]. In the latter, active cores receive FBB to increase their frequency and idle cores receive RBB to reduce their leakage.

Our approach is different in two ways. First, we apply D-FGBB in a *fine time scale*, adapting it as an application runs and the *T* changes. Secondly, we are redressing parameter variation within a core. Our goal is different in that we want to run a core at the highest frequency and/or at the lowest power that can be attained *at any given time*. In this section, we propose a mechanism to apply D-FGBB and use it in different scenarios.

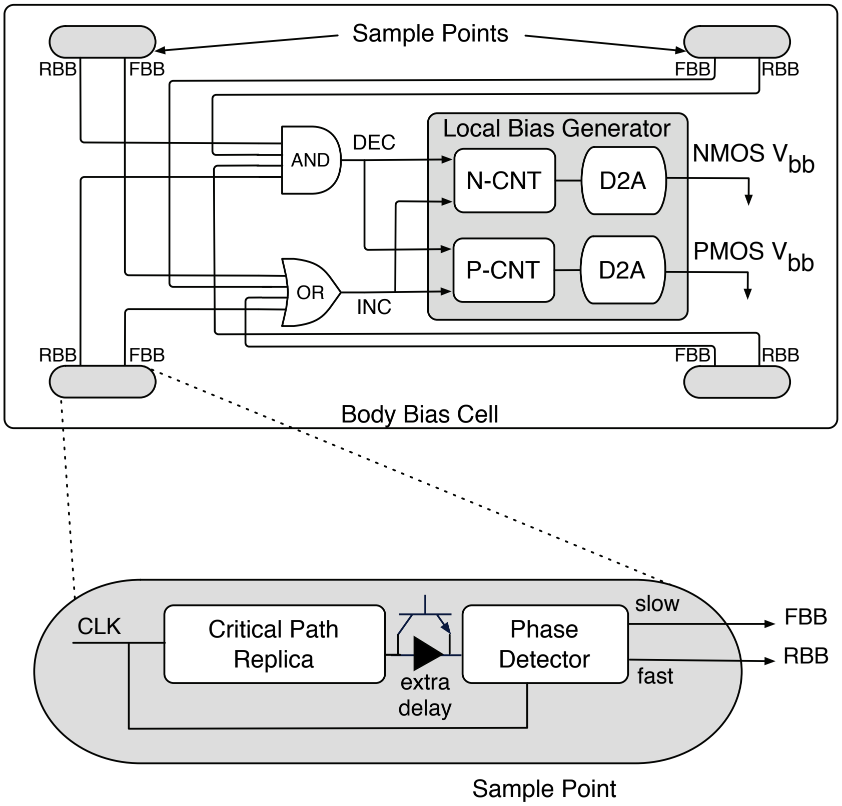


Figure 3.1: Proposed circuit to support D-FGBB in a cell.

3.3.1 A Mechanism to Apply D-FGBB

To implement FGBB in a chip, we divide it into cells that can be body-biased independently. In each cell, we add a Local Bias Generator, which is a simple circuit to generate the BB voltage. Then, we determine the optimal BB voltage that should be applied to each cell.

BB essentially trades off leakage power for delay. The optimal BB voltage is therefore the one that results in the minimum leakage consumption while ensuring that all the critical paths in the cell meet timing. The optimal BB voltage is therefore a function of the cell's critical path delays, which in turn are dependent on the V_{th} and T distributions.

Analytical solutions to determine the optimal BB voltage are not practical because of the non-determinism caused by V_{th} and T variation. We instead rely on direct measurement of critical path delay to determine the BB voltage to apply to each cell.

In [82], a dedicated control circuit estimates the delay of the transistors in the cell and adjusts the BB for the cell accordingly. The circuit consists of a critical path replica and a phase detector that recognizes when the critical path replica is not meeting the target frequency. A feedback mechanism is used to adjust the BB of the cell until the target frequency is met.

We modify that design to work for D-FGBB. A diagram of our circuit for a single cell is shown in Figure 3.1. We use multiple critical-path replicas distributed across the cell. This allows for a more accurate assessment of the cell's delay, in the presence of variation. Each critical path replica is paired with its own phase detector, forming what we call a Sample Point (Figure 3.1).

In cases of severe variation, it may happen that none of the critical path replicas captures the worst-case delay of the cell. This will be detected during normal testing of the chip. To solve this problem, we add some inverters to one of the critical path replicas of each cell (Figure 3.1). These inverters are normally bypassed by pass transistors. If a cell fails to meet the target timing during testing, some of the pass transistors in its corresponding critical path replica are enabled. This increases the delay of the critical path replica so that it becomes representative for that cell.

We use a bidirectional phase detector that identifies when the frequency supported by the critical path replica is noticeably higher than or not as high as current conditions. In the former case, it raises the RBB signal; in the latter, it raises the FBB signal (Figure 3.1).

This allows the circuit to fine-tune the BB voltage applied dynamically, by either increasing or decreasing it depending on the signal raised. It saves both time and power compared to the unidirectional calibration performed statically in [82] — which starts at the maximum RBB and gradually reduces it, finally applying FBB until the target frequency is met.

An alternative to using critical path replicas is to directly measure the delay of the actual critical paths as proposed in [18]. The critical paths in each cell are identified by the CAD tools and their inputs and outputs sampled after fabrication by a circuit similar to our phase detector. There are two advantages to this solution. First, it incurs a smaller area overhead because it does not need to replicate critical paths. Secondly, it can be more accurate because it measures the actual critical paths rather than replicas. The downside is that it is more intrusive to the hardware design.

Each cell has a local bias generator that generates separate BB voltages for NMOS and PMOS transistors (Figure 3.1). This is because NMOS and PMOS transistors can be affected differently by variation and have different optimal BB voltages. The BB values for PMOS and NMOS are stored in two bidirectional counters called P-CNT and N-CNT, respectively. The counters are incremented and decremented dynamically. Their initial values are set at a post-manufacturing calibration phase that determines the optimal BB values for NMOS and PMOS at a calibration temperature (Section 3.3.2). Thereafter, their values change only together by the same amount.

The counter values are converted to voltages by two digital-to-analog (D2A) converters based on a resistor ladder and an OP-AMP. By setting the appropriate reference voltages to the resistor ladders, and incrementing/decrementing the counters, the BB voltages range from a maximum RBB of -500mV to a maximum FBB of 500mV in 32mV steps. This has the effect of changing V_{th} by a range of $\pm$ 70mV.

The conditions for changing the BB are as follows. The counters are incremented as long as *at least one* of the critical path replicas asserts its FBB signal. This ensures that the cell receives higher BB until the slowest critical path replica meets timing. The counters are decremented as long as *all* the critical path replicas assert their RBB signals. This means that the cell receives lower BB only as long as all the critical path replicas are faster than the desired delay. This saves leakage power while meeting the target frequency. When neither of the above conditions is met, the counters hold their current values.

3.3.2 Static Calibration of FGBB

The chip manufacturer calibrates the FGBB for each chip. The goal is to bring each chip to its best frequency-leakage operating point before shipment. The calibration is performed in a controlled environment, at worst-case temperature conditions T_{cal} (for Calibration Temperature). This ensures that the chip will function properly at any *T*. At the same time, the manufacturer runs a set of test vectors designed to exercise the chip at full load and generate the maximum dynamic power dissipation.

This process first sets the initial values of the P-CNT and N-CNT counters in all cells. The goal is to correct the potential imbalance between the V_{th} of the NMOS and PMOS transistors. Using a simple circuit like the one proposed in [26, 59], each cell measures the local imbalance. Then, the cell increments the counter of the slower transistor type so that the imbalance is eliminated. For cells with no V_{th} imbalance, P-CNT and N-CNT remain at 0.

Next, the highest possible frequency at which the chip can run is determined using binary search. First an initial *Target Frequency* F_{cal}^{0} is selected and applied to the chip. F_{cal}^{0} can be a fixed percentage of the frequency expected for a chip with no V_{th} variation. In each cell, the phase detectors automatically time the critical path replicas and the local bias generator sets the optimal BB voltages for PMOS and NMOS. After the cells have been body-biased, the chip's total power is measured — both dynamic and leakage power. If the power is below a tolerable maximum value, F_{cal}^{0} can be increased; if it is above, F_{cal}^{0} must be decreased. Let us call F_{cal}^{1} the new frequency. The calibration process is then repeated for F_{cal}^1. This process may be repeated a few times, each time decreasing ΔF_{cal} until the highest possible frequency, subject to the power constraint, is found. Let us call the final value F_{cal}.

3.3.3 Using D-FGBB to Save Leakage Power

We propose to use our D-FGBB control circuit of Figure 3.1 to save leakage power by continuously adapting BB voltages as *T* changes — without changing the frequency. Recall that, as *T* goes up, gate delay goes up (Section 2.3). As a result, a chip's critical path delays also increase. The static FGBB (S-FGBB) settings are necessarily conservative because they are calibrated using the conservatively high T_{cal}. In reality, there is a significant *T* variation across and within workloads.

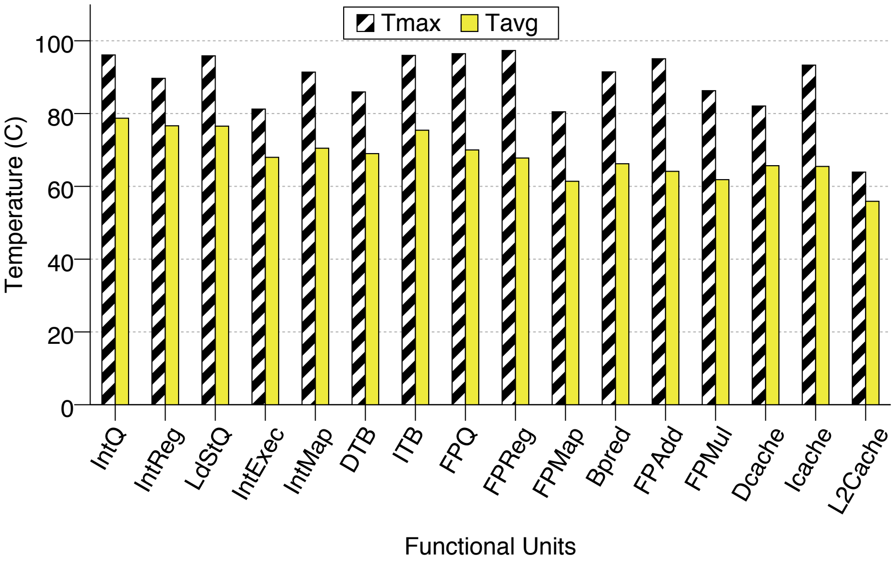


Figure 3.2: T_{max} and T_{avg} in different units of a processor running a sequence of SPECint and SPECfp codes.

To illustrate it, Figure 3.2 shows, for each unit in a processor, the T_{max} and T_{avg} of that unit when running a sequence of SPECint and SPECfp codes. The difference between T_{max} and T_{avg} is often 20-30 °C. The initial S-FGBB calibration is performed at a T_{cal} that is higher than the highest T_{max}, while units typically operate at close to T_{avg}. As a result, critical paths are generally faster than during calibration, and we can reduce the BB applied and save leakage.

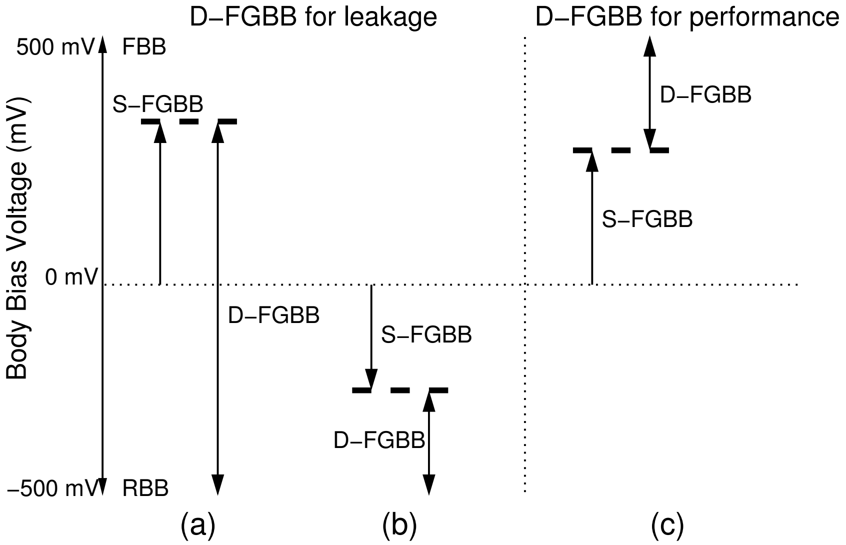


Figure 3.3: Changing the BB voltage with D-FGBB.

When the chip is first powered-up, each cell's BB is set to its S-FGBB calibration value. As the chip workload changes, our D-FGBB control circuit adjusts the BB for each cell to the optimal value for the current *T*. This is done without changing the frequency. Figures 3.3(a) and (b) show the BB voltage values under S-FGBB and D-FGBB. In (a), the cell is initially slow, and S-FGBB applies FBB at calibration time. Then, D-FGBB can dynamically reduce the FBB and even apply RBB to save leakage power. Figure 3.3(b) shows the case when the cell is initially fast enough to meet the target frequency, and S-FGBB applies RBB at calibration time. D-FGBB can still save additional leakage by applying further RBB when conditions permit.

D-FGBB dynamically adjusts the BB voltages without stopping the running application. Consequently, it induces no time overhead.

3.3.4 Using D-FGBB to Improve Performance

A second use of D-FGBB is to increase performance by adapting frequency and BB voltages as power consumption changes. This approach is used when the user wants to run the processor in a *high-performance* mode, where the goal is to deliver the highest possible performance while staying within the chip's power budget (P_{max}) at all times.

The insight that enables this mode of operation is that the manufacturer determines the chip's frequency F_{cal} conservatively, assuming a worst-case power consumption P_{max} — including worst-case dynamic power consumption — in addition to worst-case T_{cal}. Since he assumes the maximum dynamic power, he imposes a conservative limit on the leakage power — such that when both are added together, P_{max} is not exceeded. At run time, such maximum dynamic power is not always reached. Consequently, as long as P_{max} is not exceeded, we can dynamically increase the frequency beyond F_{cal} — which will increase the dynamic power and, at the same time, require our D-FGBB circuit to increase the BB of cells. The approach is shown in Figure 3.3 (c).

To support this mode of operation, we extend the S-FGBB calibration process. Specifically, recall that we calibrated the chip's F_{cal} under full load, generating the maximum dynamic power dissipation (Section 3.3.2). After this, we place the chip in idle mode, to dissipate little dynamic power, and repeat the calibration. Because the dynamic power is low, more leakage power can be expended. This in turn allows higher BB in the cells, enabling a higher processor frequency. The resulting frequency (F_{cal}^{max}) is the absolute maximum frequency that the chip's circuits can meet while not exceeding P_{max}. F_{cal} and F_{cal}^{max} are recorded in a programmable table on-chip; they are used as lower and upper bounds, respectively, on the processor frequency in high-performance mode.

At run time, the processor starts at F_{cal}. As it runs under load, it adjusts its frequency at regular intervals, taking values between F_{cal} and F_{cal}^{max}, depending on the current power consumption of the chip. We assume that the chip includes circuits to measure average power, possibly like those in Itanium's Foxton [55]. As long as the average power is less than P_{max} and the processor is under load, the frequency is increased. To meet the new frequency, the D-FGBB control circuit quickly increases the BB levels. Safety mechanisms are in place to ensure that P_{max} or T_{cal} are not exceeded. If this is about to happen, the frequency is reduced.

3.3.5 D-FGBB and Dynamic Voltage Scaling

While D-FGBB trades-off circuit delay for leakage power, Dynamic Voltage Scaling (DVS) largely trades-off circuit delay for dynamic power. Consequently, we can combine both techniques so that, for a given frequency of operation — e.g., F_{cal} — the processor consumes less power than with either technique alone. Previous work has shown that BB can complement DVS to improve the power savings of DVS alone [54]. However, that work decided the optimal combination of techniques using an analytical expression, which is not suitable in the presence of variation.

A given circuit delay can be obtained with different combinations of supply voltage and BB values, each with a different power cost. In some cases, more power can be saved with a lower supply voltage (saving dynamic power) and a higher BB to compensate for the circuit slowdown (consuming more static power). In other cases, it is better to have a higher supply voltage (consuming more dynamic power) and use up the time slack with a lower BB (saving leakage power). The best approach depends on the fraction of power dissipated of each type and on the *T*.

We propose to augment the S-FGBB calibration process of Section 3.3.2 with one additional step to find a configuration that substantially reduces the power consumed at F_{cal}. Specifically, after the manufacturer has set the BB voltages for each cell at F_{cal}, he proceeds as follows. The supply voltage is reduced in small increments through all the voltages that are supported. At each step, our D-FGBB circuit of Figure 3.1 recomputes the BB values, and the total power in the chip is also measured. When the voltage drops so much that F_{cal} can barely be met, the process stops. Then, we select the combination of supply voltage and BB values that consumes the least power. If the processor has multiple DVS domains (e.g., one for the core and one for the L2), this algorithm is first run reducing the voltage of one domain only. Once the best configuration is found, the configuration is used to run the algorithm reducing the voltage of another domain, and so on. This process will find the optimal combination of supply voltage and BB values at the calibration temperature T_{cal}.

3.4 Selecting the BB Cells

Previous work [57] has shown that microarchitectural structure plays an important role in deciding how to partition the chip into BB cells. There are advantages to using BB cells with shapes that follow the contour of microarchitectural modules such as caches, registers, or execution units when compared to simply partitioning the chip in a uniform grid of cells. There are two main reasons for this, namely variations in *T* and differences in the types of critical paths in different modules.

3.4.1 Temperature Effects

Equations (2.4) and (2.6) show that *T* significantly affects transistor leakage and gate delay. At high *T*, transistors become vastly leakier and gates slower. As a result, the BB voltage applied can be better targeted if *T* does not vary much within a cell. It is well known that the spatial *T* profile in a chip under load follows the layout of microarchitectural modules. For example, the execution unit is hot while the L2 cache is cold. Consequently, we propose organizing the chip into cells that follow the contours of groups of hot and groups of cold microarchitectural modules.

3.4.2 Critical Paths in Logic and Memories

Different microarchitectural modules have different types of critical paths. This is most obvious when comparing logic blocks such as functional units to memory structures such as the L1 cache or TLB. In the former, a critical path contains many, physically close gates and a modest amount of wire — e.g., 8-16 FO4-equivalent gates in high-end processors connected by short wires. In contrast, the critical path in memory structures has a few, physically separated transistors and much more wire — e.g., the path that stretches from a driver through a word line, a pass transistor, a bit line, and then to a sense amplifier.

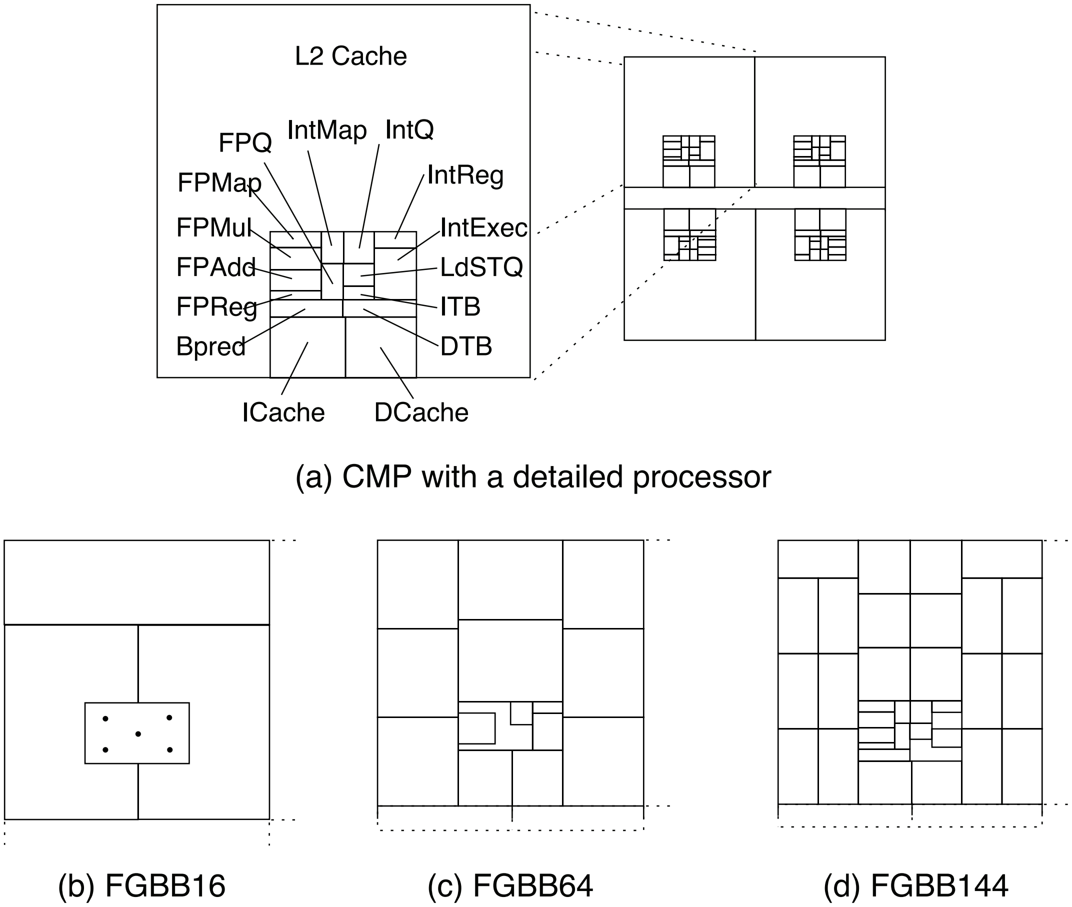
From a V_{th} variation point of view, these two critical paths differ dramatically. The transistors in a logic path are many and physically close. Their large number enables a better averaging of random V_{th} variations, while physical proximity makes them subject to the same systematic V_{th} variation. On the other hand, the transistors in the memory path are few and distant from each other. Fewer transistors means less averaging of random V_{th} variations, while farther distances implies better averaging of systematic V_{th} variations. Since these two types of critical paths are affected differently by a given BB voltage, we separate logic and memory structures into different BB cells.

3.5 Evaluation Methodology

3.5.1 Processor Chip Architecture

We use detailed simulations using the SESC [65] cycle-accurate simulator to evaluate a chip multiprocessor (CMP) with four high-performance processors at 45nm. The processor is based on the Alpha 21364, and has a 64KB L1 I-cache, a 64KB L1 D-cache, and a 2MB L2 cache. We estimate a nominal frequency of 4GHz with a supply voltage of 1V. We generate the processor layout from the Alpha 21364 chip floor-plan, without the router and I/O pads, and with an L2 cache as in [69]. We use constant scaling to scale the dimensions to 45nm.

Finally, we put four such units on a chip, and interconnect them with a wide snoopy bus. The resulting 8MB L2 cache is shared by all the cores. The resulting 132 mm^2 chip is shown in Figure 3.4(a).



(a) CMP with a detailed processor

(b) FGBB16

(c) FGBB64

(d) FGBB144

Figure 3.4: CMP floor-plan used (a) and the partitioning of one processor and its share of the bus into BB cells (b–d). Chart (b) shows the five critical path replicas in one cell.

3.5.2 Power and Temperature Model

To estimate power, we scale the results given by popular tools using technology projections from ITRS [37]. Specifically, we use SESC augmented with dynamic power models from Wattch [8] to estimate dynamic power at a reference technology and frequency. In addition, we use HotLeakage [86] to estimate leakage power at the same reference technology. Then, we obtain ITRS's scaling projections for the per-transistor dynamic power-delay product, and for the per-transistor static power. With these two factors, given that we keep the number of transistors constant as we scale, we estimate the dynamic and leakage power for the scaled technology and the frequency relative to the reference values.

We use HotSpot [69] to estimate the on-chip *T* profile. To do so, we use the iterative approach of Su *et al.* [76]: the *T* is estimated based on the current total power; the leakage power is estimated based on the current *T*; and the leakage power is added to the dynamic power. This is repeated until convergence. In our experiments, the maximum temperatures reached in the chip are in the 95-100 $^{\mathrm{o}}$C range.

We run several applications from SPECint (*bzip2, crafty, gap, gzip, mcf, parser, twolf*) and from SPECfp (*applu, equake, mesa, mgrid, swim*). A workload consists of running four instances of the same application at a time, one on each core. We use the reference input set for 1B instructions after discarding the first 1B instructions.

3.5.3 Critical Path Model

We do not have access to detailed information on the structure and distribution of a processor's critical paths. For this reason, we build a simple model that we use in our experiments. Specifically, we design different critical paths for logic modules, small memories, and large memories. For logic modules, we model a critical path as 12 FO4 gates connected in series by short wires. The wires account for 35% of the path delay [31]. We use CACTI [77] to estimate wire delays and Equation 2.6 to compute gate delays. For memory modules, we separate large memories (the two L1 caches) from the remaining SRAM structures (e.g., the register file). The latter are assumed to cycle at twice the frequency of the former. We use CACTI to determine the optimal sub-array sizes and the physical layout. In both structures, a critical path stretches from a driver driving a word line, through the word line, a pass transistor, the bit line, and the sense amplifier. We model the path as three logic gates connected by word- and bit-line wires laid out as per CACTI.

We model each logic module and each memory module as having many, spatially-distributed critical paths. Specifically, we use Bowman *et al.*'s estimate that a high-performance processor chip at our technology has about 10,000 critical paths [7]. We distribute these paths uniformly on the area taken by the cores and L1 caches — we assume that the L2 and the bus do not have critical paths. Each module gets critical paths of its type. Finally, as we superimpose the V_{th}, L_{eff}, and T variation maps on the chip, parameter variation impacts the delay of these paths. The frequency supported by a module is determined by the slowest of its critical paths.

3.5.4 Variation Model Parameters

We only model WID variation. Table 3.1 shows some of the parameter values used. For V_{th}'s σ/μ, the 1999 ITRS [36] gave a design target of 0.06 for year 2006 (although no solution existed); however, the projection has since been discontinued. On the other hand, Kahng [40, 41] reckons that the ITRS variability projections (for at least the gate-length parameter that he examines) are too optimistic. Consequently, we use a default V_{th}'s σ/μ of 0.12, which we vary in some experiments. Moreover, according to [43], the random and systematic components are approximately equal. Hence, we assume that they have equal variances. This means that, using Equation 2.3, $\sigma_{sys} = \sigma_{rand} = \sigma/\sqrt{2}$.

To set V_{th}'s ϕ, we note that Friedberg *et al.* [25] found that the gate length had a correlation range close to 0.5 of the chip's width. Since the systematic component of V_{th}'s variation directly depends on the gate length's variation, we use a default ϕ = 0.5 for V_{th}.

As indicated in Section 2.2.4, based on the 1999 ITRS [36], we set L_{eff}'s σ/μ to 0.5 of V_{th}'s σ/μ. Moreover, for L_{eff}, we also assume that $\sigma_{sys} = \sigma_{rand} = \sigma/\sqrt{2}$ and ϕ = 0.5.

Each individual experiment uses a batch of 200 chips that have a different V_{th} (and L_{eff}) map generated with the same μ, σ, and ϕ. To generate the per-chip V_{th} and L_{eff} maps, we

Parameter Values
Tech: 45nm; Nominal frequency: 4GHz; V_{dd}: 1V; T_{cal}: 100 °C
V_{th}: μ: 150mV at 100 °C; σ/μ: 0.12; $\sigma_{sys} = \sigma_{rand} = \sigma/\sqrt{2}$; ϕ: 0.5
L_{eff}: σ/μ: 0.5 $\times V_{\text{th}}$'s σ/μ; $\sigma_{sys} = \sigma_{rand} = \sigma/\sqrt{2}$; ϕ: 0.5
Body bias application: Maximum bias: ±500mV; Resolution: 32mV
Number of FGBB cells per chip: 16, 64, or 144
Number of critical path replicas per cell: 5
Number of chips per experiment: 200

Table 3.1: Parameter values used.

use the geoR statistical package [66] of R [64]. We use a resolution of 1M grid points per chip. To relate BB to V_{th}, we use the nonlinear formula from [78] that takes into account short-channel effects.

3.5.5 BB Environments Evaluated

We evaluate chips with FGBB applied at different granularities, from the trivial case that has a single BB cell (*FGBB1*), to environments with 16, 64 and 144 BB cells per CMP chip (*FGBB16, FGBB64, and FGBB144*). When partitioning a chip into cells, we first separate groups of hot units from groups of cold ones. Then, in each group, we separate logic, large memories, and small memories (Figures 3.4(b) to (d)). A large module like the L2 cache is broken into multiple cells. Each cell has five uniformly-spaced critical path replicas (Figure 3.4(b) shows them for one cell). The slowest of such replicas determines the cell's BB voltage.

We consider three different scenarios: FGBB set statically (*S-FGBB*), FGBB set dynamically as the chip runs (*D-FGBB*), and no BB applied (*NoBB*). As a reference, we also consider chips with no process-induced V_{th} variation (*NoVar*). Note that *NoVar*'s V_{th} is not constant due to T variation. For our DVS experiments, we use one DVS domain per processor and one for the L2 cache.

In D-FGBB, the BB voltage changes infrequently because it tracks gate delay changes due to *T*. A BB update occurs when the delay changes by $\approx$2%, which corresponds to a *T* change of $\approx$5 ^{o}C. We assume that the circuit in Figure 3.1 can detect such delay changes. Otherwise, we can use more elaborate circuits, which have high accuracy [27, 47]. In our D-FGBB simulations, we recompute the BB every 2ms.

3.6 Evaluation

We first assess the effect of V_{th} variation on frequency and leakage. Then, we focus on how D-FGBB improves on S-FGBB in three scenarios: normal operation, high performance, and low power.

3.6.1 Characterizing Variation

Figure 3.5 shows chip frequency (a) and chip leakage power (b) as V_{th} variation (measured in σ/μ) changes. For each value of σ/μ, the figure shows bars for three different ϕ. In all cases, frequency and leakage are relative to the *NoVar* chip. The bands in the bars show the variation across chips in the batch.

As V_{th} variation increases, the average frequency of the chips decreases and their average leakage power goes up. On average, at 0.12 variation and ϕ=0.5, the frequency is 10% lower and the leakage over 20% higher. Clearly, variation is undesirable. The long bands show high variation across chips in the batch. This is due to the *T* variation. At high *T*, a transistor becomes slower and leakier. Consequently, if transistors with very high V_{th} happen to "fall" on the hottest region of the chip, the chip is likely to have low frequency. On the other hand, if many transistors with very low V_{th} fall on the hottest area, the chip is likely to have high leakage.

We see two main trends. First, across chips in one experiment, leakage varies more than frequency — since leakage is exponential with *T*, an unfavorable V_{th} distribution can sig-

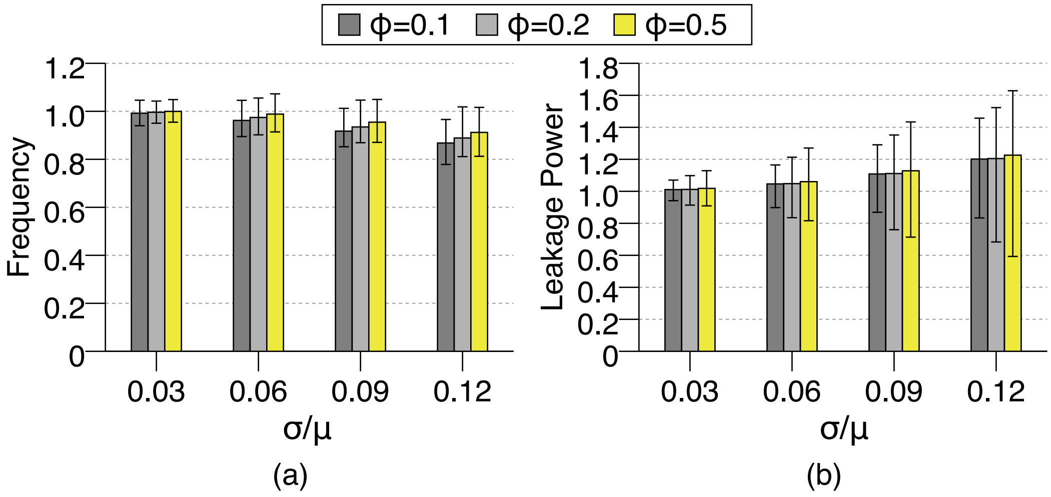


Figure 3.5: Impact of V_{th} variation on the chip's frequency (a) and leakage power (b).

nificantly increase leakage power. Second, as ϕ decreases, the average frequency decreases as well. The reason is that, given a more uniform distribution of high-V_{th} transistors across each die (low ϕ) there is a higher chance that some will fall on the hottest region of the chip. Transistors that have high-V_{th} and are also very hot are very likely to be the slowest on the die and therefore likely to impact the maximum frequency of a chip. On the other hand, the probability that the chips will have no high-V_{th} transistors in the hot area increases with ϕ. Therefore, for larger ϕ's the average frequency of the chips is higher.

3.6.2 Normal Operation: D-FGBB Improves a Chip's Operating Point

S-FGBB can be used to tune the chips in a batch so that they fall into desirable frequency-leakage bins [82]. The goal is to place each chip at the highest possible frequency bin where it still meets the power consumption constraint. In this section, we summarize the impact of S-FGBB and then show how D-FGBB further improves a chip's operating point.

The Acceptable Region for a chip [82] is bounded by two conditions: (i) the frequency should be higher than a given minimum value, and (ii) the sum of dynamic and leakage power should be less than a given maximum value. In a frequency-leakage plot such as Figure 3.6(a), these constraints require that the chip be above a horizontal line and to the left of a slanted line, respectively. The slanted line has this shape because, as frequency increases, the dynamic power increases linearly and, therefore, the amount of tolerable leakage power decreases linearly. Inside the Acceptable Region, higher frequency is better.

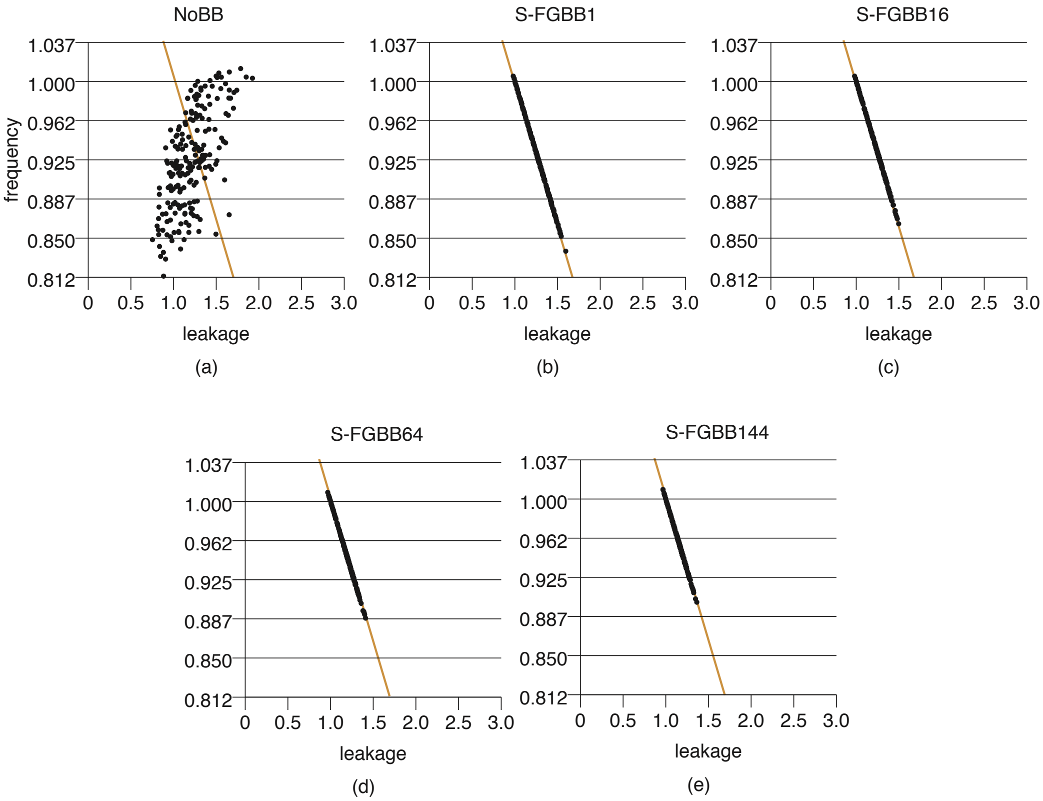


Figure 3.6: Frequency versus leakage power for a batch of 200 chips at T_{cal} and full load under various schemes.

Figure 3.6(a) shows a scatter plot of the frequency and leakage power for our 200 chips, with axes normalized to *NoVar* (no process-induced V_{th} variation). We build the slanted line so that it would include the *NoVar* chip, which is point (1,1). We then arbitrarily set the horizontal line to 0.85 of the frequency of the *NoVar* chip, and divide the range into four equally-spaced frequency bins. As a fraction of the *NoVar* frequency, the ranges of the bins are: 0.850–0.887, 0.887–0.925, 0.925–0.962, and over 0.962. These bins are in the ballpark of those used in commercial processors.

3.6.3 Impact of S-FGBB

In Figure 3.6(a), some chips fall outside the Acceptable Region. By applying S-FGBB to a chip, we can move it into the Acceptable Region or, if it is already there, move it to a higher frequency point. Using the axes and the slanted line of Figure 3.6, Figure 4.3.1 graphically shows the impact of our S-FGBB calibration algorithm of Section 3.3.2.

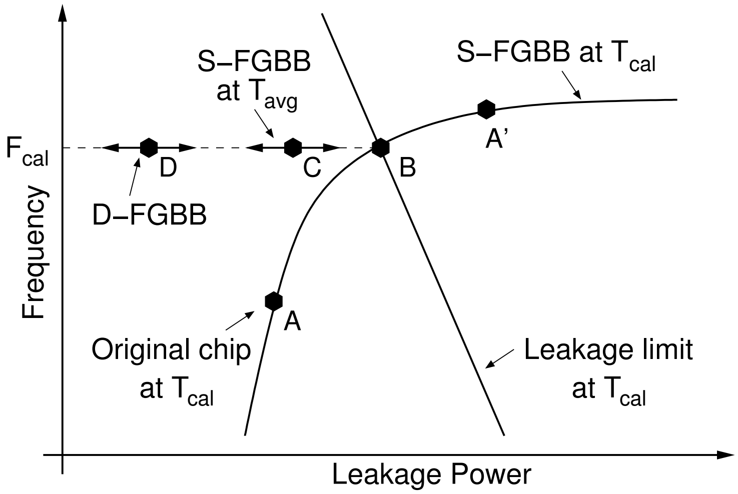


Figure 3.7: Impact of S-FGBB and D-FGBB on a chip's operating point.

Consider a chip that is originally operating at point *A*. Our algorithm can move the chip along the curve labeled *S-FGBB at T_{cal}*. The result of the algorithm is to bring the chip to point *B*, at frequency F_{cal}, where the chip dissipates the maximum allowed power — thus, point *B* is on the slanted line. Point *B* is more desirable than *A* in that it is inside the Acceptable Region and is potentially in a higher frequency bin than *A*. Increasing the frequency beyond F_{cal} would push the chip to the left of the slanted line, where power consumption is excessive. In cases where the original chip is operating at point *A'*, the S-FGBB algorithm reduces the frequency and brings it to point *B*.

The actual curve followed from *A* depends on the number of FGBB cells. The schemes with more cells such as FGBB144 target their BB voltages better and push the chip to a *B* position that is higher in the slanted line — thus delivering chips in better bins.

To show it, we take the batch of chips of Figure 3.6(a) and apply our S-FGBB algorithm using the FGBB1, FGBB16, FGBB64, and FGBB144 schemes. The resulting frequency-leakage scatter plots are shown in Figures 3.6(b)-(e). The charts show that all the schemes move practically all the chips to the slanted line, in the Acceptable Region. However, the schemes differ in how high they push the chips. The more BB cells they use, the more effective they are.

The different impact of the schemes is best seen in Figure 3.8, which shows how many chips fall in each frequency bin for the different schemes as a fraction of the 200 chips. Chart (a) corresponds to our experiment, while (b) repeats it for V_{th}'s $\sigma/\mu = 0.09$.

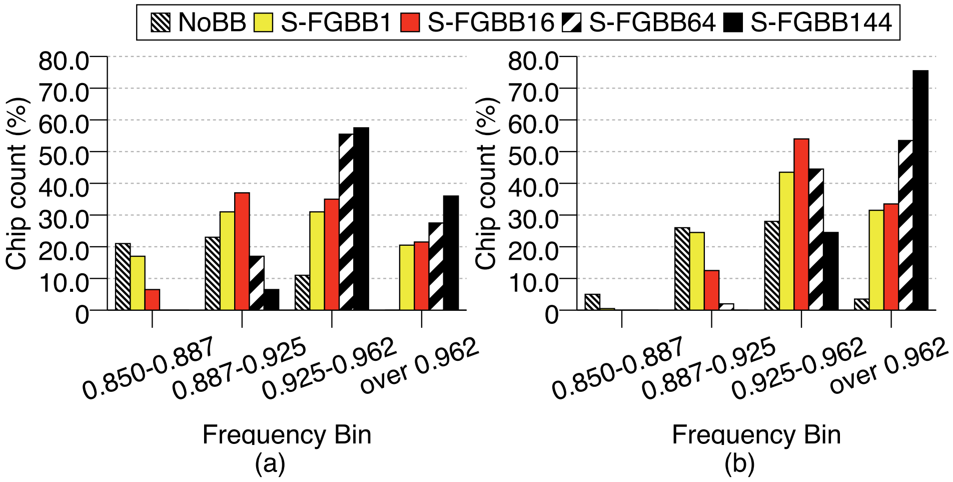


Figure 3.8: Frequency binning obtained by S-FGBB with different numbers of BB cells, for $\sigma/\mu = 0.12$ (a) and $\sigma/\mu = 0.09$ (b).

Figure 3.8(a) shows that FGBB64 and FGBB144 move many chips to the top bin. Specifically, FGBB144 has 36% of the chips in the top bin and 93% in the top two. On the other hand, NoBB has none in the top bin and only 11% in the top two. Chart (b) shows that the trends are the same for $\sigma/\mu = 0.09$. Specifically, as we go from NoBB to FGBB144, the number of chips in the top bin changes from 4% to 75%. Consequently, our results are valid for smaller variations as well.

In the rest of the evaluation, when we refer to the average frequency and leakage of the NoBB or other schemes, we count all the chips — rather than dropping from the average those that fall outside the Acceptable Region. While in a practical environment they would be dropped, we feel the results are more intuitive this way.

3.6.4 Leakage Reduction with D-FGBB

Applying the D-FGBB algorithm of Section 3.3.3 can substantially reduce the leakage power consumed by the chip. To see it graphically, consider Figure 4.3.1. The chip was calibrated with S-FGBB at T_{cal}, resulting in point *B*. However, given that the chip's *T* during execution is close to T_{avg}, the chip typically operates around point *C*, moving to the left and right as shown depending on the current *T* conditions. If we apply D-FGBB, we push the chip's working point to moving around point *D* in the figure. The result is leakage power savings.

Figure 3.9(a) compares the leakage power of the chips under NoBB, and with 1, 16, 64, or 144 cells under S-FGBB and D-FGBB. We report the average across all the applications and normalize the bars to NoBB. We see that D-FGBB reduces the leakage substantially over S-FGBB. Specifically, with D-FGBB, the leakage power is reduced by 28–42% compared to S-FGBB — where the highest reductions correspond to the chips with more cells. In all cases, S-FGBB dissipates about the same amount of leakage power as NoBB.

Figure 3.9(b) shows the total power in this experiment. The figure also includes an environment with DVS alone and one where D-FGBB is combined with DVS as detailed in Section 3.3.5. All bars are normalized to NoBB. Recall that, as we increase the number of cells, the frequency increases. However, for the same number of cells, the frequency is the same. From the figure, we see that D-FGBB reduces the total power consumption by 15–22% relative to S-FGBB for the same frequency, with the higher reductions corresponding to the schemes with more cells. If we combine D-FGBB and DVS, the total power saved

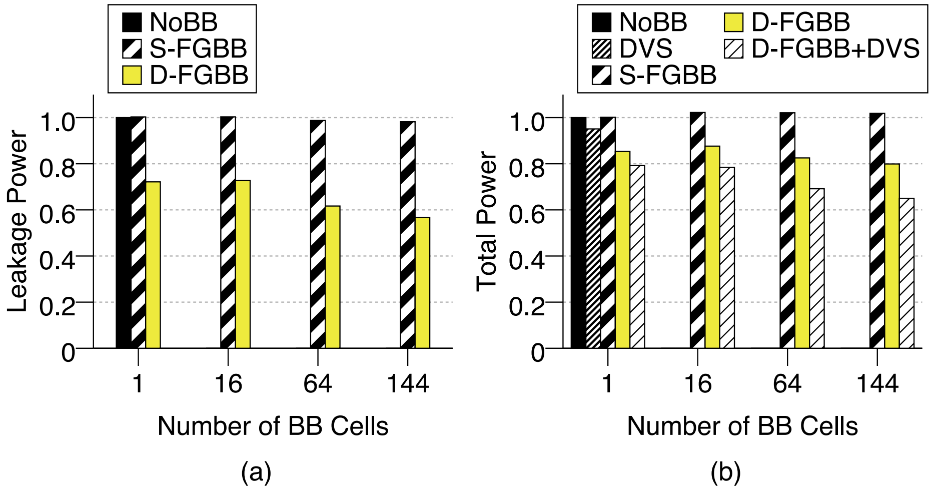


Figure 3.9: Leakage (a) and total power (b) of the chips for different FGBB schemes in normal operation.

is 21–36% of the S-FGBB power — again, with the schemes with more cells doing the best. This large impact is possible because DVS lowers the voltage of the domain that dissipates the most dynamic power (namely, the core), while D-FGBB applies higher BB to ensure that the target frequency is met. This results in dynamic power savings that add to the leakage savings of D-FGBB. Finally, DVS alone can only reduce less than 5% of the power in NoBB. This is because the voltage can be lowered little while still meeting the target frequency.

3.6.5 High Performance: D-FGBB Improves Frequency

A second application of D-FGBB is to improve performance by increasing the average frequency of a chip beyond the F_{cal} determined at calibration (Section 3.3.4). Figure 3.10 compares the average frequency of the chips with S-FGBB and this use of D-FGBB. The figure considers chips with different numbers of cells, and normalizes the bars to NoBB. We see that D-FGBB increases the frequency by 7–9% over S-FGBB for the same number of cells. Compared to NoBB, the frequency increase is 7–16%. With more cells, the frequency is higher because BB can be tuned better.

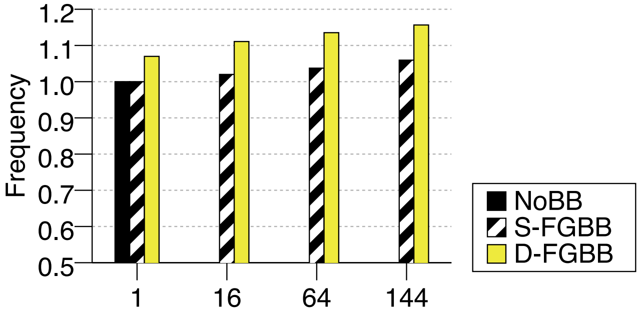


Figure 3.10: Average frequency of the chips for different FGBB schemes.

The frequency increase varies across applications, depending on their dynamic power consumption. Those with low dynamic power consumption see the biggest boosts in frequency. However, applications benefit differently from a frequency boost, depending on whether they are memory- or compute-intensive. Figure 3.11 compares the execution time of the applications with S-FGBB144 and D-FGBB144. In the figure, the bars are normalized to NoBB. On average, D-FGBB144 reduces the execution time by 6% over S-FGBB144. Moreover, compared to NoBB and S-FGBB1 (not shown in the figure), the reduction is 10%.

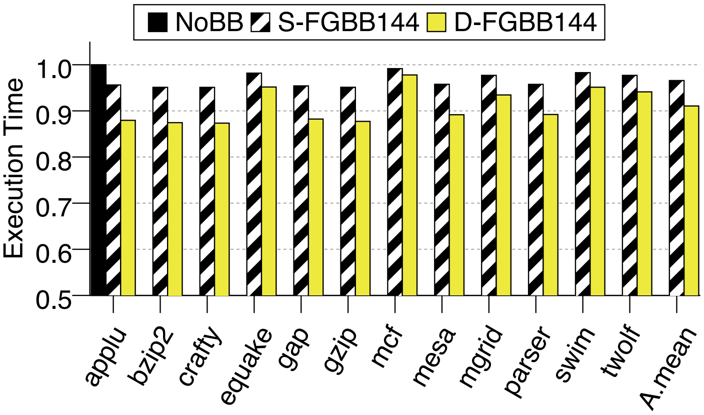


Figure 3.11: Execution time of the applications for different FGBB schemes.

The speedups delivered by D-FGBB come at a significant cost in total power consumption. Increasing the frequency induces higher dynamic power; applying the more aggressive BB voltage needed to increase frequency induces higher leakage power. The resulting total power for S-FGBB and D-FGBB is shown in Figure 3.12. Because of the high power cost, this mode of operation is only appealing when the highest possible performance is needed.

3.6.6 Low Power: D-FGBB Reduces Leakage

Finally, we consider an environment where we do not attempt to improve the original frequency of the chip with the S-FGBB calibration step of Section 3.3.2. Instead, we take each chip in the batch in turn, identify the frequency at which it runs, and then apply D-FGBB (or S-FGBB) to save leakage. Our goal is to save as much leakage as possible. We call this environment *low power* mode.

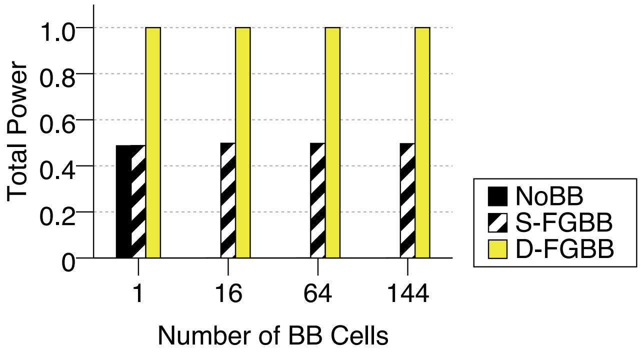


Figure 3.12: Total power of the chips for different FGBB schemes.

First, we look at the case when the frequency of the chip does not change. The result is shown in Figure 3.13. In Figure 3.13(a), we repeat the frequency-leakage scatter plot of Figure 3.6(a), this time at usual *T* and load conditions. As a result, the leakage power is significantly lower than in the worst case presented in Figure 3.6(a). Then, Figures 3.13(b)-(e) show the result of applying S-FGBB or D-FGBB with different numbers of cells, to reduce leakage at constant frequency.

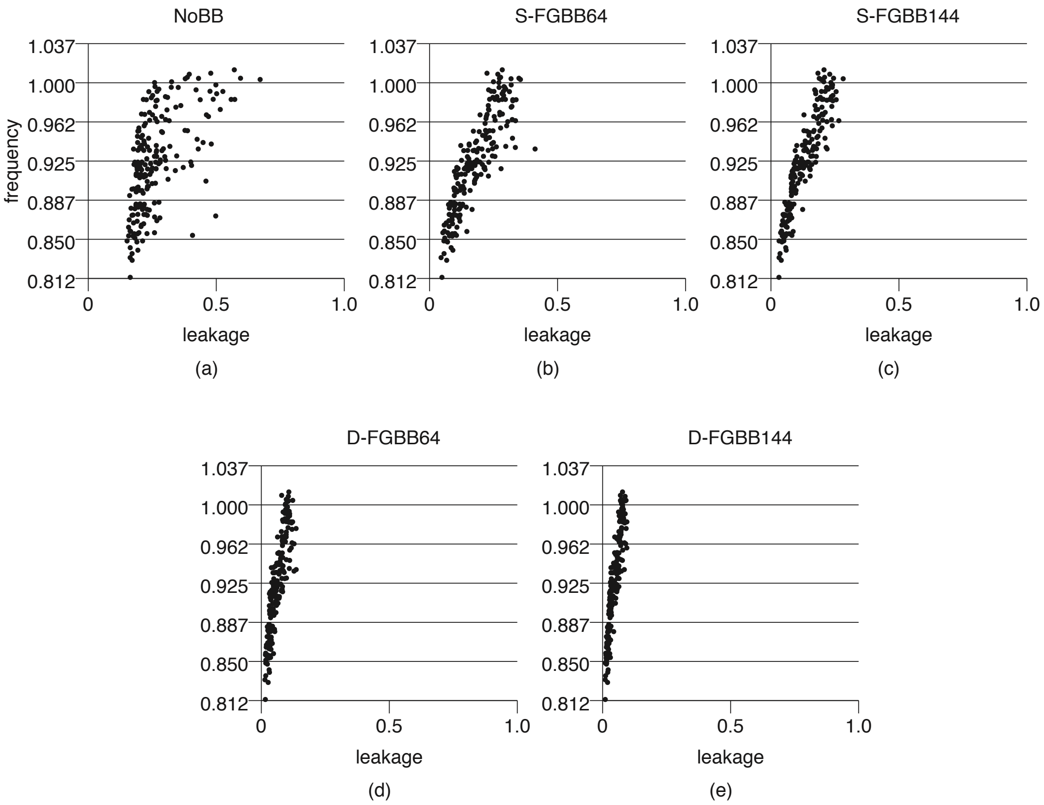


Figure 3.13: Frequency versus leakage power for a batch of 200 chips at usual *T* and load conditions.

Comparing Chart (a) to (b)-(c), we see that, if we apply S-FGBB, the chips move to the left, therefore saving leakage. Moreover, Charts (d)-(e) show that D-FGBB reduces the leakage of the chips even further. The higher the number of cells per chip is, the higher the leakage reduction is.

Figure 3.14 extends these experiments to all the BB environments. Figure 3.14(a) shows the average leakage power of the chips normalized to NoBB. The figure shows that both S-FGBB and D-FGBB save substantial leakage, especially as the number of cells per chip increases. However, D-FGBB is much more effective. D-FGBB reduces the leakage by 10–51% compared to S-FGBB, and by 12–69% compared to NoBB. Even with only 16 cells per chip, D-FGBB saves substantial leakage.

Figure 3.14(b) shows the total power consumption for the different FGBB schemes,

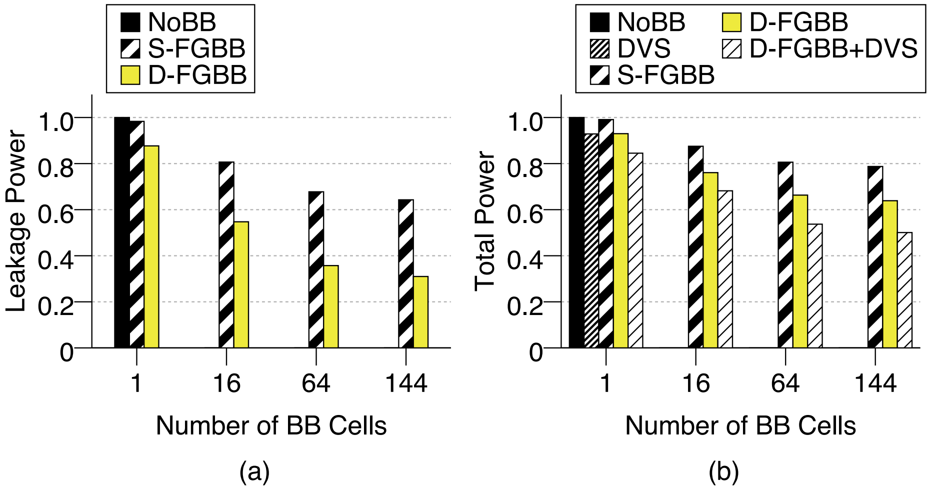


Figure 3.14: Leakage (a) and total power (b) of the chips for different FGBB schemes at constant frequency.

DVS, and D-FGBB+DVS. The savings induced by D-FGBB are still large. Specifically, D-FGBB reduces the total power consumption by 6–19% relative to S-FGBB. When combined with DVS, D-FGBB+DVS reduces total power consumption by 15–36% compared to S-FGBB. DVS alone is not very effective.

3.6.7 Dynamic Voltage and Frequency Scaling (DVFS)

Since many processors today use DVFS to save power, we would like to examine how the effectiveness of D-FGBB changes as V_{dd} decreases with DVFS. For that, we take each chip in the batch and, for a set of supply voltages V_{dd}^i ranging from 1V to 0.6V, determine the corresponding frequency F_i before BB. Then, we apply D-FGBB at F_i. Finally, we measure the leakage and total powers for each V_{dd}^i before and after applying D-FGBB. The results are shown in Figure 3.15, where all bars are normalized to NoBB with V_{dd}=1V.

Figure 3.15(a) shows that D-FGBB retains its relative effectiveness at reducing leakage as V_{dd} decreases from 1V to 0.6V — for all numbers of cells. Naturally, the absolute reduction decreases as V_{dd} decreases because there is less leakage to start with. Figure 3.15(b)

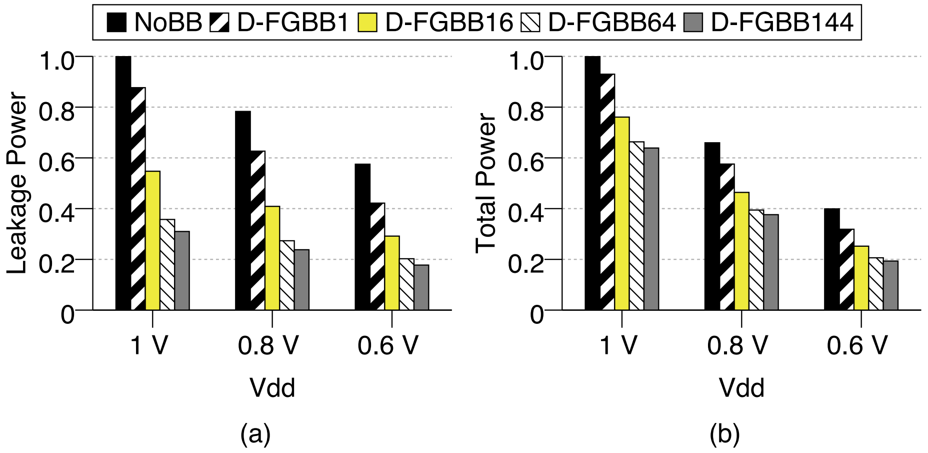


Figure 3.15: Leakage (a) and total power (b) at different voltage-frequency pairs, without and with D-FGBB.

shows that the total power savings are smaller but still very significant.

On the other hand, if we use S-FGBB, the BB levels are fixed at manufacturing time and cannot change with different voltages. When the same experiment is attempted with S-FGBB, we observe that the BB levels set at V_{dd}=1 are such that, as the voltage decreases, the processor cannot meet timing at the lower frequencies. Consequently, S-FGBB and DVFS cannot be easily combined.

3.6.8 Estimated Area Overhead of D-FGBB

To estimate the area overhead of D-FGBB, we use published data on BB support in real chips. Specifically, we use the area overhead reported in [58, 82] and scale it down to 45nm. We consider two implementations: one that uses critical path replicas and one that uses actual critical paths. Figure 3.16 shows the overhead as a fraction of the chip area. We see that the overhead with replicas varies between $<2\%$ and 4%, increasing with the number of BB cells. If actual critical paths are used rather than replicas, the overhead decreases to $\approx 3\%$ for 144 cells.

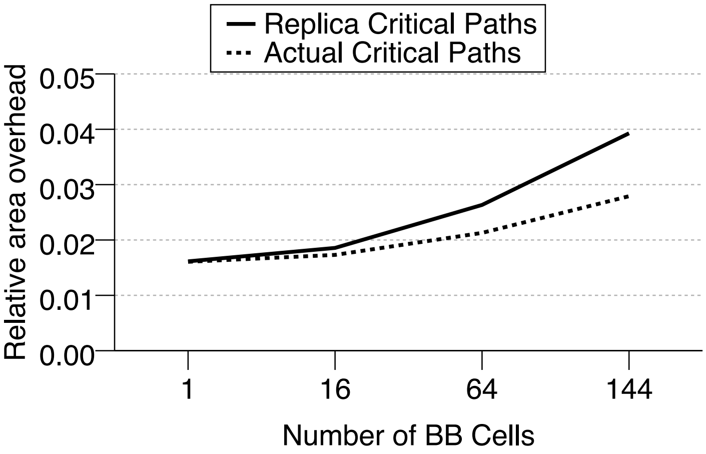


Figure 3.16: Area overhead of D-FGBB as a fraction of the chip area.

CHAPTER 4

Variation-Aware Application Scheduling and Power Management for Chip Multiprocessors

4.1 Introduction

In the context of Chip Multiprocessors (CMP), within-die process variation in current and near-future technologies causes individual cores in the chip to differ substantially in the amount of power that they consume and in the maximum frequency that they can support. This effect, which has been reported elsewhere [34] and will be confirmed in this work, suggests that it is no longer accurate to think of large CMPs as homogeneous systems.

In these environments, it is suboptimal to schedule applications ignoring variation effects. Instead, if applications are scheduled in a variation-aware manner, taking into account the different power and frequency characteristics of each individual core, substantial savings in power or large increases in throughput are attainable. Similarly, it is suboptimal to perform power management based on Dynamic Voltage and Frequency Scaling (DVFS) assuming that all cores have the same properties. Instead, if DVFS is applied in a per-core manner, fully aware of the heterogeneity of the cores, substantially more cost-effective working points can be obtained.

Interestingly, the technology for variation-aware application scheduling and power management is now available. Indeed, sophisticated on-chip power monitors and controllers such as those in Intel's Foxton technology [55] can be used to measure and manage the power heterogeneity of the cores. Moreover, the ability to support multiple on-chip frequency domains and change the frequency of each core independently as in AMD's Quad-Core Opteron [21] can be used to exploit the frequency heterogeneity of the cores.

In this work, we propose simple, variation-aware algorithms for application scheduling in a CMP to save power or improve throughput. Moreover, we complement these algorithms with variation-aware power-management DVFS algorithms to maximize throughput at a given power budget. One such power-management algorithm, called *LinOpt*, uses linear programming to find the voltage and frequency levels for each of the cores in the CMP. *LinOpt* runs on-line periodically, using profile information provided by the chip manufacturer and by on-chip power and IPC sensors. In a 20-core CMP, the combination of variation-aware application scheduling and *LinOpt* increases the average throughput by 12–17% and reduces the average ED^2 by 30–38% — all relative to using variation-aware scheduling together with a simple extension to Foxton's power management algorithm. Moreover, *LinOpt*'s throughput is within 2% of that of a simulated annealing algorithm, which has a computation time orders of magnitude higher.

4.2 Application Scheduling and Power Management under Process Variation

In an environment with process variation, each processor in a CMP typically consumes a different amount of power and can support a different maximum frequency. Given that the core-to-core differences can be substantial, it makes sense to schedule applications and perform power management as if the variation-affected CMP was a heterogeneous system.

	Uniform Frequency	Non-Uniform Frequency
No DVFS	Name: *UniFreq* – Minimize Power	Name: *NUniFreq* – Minimize Power – Maximize Performance
DVFS	Name: *UniFreq+DVFS* – Maximize Performance at a Power Budget	Name: *NUniFreq+DVFS* – Maximize Performance at a Power Budget

Table 4.1: CMP configurations for application scheduling and power management under variation.

In this environment, there are two high-level design issues (Table 4.1). The first is whether the different cores of the CMP have to cycle at the same frequency (Uniform Frequency) or not (Non-Uniform Frequency). The second is whether the frequency and voltage of the cores can be changed dynamically (DVFS) or not (No DVFS). For these configurations, we present a set of simple variation-aware algorithms for application scheduling and power management, aimed at minimizing power or maximizing performance. Note that the scheduling algorithms are intended to complement the other scheduling criteria used by the OS, such as priority, fairness, or starvation-avoidance.

4.2.1 UniFreq: Uniform Frequency & No DVFS

In this configuration all cores run at the frequency of the slowest one. Moreover, frequency does not change dynamically. We call this configuration *UniFreq* (Table 4.1). The only inter-core variation is in power consumption. Since all cores run at the same frequency, the goal of the scheduler is to minimize the power consumption at the given frequency.

The top part of Table 4.2 shows three possible algorithms that the scheduler can use in this configuration to minimize power. In *Random*, threads are mapped on cores randomly. This is our baseline. In *VarP*, the cores are ranked by their static power consumption from lowest to highest. Then, the *N* threads are randomly mapped on the top *N* cores (one thread per core). Finally, in *VarP&AppP*, in addition to ranking the cores as before, the threads are ranked by their dynamic power consumption from highest to lowest. Then, the highest-power threads are mapped on the lowest-power cores. The intuition is to “even out” the power consumption across cores, avoiding hot spots. Because of the exponential dependence of leakage on temperature, keeping the power consumption (and temperature) as uniform as possible saves power.

The configuration in Table 4.1 where the frequency is uniform and there is DVFS is called *UniFreq+DVFS*. It is a generalization of *UniFreq*, where the goal is now to maximize

Algorithms to Minimize Power	
Random	Map threads on cores randomly
VarP	Map threads randomly on cores with lowest power
VarP&AppP	Map threads with highest dynamic power on cores with lowest static power
Algorithms to Maximize Performance	
Random	Map threads on cores randomly
VarF	Map threads randomly on cores with highest frequency
VarF&AppIPC	Map threads with highest IPC on cores with highest frequency
Algorithms to Maximize Performance at a Power Budget (P_{target})	
*Random+Foxton**	Map threads on cores randomly and reduce (V_i,f_i) of cores round robin to meet P_{target}
*VarF&AppIPC+Foxton**	Map threads on cores with *VarF&AppIPC* and reduce (V_i,f_i) of cores round robin to meet P_{target}
VarF&AppIPC+LinOpt	Map threads on cores with *VarF&AppIPC* and use *LinOpt* to meet P_{target}
VarF&AppIPC+SAnn	Map threads on cores with *VarF&AppIPC* and use *SAnn* to meet P_{target}

Table 4.2: Algorithms for application scheduling and power management.

performance at a given power budget. We can use the same scheduling algorithms as in *UniFreq* — e.g., mapping the threads with the highest dynamic power on the cores with the lowest static power — and then reduce the frequency and voltage until the power budget is met. Since most aspects of this configuration are covered in the other configurations, we do not consider it further in this work.

4.2.2 NUniFreq: Non-Uniform Frequency & No DVFS

In the *NUniFreq* configuration (Table 4.1), different cores run at different frequencies — the highest that each supports. However, frequencies do not change dynamically. In this configuration, two simple scheduling goals are to minimize power or to maximize performance.

To minimize power, we use the *VarP* or *VarP&AppP* algorithms discussed in the previous section. To maximize performance, we use the algorithms in the middle of Table 4.2. In *VarF*, the cores are ranked by the maximum frequency they support, from highest to lowest. Then, the *N* threads are randomly mapped on the top *N* cores (one thread per core). In *VarF&AppIPC*, in addition to ranking the cores as before, the threads are ranked by their average IPC from highest to lowest. Then, the highest-IPC threads are mapped on the highest-frequency cores. The intuition is that low-IPC threads typically benefit less from high frequency — because they are often memory-bound.

4.2.3 NUniFreq+DVFS: Non-Uniform Frequency & DVFS

In this configuration, different cores run at different frequencies and DVFS can be applied independently to each core. We call this configuration *NUniFreq+DVFS* (Table 4.1). The challenge in such a system is to dynamically determine what voltage and frequency each core should have, depending on the characteristics of the applications and the constraints on the system. In large CMPs, core-level power management decisions will be less effective because they do not consider the constraints of the entire system. What we need instead is a global (chip-wide) approach to power management, with global goals and constraints. One possible goal is to maximize overall system throughput under a set of constraints. The main constraint can be keeping total power below a power target.

The lower part of Table 4.2 shows possible algorithms to maximize throughput at the target power. To simplify the problem, we construct each algorithm in two steps. First, we select one of the scheduling algorithms that maximizes performance (*VarF* or *VarF&AppIPC*), to map the threads to cores. Then, we use a power management algorithm to find the best (V_i,f_i) pair for each active core *i*, that maximizes overall throughput while keeping the total power no higher than P_{target}.

Because of the non-linear dependence of power on *V*, and the exponential size of the search space, finding the optimal solution to the second part of the problem is very expensive. Previous solutions that have looked at global optimization of DVFS on CMPs [35] have used an *exhaustive* search through the solution space. This is feasible only for very small systems and does not scale. For a system like the one we evaluate (a 20-core CMP with many per-core voltage-frequency pairs), exhaustive search is too expensive.

Our approach is to approximate the problem to a linear optimization problem, and then use linear programming [63] to solve it. We call our algorithm *LinOpt*. To evaluate the impact of this approximation on the accuracy of the solution we also use a more costly non-linear algorithm based on simulated annealing (*SAnn*) to solve the original, non-linear problem. In Section 4.4.5 we show that *SAnn* provides a solution that is very close to the optimal one, obtained through exhaustive search.

Overall, we examine the following algorithms (Table 4.2). In *Random+Foxton**, we map threads on cores randomly. Then, from among the active cores, we select one core at a time in a round-robin manner, and reduce that core's (V_i,f_i) one step. We stop when the chip-wide P_{target} constraint is satisfied and a per-core power constraint ($P_{coremax}$) is satisfied for all cores. This power management algorithm is the simplest one, and a small extension over the one implemented by the Foxton controller in the Itanium II [55] — where the two cores have the same (V,f) pair. We use *Random+Foxton** as our baseline algorithm.

In the other algorithms, we map threads on cores using *VarF&AppIPC* and then use different power management algorithms to set the (V_i,f_i) pair for each active core *i*. Specifically, we consider *Foxton**, *LinOpt*, and *SAnn*. To be effective, both *LinOpt* and *SAnn* have the same goal as the *VarF&AppIPC* scheduling algorithm, namely to maximize performance. In the following, we present the *LinOpt* and *SAnn* algorithms.

4.2.4 LinOpt: Power Management Using Linear Programming

Linear programming [63] is a mathematical technique for solving linear optimization problems of the following form: for N independent variables $x_1, ..., x_N$, maximize the objective function:

$$g = a_1x_1 + a_2x_2 + ... + a_Nx_N$$

subject to N primary constraints $x_1 \geq 0, x_2 \geq 0, ..., x_N \geq 0$ and to any number of additional constraints of the form:

$$b_1x_1 + b_2x_2 + ... + b_Nx_N + b \leq B$$

where $a_{1..N}$, $b_{1..N}$, b and B are problem-specific constants.

The problem we want to optimize is the following. Given a set of N cores $C_{1..N}$ that can each run at M different voltage levels $V_{1..M}$ (each with its corresponding frequency), find the best selection of voltage levels $(v_1, ..., v_N)$ for cores $C_{1..N}$ that maximizes the average throughput (*TP*) subject to the following constraints: (i) the total chip power is less than P_{target} and (ii) the power of each core is less than $P_{coremax}$.

We would like to express this problem as a linear optimization problem. To do this, we need to make sure that the objective function as well as all the constraint inequalities are linear functions. This requires some approximation.

We start with the objective function, which is the average throughput *TP*, measured in millions of instructions per second (MIPS). If we call tp_i the throughput of core i, we have:

$$TP = \frac{tp_1 + tp_2 + ... + tp_N}{N}$$

We express *TP* as a linear function of the variables we are trying to find, namely the set of optimal voltage levels $(v_1, ..., v_N)$ for the cores. By definition, $tp_i = f_i \times ipc_i$, where f_i is the frequency of core i and ipc_i is the IPC of the thread running on i. Now, both f and

ipc are largely linear functions of v — with parameters that depend on the core used and the application. Consequently, we can write $tp_i = a_i v_i$, where a_i is application and core dependent. We obtain, for a given assignment of applications to cores:

$$TP = \frac{a_1}{N}v_1 + \frac{a_2}{N}v_2 + ... + \frac{a_N}{N}v_N$$

where $v_{1..N}$ are the set of voltages we are trying to find. Next, we define the constraints of the optimization problem. Two trivial sets of constraints are the upper and lower bounds on the values of $v_{1..N}$:

$$v_1, v_2, ...v_N \leq V_{high} \text{ and } v_1, v_2, ...v_N \geq V_{low}$$

Next, we define the main constraint, which specifies that the total CMP power is less than P_{target}. For this, we need to express the total power of each core i as a linear function of supply voltage as $p_i = b_i v_i + c_i$, where b_i and c_i are both core- and thread-specific constants. In reality, the total power of a core is clearly not a linear function of supply voltage — dynamic power is quadratic in supply voltage (or cubic, if we add the corresponding change in frequency) and static power is more than linear. However, in practice, the solutions that we obtain with this linear approximation are good. They satisfy the power constraints with little slack and provide a performance very close to that obtained with more time consuming, non-linear formulations. This will be shown in Section 4.5.5.

Because of variation, we cannot generate the $p = f(v)$ function analytically. Instead, we experimentally measure the power of a thread-core pair at three voltage levels, namely V_{low}, V_{high} and V_{mid} (the average of the two). Then, as shown in Figure 4.1, we find the values of the constants b_i and c_i that minimize the differences $dErr$ for all three points.

The P_{target} constraint equation can then be written as follows, where all the $c_{1..N}$ constants are folded into c:

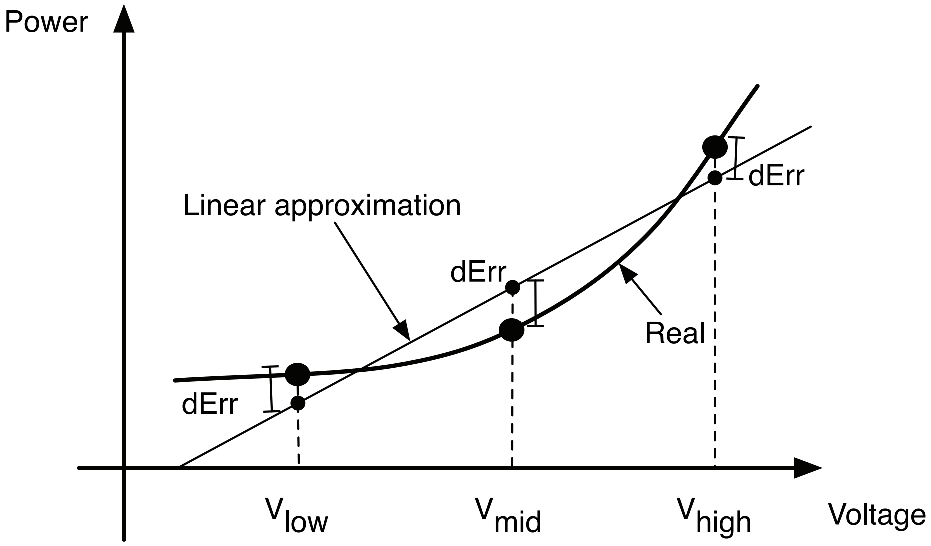


Figure 4.1: Linear approximation of the power dependence on voltage.

$$b_1v_1 + b_2v_2 + ... + b_Nv_N + c \leq P_{target}$$

Finally, the last set of constraints specifies that the power $p_{1..N}$ of each core should be less than $P_{coremax}$:

$$b_iv_i + c_i \leq P_{coremax}, \forall i \in 1..N$$

There are several techniques for solving linear programming problems. We choose the Simplex method [63] because it is relatively straightforward to implement and, in practice, it is often fast to compute.

The inputs to *LinOpt* are the constants $a_{1..N}$, $b_{1..N}$ and $c_{1..N}$, as well as the power constraints P_{target} and $P_{coremax}$. In Section 4.3, we show how we use profiling to compute these constants. The output of *LinOpt* is the best voltage for each core $(v_1, ..., v_N)$.

4.2.5 Other Global Optimization Solutions

We also examine solving the optimization problem of Section 4.2.4 using a non-linear algorithm. We choose simulated annealing (*SAnn*) — a well-known probabilistic algorithm for solving global optimization problems [46]. The goal of *SAnn* is the same as the one used with *LinOpt*: maximize throughput under power constraints. For a given mapping of threads to cores, the search space of the *SAnn* algorithm consists of all possible combinations of voltage levels (and their corresponding frequency levels) for each of the cores.

Unlike *LinOpt*, *SAnn* computes the power at each voltage level accurately, without the linear approximation. This should allow *SAnn* to generate a better solution. However, unlike linear programming, simulated annealing may not find the global optimum and, instead, produce a local optimum. In practice, the results of Section 4.5 show that *SAnn* produces better results than *LinOpt*. However, *SAnn* is orders of magnitude slower than *LinOpt*, which makes it impractical for on-line use.

4.3 System Implementation

Our target system is a CMP with many cores — 20 in our evaluation. The algorithms for application scheduling and power management of Section 4.2 are run by supervisor code. The power management algorithm can be run by either a core or a Power Management Unit (PMU) as in the Itanium II [55]. The application scheduling and power management algorithms may use profile information about core frequency and power, or application dynamic power and IPC. In what follows, we briefly outline the frequency, voltage and power control, and the profiling support.

4.3.1 Frequency, Voltage and Power Control

A CMP with per-core frequency control requires separate PLLs to generate the clock signal for each core. These PLLs are controlled independently. Moreover, there is a synchronization mechanism between the different frequency domains, such as FIFO buffers. All this support is already present in AMD's Quad-Core Opteron [21].

To support per-core voltage control, the CMP needs per-core power grids and voltage regulators that generate the different voltages. Currently, such regulators are on the board, but new technologies will soon make it possible to place them on the processor package or even on the die [45]. In this case, voltage transition speeds will be orders of magnitude faster. In this work, however, we conservatively assume that the voltage and frequency transition speeds are those of current systems such as Xscale [14].

To run a power management algorithm such as *LinOpt*, we need on-chip sensors that provide power consumption information as in Itanium II [55], on a per-core basis. If a PMU is used to run the algorithm, a simple on-chip design like the controller in Itanium II can be used. Such a design consumes less than 0.5W and takes up less than 0.5% of the die area.

Figure 4.2 shows a timeline of the execution of the algorithms. At every OS scheduling interval, the OS revisits its assignment of threads to cores using one of the scheduling algorithms — for instance, *VarF&AppIPC*. At more frequent intervals, e.g., every 10ms, the *LinOpt* algorithm runs and sets the cores to the best (V, f) pairs.

4.3.2 Profiling Support

For the application scheduling and power management algorithms to be effective, they need some profile information. Some of this information is provided by the chip manufacturer, while other is provided dynamically by sensors as applications execute. Table 4.3 summarizes the profile information needed for each application scheduling and power man-

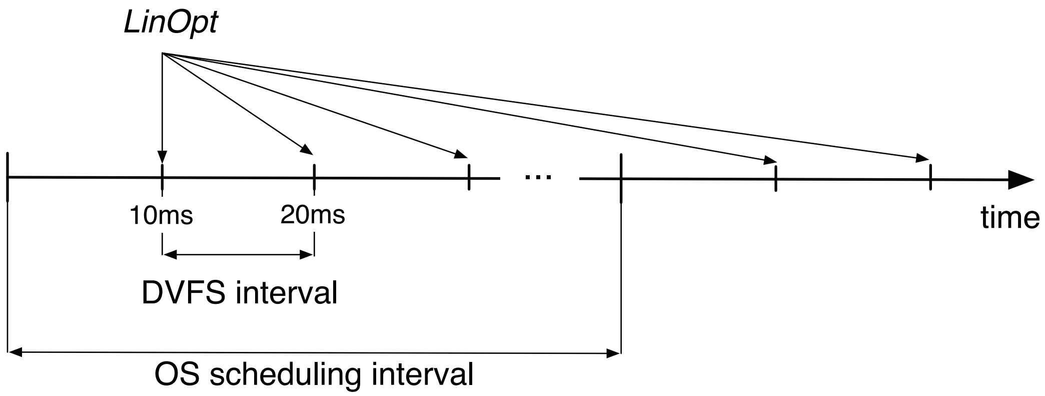


Figure 4.2: Execution timeline for application scheduling and *LinOpt* invocation.

agement algorithm.

For the *VarP* scheduling algorithm, we need, for each core, the static power consumption at the maximum voltage. This is provided by the manufacturer, who measures the power while keeping the chip under zero load. Note that this is only an estimate of the actual static power of the cores at run time because the static power is heavily dependent on the temperature. However, it is good enough because we are only interested in a *ranking* of cores based on their static power.

For the *VarP&AppP* algorithm, we need, for each core, the static power consumption at *each* voltage level (Table 4.3). All these values are also provided by the manufacturer. In addition, we need, for each thread, the dynamic power it consumes while running on one random core (Table 4.3). This information is obtained by reading the power sensors in the core for a given section of the thread's execution. The measured power is the total power, so we need to subtract the static power. Since the core may be running at a voltage different than the maximum value, we need to know the static power consumption at the current voltage — hence the need for the previous information.

Note that each thread is potentially profiled on a different core. To compare the resulting dynamic powers obtained, the power measured is scaled according to the frequency and voltage of the particular core used. We assume that the other factors that determine the

Algorithm		Profile Information Required
Sched.	*VarP*	For each core: static power consumption at the maximum voltage
	VarP&AppP	For each core: static power consumption at each voltage level For each thread: dynamic power consumption while running on one random core
	VarF	For each core: maximum frequency supported at the maximum voltage
	VarF&AppIPC	For each core: maximum frequency supported at the maximum voltage For each thread: IPC while running on any core
Power Manag.	*Foxton**	For each core: total power consumption
	LinOpt	For each core: table of (voltage, frequency) pairs For each selected thread-core pair: IPC of the thread while running on the core For each selected thread-core pair: total power at 3 (or 2) voltage levels

Table 4.3: Profile information needed for the application scheduling and power management algorithms.

dynamic power are largely constant across cores. Again, these measurements are good enough because we are only interested in a ranking of threads based on their dynamic power.

The *VarF* algorithm needs, for each core, the maximum frequency supported at the maximum voltage. This is provided by the manufacturer.

The *VarF&AppIPC* algorithm additionally needs, for each thread, the IPC it delivers while running on one random core (Table 4.3). The IPC is obtained by reading simple performance counters in the core for a given section of the thread's execution. Each thread is profiled on a potentially different core. As indicated in Section 4.2.4, we assume that the IPC changes negligibly with frequency and voltage changes — although a correction could be made based on the measured miss rate. We also assume that no other core property affects the IPC. Again, we are only interested in a ranking of threads based on their IPC.

We now consider the power management algorithms. *Foxton** needs information about total power consumption for each core. *LinOpt*, as described in Section 4.2.4, needs some additional information (Table 4.3). First, for each core, it needs a table of (voltage, frequency) pairs. This table is supplied by the manufacturer. Then, for each of the thread-core pairs selected by the scheduling algorithm, we need the IPC of the thread while running on the core. As indicated before, this is obtained dynamically with performance counters and is assumed largely independent of the frequency and voltage. With these two sets of values, we can generate the $a_{1..N}$ constants of Section 4.2.4.

Finally, for each of the thread-core pairs selected by the scheduling algorithm, we need the power consumed at three (or at the very least two) voltages. This information allows us to generate a curve like Figure 4.1 for each of the selected thread-core pairs, and then generate the $b_{1..N}$ and $c_{1..N}$ constants of Section 4.2.4. This information is obtained dynamically with power sensors in the cores.

Since the IPC and power of a thread-core pair changes with time, IPC and power profiling is on all the time, and the *LinOpt* algorithm is run periodically at short intervals. At longer intervals, we run the scheduling algorithm, which may change the assignment of threads to cores based on the new conditions. This is shown in Figure 4.2.

Note that, in all algorithms, the system continuously monitors the total power and the per-core powers. These values are compared to P_{target} and $P_{coremax}$, respectively.

4.4 Evaluation Methodology

We use the SESC cycle-accurate execution-driven simulator [65] to model a large CMP with 20 2-issue out-of-order cores on 32nm technology. Each core is like an Alpha 21264. Figure 4.3 shows the floorplan of the CMP and Table 4.4 summarizes the architecture configuration. In the following, we discuss the different parts of our infrastructure.

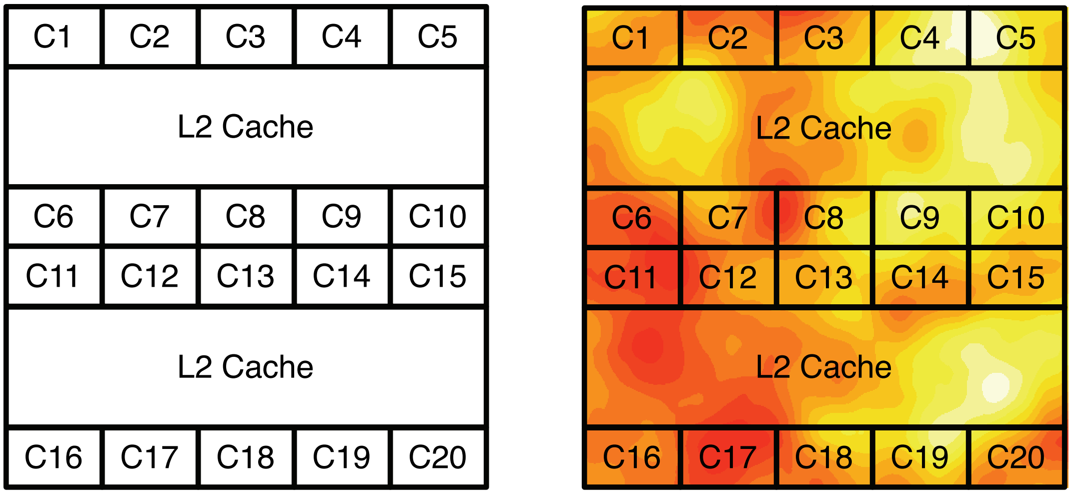


Figure 4.3: Floorplan of the 20-core CMP and superimposition of a V_{th} variation map.

4.4.1 Variation Model Parameters

To model WID variation, we use the VARIUS model [68, 79] to generate V_{th} and L_{eff} variation maps. We then superimpose these maps on our floorplan as shown in Figure 4.3. This allows us to model how variation affects core power and frequency.

Table 4.4 shows some of the process parameters used. There is little public-domain information on likely values for V_{th} σ/μ and ϕ. For σ/μ, the 1999 ITRS [36] gave a design target of 0.06 for year 2006 (although no solution existed); however, the projection has been discontinued since 1999. Toyoda [81] presents a measured $\sigma/\mu = 0.07$ for V_{th} in chips at $130nm$ technology. In this work, we consider a range of values for V_{th}'s σ/μ, namely 0.03–0.12, and use as default 0.12. Moreover, we assume that the random and systematic components have equal variances. For ϕ, we use Friedberg's *et al.* [25] measurement that the gate length had a correlation range close to 0.5 of the chip's width. Since the systematic component of V_{th}'s variation directly depends on the gate length's variation, we set ϕ = 0.5 for V_{th}.

Based on the 1999 ITRS [36], we set L_{eff}'s σ/μ to 0.5 of V_{th}'s σ/μ. Moreover, for L_{eff}, we also assume that that the random and systematic components have equal variances, and that ϕ = 0.5.

Overall: CMP with 20 out-of-order Alpha 21264-like procs.
Technology: 32nm, 4GHz (nominal)
Branch prediction: 4K-entry BTB, 7-cycle mispred. penalty
Core fetch/issue/commit width: 4/2/2
Register file size: 80 entry; Scheduler size: 20 fp, 40 int
Private data and instr. L1: 2-way 16K each; 2-cycle access
Shared L2: 8-way 8 MB; 8-12 cycle access
Cache line size: 64 bytes all
Memory access time: 400 cycles
Die size: $340mm^2$; V_{DD}: 0.6-1V (default is 1V)
Number of dies per experiment: 200
V_{th}: μ: 250mV at 60 °C
σ/μ: 0.03-0.12 (default is 0.12)
ϕ (fraction of chip's width): 0.5

Table 4.4: Summary of the architecture configuration.

Each individual experiment uses a batch of 200 chips that have a different V_{th} (and L_{eff}) map generated with the same μ, σ, and ϕ. To generate each map, we use the geoR statistical package [66] of R [64]. We use a resolution of 1M points per chip.

4.4.2 Power and Temperature Model

To estimate power, we scale the results given by popular tools using technology projections from ITRS [37]. Specifically, we use SESC, which is augmented with dynamic power models from Wattch [8] to estimate dynamic power at a reference technology and frequency. In addition, we use HotLeakage [86] to estimate leakage power at the same reference technology. Then, we obtain ITRS's scaling projections for the per-transistor dynamic power-delay product, and for the per-transistor static power. With these two factors, given that we keep the number of transistors constant as we scale, we can estimate the dynamic and leakage power for the scaled technology and frequency relative to the reference values.

We use HotSpot [69] to estimate on-chip temperatures. To do so, we use the iterative approach of Su *et al.* [76]: the temperature is estimated based on the current total power;

the leakage power is estimated based on the current temperature; and the leakage power is included in the total power. This is repeated until convergence.

4.4.3 Critical Path Model

To determine how the frequency of the processors is affected by variation, we need to model the structure and distribution of critical paths in the processors. For this, we use the models in [68], which include models for critical path distributions in pipeline stages with logic and in stages with memory structures. The distribution of critical path delays for logic stages is obtained using experimental data from Ernst *et al.* [23], who characterized a multiplier unit. For memory stages, [68] extends the model of Mukhopadhyay *et al.* [56] for the access time of a 6-transistor SRAM cell, to include the time to access the whole SRAM structure. With this model, we can estimate the frequency of the processors. We use CACTI 4.0 [77] to estimate path layouts and wire delays, and the alpha-power law [67] to compute gate delays.

4.4.4 Workloads

We evaluate our algorithms with a collection of applications from SPECint (*bzip2, crafty, gap, gzip, mcf, parser, twolf,* and *vortex*) and SPECfp (*applu, apsi, art, equake, mgrid,* and *swim*). We use applications from this pool to construct multi-programmed workloads that contain from 1 to 20 applications — where each application runs on a different core. This approach to construct workloads has been used elsewhere [35, 48]. Each experiment is repeated 20 times; each time with a different set of applications. We report the average outcome of the 20 trials. Each application runs with the reference input set for about 12 billion instructions.

4.4.5 Optimization Algorithms

The *LinOpt* algorithm uses the Simplex method [63] to solve the linear optimization problem. We use profile information as described in Section 4.3.2 to generate all the constants required. To approximate the power dependence on voltage as in Figure 4.1, we measure the power at 1, 0.8, and 0.6V. In our experiments of Section 4.5, we run *LinOpt* every 10ms.

For *SAnn*, we use the implementation of simulated annealing in the R statistical package [64]. The goal of *SAnn* is the same as *LinOpt*: maximize throughput under power constraints. In *SAnn*, the initial values of voltage and frequency for each core are determined using a simple greedy heuristic. The initial Annealing Temperature (AT) is determined based on the complexity of the problem: for a large number of threads, more randomness is needed in the initial search and, therefore, a higher value of the initial AT is used. As the number of threads decreases, we use a lower initial AT. At each AT, the next point in the solution space is generated from a Gaussian Markov kernel with scale proportional to the current AT. The algorithm automatically decreases the AT according to a logarithmic cooling schedule. The algorithm stops after 1 million function evaluations.

We use the results of *SAnn* as an upper bound for what *LinOpt* can achieve. We therefore want to make sure that *SAnn* produces a solution as close to the optimal one as possible. Consequently, we tune the constants in *SAnn* by comparing its results, for several configurations, to an *exhaustive search* through the solution space. Since the exhaustive search is very time consuming, we can only perform it for configurations of up to 4 threads. In all these cases, the *SAnn* throughput results are within 1% of those for the exhaustive search.

4.4.6 Metrics

In our evaluation, we use the following metrics: total power (which includes the static and dynamic powers of processors, L1 caches, and the L2 cache), average frequency of the active cores, throughput (measured in millions of instructions per second or MIPS), and the energy delay-square product (ED^2). We also give the weighted throughput [70], which uses the weighted IPC of the applications. The weighted IPC of an application is computed as the application's IPC normalized to the application's IPC at reference conditions. This metric gives equal weight to all the applications when measuring total system throughput.

4.5 Evaluation

We begin by examining the effect of process variation on core-to-core variation in power and frequency. Next, we evaluate the variation-aware algorithms for application scheduling and power management of Table 4.2 for the *UniFreq, NUniFreq*, and *NUniFreq+DVFS* configurations.

4.5.1 Variation Effects on Power and Frequency

To examine the potential of variation-aware algorithms, we measure the core-to-core variation. For a given die, we successively run all of our applications on a given core and compute the average power consumed by the core (which includes the L1 caches) per application. We repeat the experiment for all the cores. Then, we compute the ratio between the power consumed by the most power-consuming core to the power of the least power-consuming core. Figure 4.4(a) shows the resulting ratio for all the 200 dies in the form of a histogram. We see that, in most of the dies, there is 40-70% variation in total power. The average is around 53%.

We now examine core-to-core frequency variation. Since circuit delay increases with temperature, we measure the frequency of each core at the maximum temperature that any application reaches, which we measure to be around 95 ^{o}C. Then, for each die, we compute the ratio between the frequencies of the fastest and the slowest cores. Figure 4.4(b) shows the resulting ratio for all the 200 dies in the form of a histogram. We see that, in most of the dies, there is 20-50% variation in core frequency. The average is around 33%.

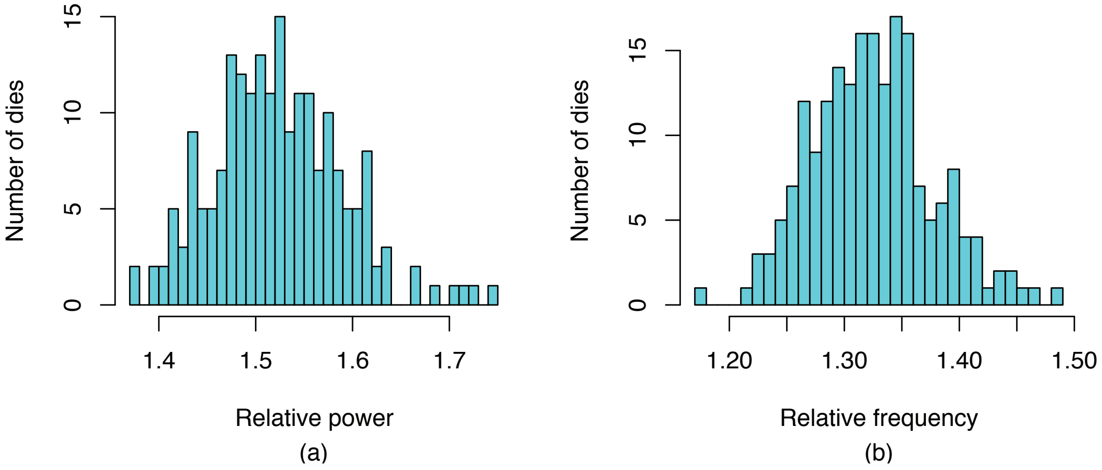


Figure 4.4: Histograms of the ratio between the powers consumed by the most and least power-consuming cores in the die (a) and between the frequencies of the fastest and slowest cores in the die (b).

Figure 4.5 shows how the average ratio between maximum and minimum core power (a) and frequency (b) changes with different values of V_{th} σ/μ. As expected, the core-to-core variation in both power and frequency increases with larger σ/μ. Even for a small σ/μ=0.06, the variation is very significant. Consequently, variation-aware algorithms for application scheduling and power management have good potential.

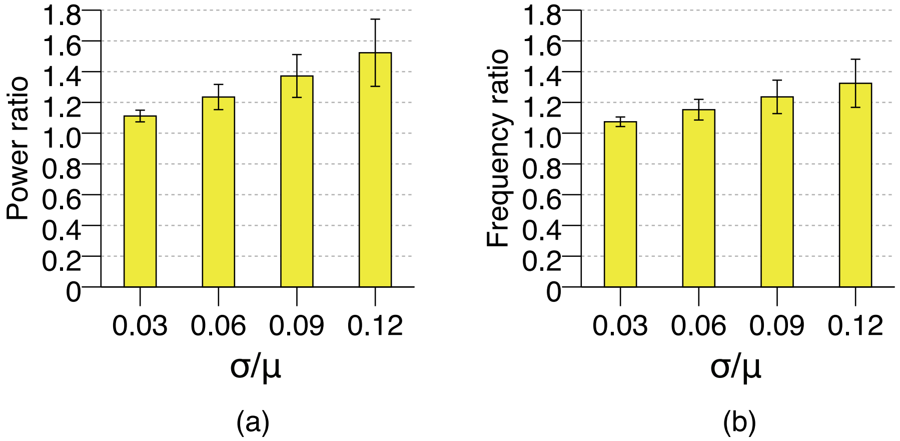


Figure 4.5: Average ratio between the maximum and minimum core power (a) and core frequency (b) for different values of V_{th} σ/μ, for 200 dies.

Due to variation, two different cores may achieve the same frequency at different voltage levels and with very different power costs. Moreover, the relative power efficiency of the two cores can change with the frequency. All this variation makes the job of a global power management scheme very challenging. As an illustration, we consider one sample die and identify the highest-frequency core (which we call *MaxF*) and the lowest-frequency one (which we call *MinF*). We run the *bzip2* application and, as we change the voltage levels, we measure the core power.

The result is shown in Figure 4.6. The figure shows the core power as a function of the frequency. Each core has a curve, where the dots represent voltage levels changing from 1V (top right) to 0.6V (bottom left). Power and frequency axes are normalized to the values for *MaxF* at 1V. The figure shows that, say, a 0.8 frequency can be obtained by *MaxF* at 0.7V or by *MinF* at 1V — but *MaxF* consumes less power. The figure also shows that, for frequencies below 0.74, *MinF* is more power efficient, while above that, *MaxF* is more efficient.

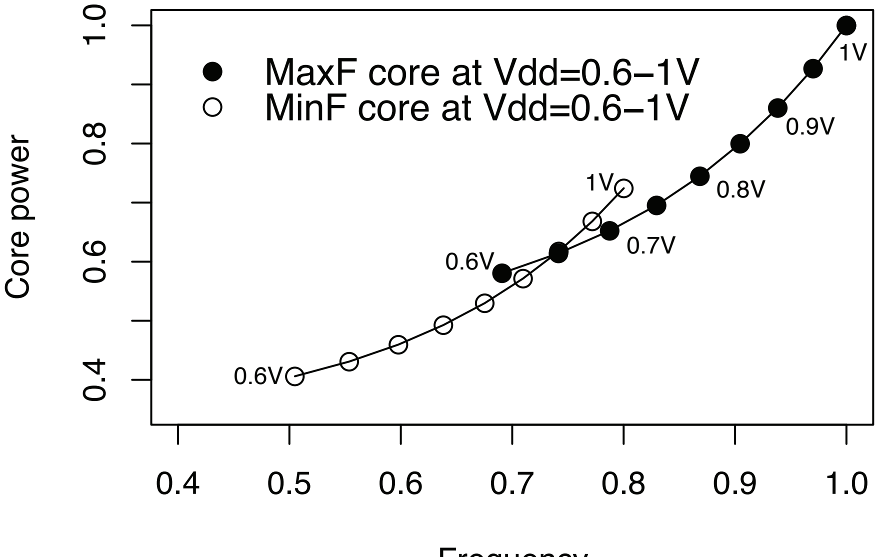


Figure 4.6: Core power as a function of frequency for the highest- and lowest-frequency cores in a sample die.

4.5.2 Application Power and IPC

Application scheduling and power management algorithms also leverage the fact that applications have a varied behavior. For example, Table 4.5 shows, for each of our applications, the average dynamic power of the core (which includes the L1 cache) at 4GHz and 1V, and the average IPC. From the table, we see that there is significant variation in both dynamic power (up to 2.9×) and IPC (up to 12×) across applications.

	applu	apsi	art	bzip2	crafty	equake	gap	gzip	mcf	mgrid	parser	swim	twolf	vortex
Dynamic power (W)	4.3	1.6	2.4	3.7	3.9	2.1	3.5	2.7	1.5	2.2	2.8	2.2	2.3	4.4
IPC	1.1	0.1	0.2	1.1	1.1	0.3	1.0	0.7	0.1	0.4	0.7	0.3	0.4	1.2

Table 4.5: Average dynamic power of the core at 4GHz and 1V, and IPC for the applications used.

4.5.3 UniFreq: Uniform Frequency & No DVFS

The first configuration we evaluate is one with all the cores running at the same frequency and no DVFS. Figure 4.7(a) shows the total power consumed in the *Random*, *VarP* and *VarP&AppP* scheduling algorithms of Table 4.2, as we vary the number of threads in the workload. For a given number of threads, the bars are normalized to *Random*. Note that the cores that are not used are assumed to be powered off.

Focusing on *VarP*, we see that for a lightly-loaded system, the savings in power are substantial — around 10% for 4 threads in the system. As the system load increases and more threads have to be scheduled, the power savings decrease. This is because more of the high-power cores need to be included in the scheduling pool. For full utilization (20 threads), *VarP* shows no power improvement.

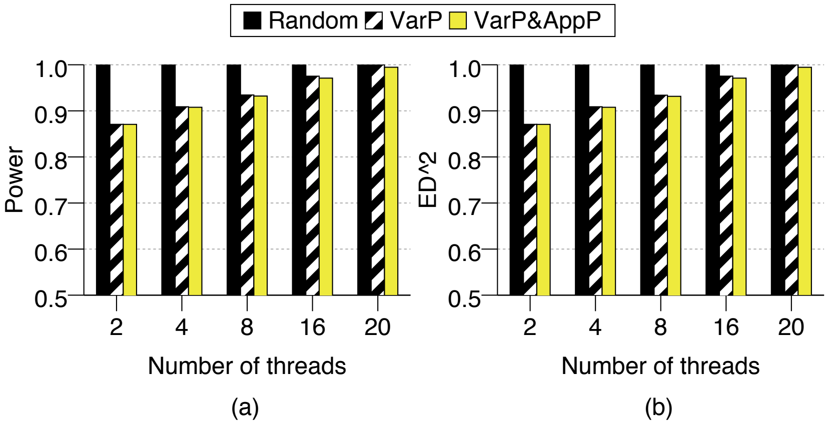


Figure 4.7: Total power consumption (a) and ED^2 (b) relative to *Random* in *UniFreq*.

VarP&AppP consumes the same power as *VarP*. The power averaging effect sought with *VarP&AppP* is not significant enough to reduce the temperature noticeably and, therefore, reduce the leakage power.

Figure 4.7(b) shows the ED^2 of the system for the different scheduling algorithms. Since *VarP* and *VarP&AppP* reduce the power at no cost in frequency, the ED^2 reduction is the same as the power reduction in Figure 4.7(a).

4.5.4 NUniFreq: Non-Uniform Frequency & No DVFS

NUniFreq allows each core to run at its maximum frequency. It can be shown that, under full occupancy (i.e., 20 threads), this increases the average core frequency by about 15% over *UniFreq* and increases the average power consumption by 10%. This in turn causes an average reduction in ED^2 of almost 20% for our applications. In what follows, we examine how variation-aware scheduling can further improve on these gains.

We start by evaluating algorithms to minimize power, namely *VarP* and *VarP&AppP*. Figures 4.8(a)–(b) are similar to 4.7(a)–(b) for *NUniFreq*. Figure 4.8(a) shows that the savings due to *VarP* and *VarP&AppP* are 14% for 4 threads. They decrease for more threads.

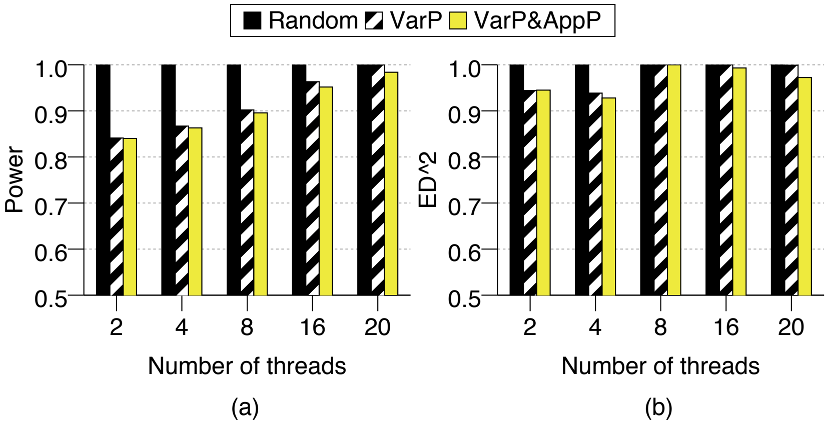


Figure 4.8: Total power consumption (a) and ED^2 (b) relative to *Random* in *NUniFreq*.

Figure 4.8(b) shows that these algorithms reduce ED^2 less than they did in 4.7(b). This is because in *NUniFreq*, different cores have different frequencies and *VarP* and *VarP&AppP*, by selecting the least-consuming cores, they may also end up selecting the lower-frequency ones, thus hurting ED^2. Note that the goal of this optimization is to reduce power, not ED^2.

We now consider algorithms to maximize performance, namely *VarF* and *VarF&AppIPC* (Table 4.2). Figure 4.9(a) shows the average frequency at which the threads run, normalized to that of *Random*. Recall that *VarF* and *VarF&AppIPC* select the same set of cores; therefore, their bars are the same. The figure shows that *VarF* increases the average frequency by 10% over *Random* for 4 threads. The frequency improvements decrease as the number of threads increases.

The benefits of *VarF&AppIPC* are apparent in Figure 4.9(b), which shows the average throughput in millions of instructions per second (MIPS) relative to *Random*. By scheduling high-IPC threads on high-frequency cores, *VarF&AppIPC* consistently delivers a higher throughput. Specifically, the throughput is 5–10% higher than *Random*. *VarF*, on the other hand, only delivers improvements for lightly-loaded systems — for the 20-thread configuration, it effectively works as *Random*.

Finally, Figure 4.10 shows ED^2 for the same algorithms. For lightly-loaded sys-

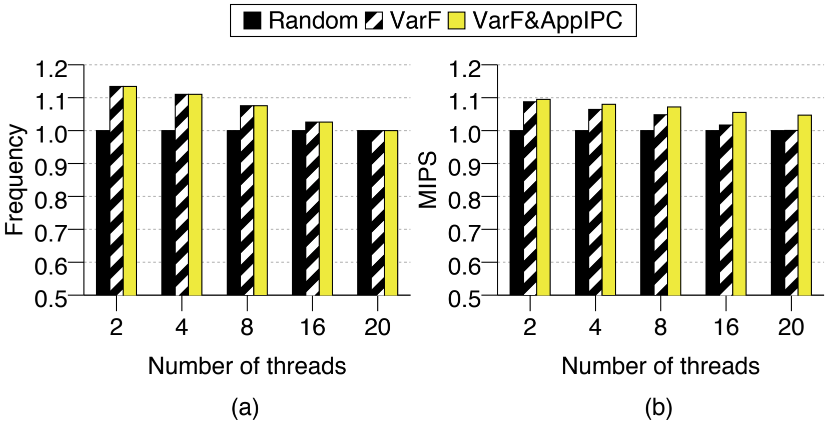


Figure 4.9: Average frequency (a) and average throughput (b) relative to *Random* in *NUniFreq*.

tems (4 threads or less), the higher throughput of *VarF* and *VarF&AppIPC* come at the cost of a higher ED^2. This is because the high-frequency cores selected dissipate more power, and the increase in throughput does not compensate. Again, the goal of the optimization was to improve performance, not ED^2. However, under higher loads (8 to 20 threads), *VarF&AppIPC* has a substantially lower ED^2 than *Random* or *VarF*. Specifically, *VarF&AppIPC*'s ED^2 is 10–13% lower than *Random*'s. The reason is that *VarF&AppIPC* increases the throughput substantially.

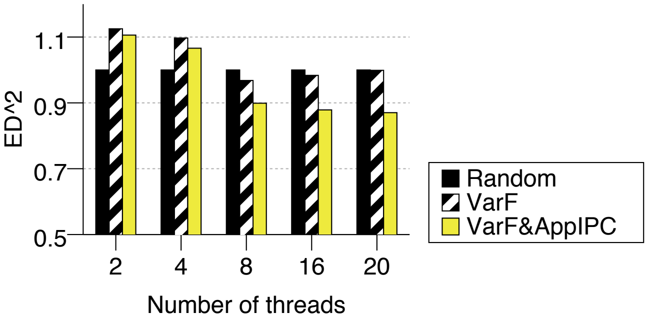


Figure 4.10: ED^2 relative to *Random* in *NUniFreq*.

4.5.5 NUniFreq+DVFS: Non-Uniform Frequency & DVFS

For *NUniFreq+DVFS*, we evaluate the algorithms in Table 4.2 that maximize performance at a given power budget, namely *Random+Foxton**, *VarF&AppIPC+Foxton**, *VarF&AppIPC+LinOpt*, and *VarF&AppIPC+SAnn*. We evaluate them under three Power Environments: *Low Power*, *Cost-Performance*, and *High Performance*. In these environments, the P_{target} when 20 threads are active is set to 50W, 75W, and 100W, respectively. When there are fewer threads, P_{target} is scaled down proportionally.

Figure 4.11(a) shows the average throughput of all the algorithms normalized to *Random+Foxton**, in the *Cost-Performance* Power Environment. We show results for different loads on the system, ranging from 4 to 20 threads. We see that *VarF&AppIPC+Foxton** only improves the average throughput by 4–6% for different numbers of threads. However, *VarF&AppIPC+LinOpt* is much more effective at boosting throughput. Specifically, it improves the average throughput by 12–17%. Moreover, its throughput is within 2% to that of *VarF&AppIPC+SAnn*. This is despite the fact that *VarF&AppIPC+SAnn* is orders of magnitude more costly in computation time. Also *VarF&AppIPC+SAnn*'s throughput is only 1% lower than the optimal throughput obtained through exhaustive search.

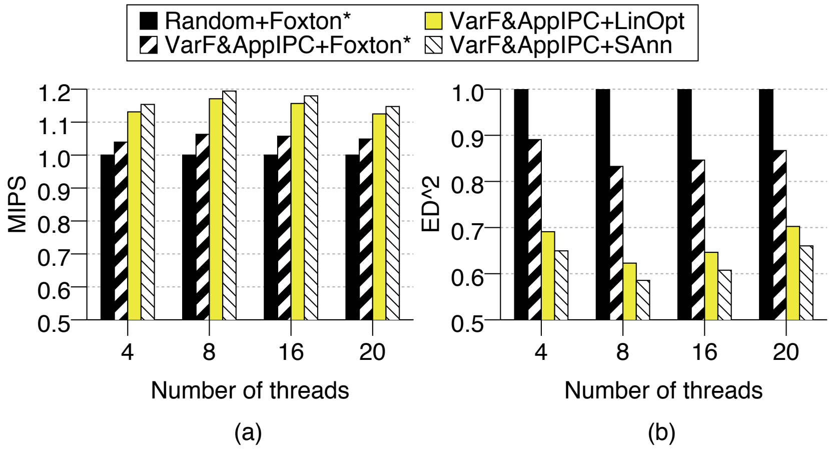


Figure 4.11: Average throughput (a) and ED^2 (b) for different algorithms relative to *Random+Foxton** in the *Cost-Performance* Power Environment.

Figure 4.11(b) shows the ED^2 for the same experiment. *VarF&AppIPC+LinOpt* reduces ED^2 by 30–38%. This is a very remarkable reduction, and is very close to that of *VarF&AppIPC+SAnn*.

We now compare the three Power Environments. Figure 4.12 shows the average throughput of the algorithms normalized to *Random+Foxton** in the three Power Environments. All experiments are for 20-thread runs. We can see that the relative throughput gains of *VarF&AppIPC+LinOpt* are highest when the power target is low. The same is true for the other algorithms. For *VarF&AppIPC+LinOpt*, the average throughput gains in the *Low Power*, *Cost-Performance*, and *High Performance* environments are 16%, 12% and 11%, respectively.

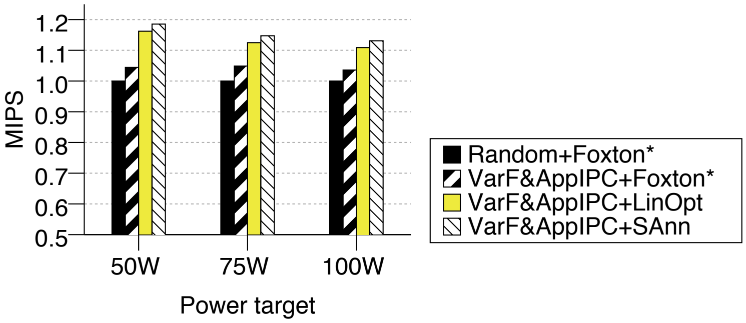


Figure 4.12: Average throughput for different algorithms relative to *Random+Foxton** in the three Power Environments. All experiments are for 20-thread runs.

Finally, we examine the impact of the algorithms on weighted throughput. This metric uses normalized IPC for each application and, therefore, is fairer to applications with low intrinsic IPC. Our algorithms improve throughput by adapting to IPC changes within each application — speeding up high-IPC sections and slowing down (and therefore saving power in) low-IPC sections.

Figure 4.13 shows the same experiments as Figure 4.11 but with weighted throughput as the optimization goal. We can see that the two figures are very similar, except for slightly smaller throughput improvements and ED^2 reductions in Figure 4.13. For example, *VarF&AppIPC+LinOpt* improves the weighted throughput by 9–14% and reduces ED^2 by 24–33% — rather than by 12–17% and 30–38%, respectively, in Figure 4.11.

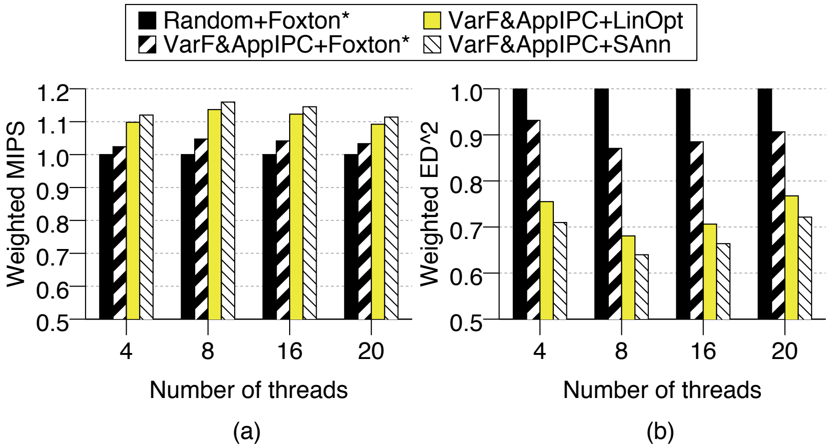


Figure 4.13: Average weighted throughput (a) and weighted ED^2 (b) for different algorithms relative to *Random+Foxton** in the *Cost-Performance* Power Environment.

4.5.6 LinOpt Granularity

How often we run *LinOpt* impacts our ability to keep the system power close to P_{target}. Specifically, if we use long intervals between *LinOpt* runs, the power consumed is likely to deviate more from P_{target} than if we use short intervals. Figure 4.14 considers different interval durations and measures the deviation between power consumed and P_{target}. Deviation is measured as follows. At every ms, the average power consumed in the past 1ms is compared to P_{target} and the absolute difference is recorded. Then, all the values recorded in the interval between two *LinOpt* runs are averaged out and plotted in Figure 4.14. The figure includes lines for 20- and 4-thread runs.

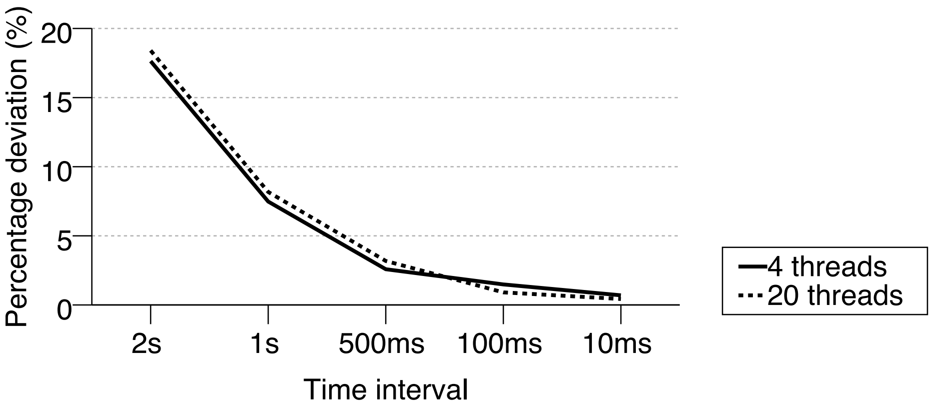


Figure 4.14: Average deviation of power consumption from P_{target} for different intervals between *LinOpt* runs.

Figure 4.14 shows that, as we decrease the interval between *LinOpt* runs, the power deviation decreases. For the 10ms-intervals that we use in our experiments, the deviation is less than 1%.

4.5.7 LinOpt Execution Time

The Simplex method used to solve *LinOpt* is generally very fast. The algorithm involves a variable number of steps, where the computation time of each step is affected by the size of the problem (the number of threads that are scheduled) and the number of constraints. Figure 4.15 shows the execution time of the algorithm on a 4GHz processor like the one considered in this work. The figure shows data for different numbers of running threads and the different Power Environments.

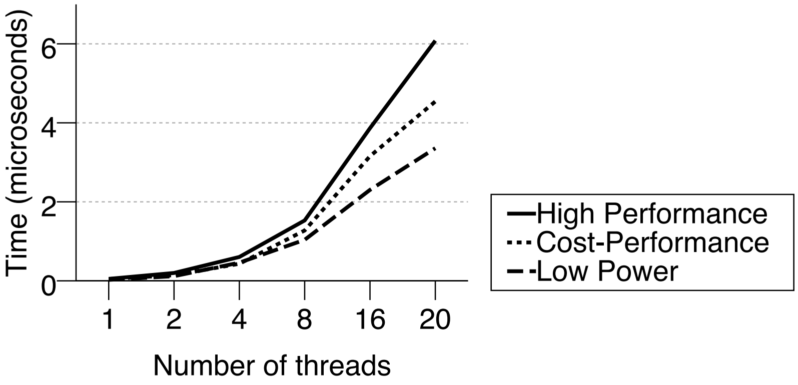


Figure 4.15: Execution time of the *LinOpt* algorithm for different numbers of threads in the three Power Environments.

The figure shows that the execution time increases with the number of threads. Moreover, it also increases as we go to less power-constrained environments such as *High Performance*. This is because a less strict environment increases the search space, making it harder to find a solution. Overall, the longest running time is 6μs. Since we run *LinOpt* every 10ms, the overhead is negligible.

CHAPTER 5

Related Work

The work presented in this thesis is related to an extensive body of research. In this Chapter we briefly describe some of this work and how it relates to this thesis.

5.1 Variation Measurement and Modeling

While the problem of process variation has long been known to the VLSI community, Borkar *et al.* [6] are one of the first to provide a microarchitectural perspective. Other key contributors are Bowman *et al.* [7], who provided a model to estimate chip frequency in the presence of WID process variation.

Substantial effort has been devoted to modeling parameter variation [73]. Most of the models are analytical and some have been validated with data obtained through measurements of test chips. For instance, Stine *et al.* [75], Orshansky *et al.* [60], and Friedberg *et al.* [25] performed measurements of test chips to characterize gate length variation. They observed that a significant portion of the WID variation is systematic.

An important issue has been how to model the spatial correlation of systematic variation. While we use a multivariate normal distribution with a spherical spatial correlation structure, other approaches include quad-tree modeling [53] and regression-based approaches [34]. In the former, it is difficult to control aspects of the correlation structure and the distribution parameters; in the latter, the model is deterministic — it models the distribution of the systematic component of just one die. It cannot be used to study a set of dies to find average statistics.

Cao and Clark [9] proposed a model that attributes V_{th} variation to gate length variation and studied the impact of spatial correlation on the delay of one critical path. Datta *et al.* [17] developed a statistical approach for pipeline delay analysis to show the importance of logic depth in variability analysis.

Chang and Sapatnekar [10] and Srivastava *et al.* [72] considered the impact of spatially correlated variation on leakage and/or performance. They generated distributions of coarse grid points by assuming a linear [10] or reciprocal [72] correlation function. They did not discuss trade-offs between the various correlation lengths and leakage/frequency.

5.2 Body Bias and Adaptive Supply Voltage

There is abundant work on body bias. Section 3.2 has outlined some of the main issues. In addition, Kumar *et al.* [49] pointed out the importance of BB adaptation to T changes. However, they rely on a static method, based on a mathematical model, to find the optimal BB voltages at manufacturing time, for all possible values of V_{th} and T that a circuit can have. In the presence of variation and given the scale of today's processors, this is a daunting task. Finally, Martin *et al.* [54] and Chen and Naffziger [13] examined the combination of BB and DVS.

Adaptive Supply Voltage (ASV) has also been proposed as a solution for process variation. ASV dynamically changes the supply voltage (V_{dd}) as a mechanism for trading off frequency for power. Previous work [13, 83] has shown ASV to be about as useful as S-FGBB at mitigating the effects of process variation. The hardware overhead is also comparable. Fine-grain ASV requires a voltage regulator for each cell, just like FGBB. However, a higher V_{dd} increases the rate of electromigration which affects the lifetime reliability of the circuit. At the same time, lower V_{dd} results in SRAM stability problems.

Other researchers have proposed microarchitectural techniques to mitigate or tolerate parameter variation. They target register file and execute units [53], data caches [61], pipeline balancing [80].

5.3 Handling Core-To-Core Process Variation

Humenay *et al.* [34] examine process variation in a CMP and point out the core-to-core variation in frequency. They estimate the maximum difference in core frequencies to be approximately 20%. They suggest Adaptive Body Bias (ABB) and Adaptive Supply Voltage (ASV) to reduce some of this variation — at the cost of increasing power variation. Donald and Martonosi [20] also examine process variation in a CMP and focus on the core-to-core variation in power. They suggest turning off cores that consume power in excess of a certain computed value, with the goal of maximizing the chip-wide performance/power ratio. In our work, we combine application scheduling and global DVFS power management.

5.4 Scheduling for Heterogeneous Architectures

Balakrishnan *et al.* [4] study how heterogeneous CMPs impact parallel workloads. They suggest fine-granularity threading as a solution for alleviating the performance instability caused by heterogeneous CMPs. Kumar *et al.* [48] propose a CMP with different-complexity cores and the same ISA. The goal is to reduce power consumption by using the simpler, more power-efficient cores to run memory-bound applications. They schedule applications to cores based on application ILP, and adapt to phase changes within an application. Our work considers design-identical cores that are affected by variation. Since we do not know beforehand the power and frequency of each core, this information is obtained post-manufacturing and is unique to each CMP.

Kadayif *et al.* [39] examine the benefits of exploiting the heterogeneity of workloads in the context of multicore embedded systems. Specifically, they use the compiler to assign different voltages and frequencies to different processors depending on the characteristics of the workload. Also in embedded systems, linear programming has been used to solve the problem of scheduling tasks on CMPs [71]. The goal is to minimize power while meeting strict timing constraints. Our optimization approach is different, since we do not have timing constrains. Moreover, due to variation, our cores are heterogeneous.

Other researchers like Heo *et al.* [28] and Stavrou and Trancoso [74] minimize power density or temperature hot spots by judiciously scheduling jobs or migrating them from core to core.

5.5 Power Management in Chip Multiprocessors

Given the importance of power, power management control is an area of high interest. Perhaps the most sophisticated design is Intel's Foxton technology [55], which has been implemented in the Itanium II processor. Foxton is a control system that maximizes performance while staying within target power and temperature. It consists of power and temperature sensors, and a small on-chip hardware controller. If the power consumption is less than the target, the controller increases the core voltage — and the frequency follows. The opposite occurs if the power is over the target. Both cores in the Itanium II have the same voltage and frequency.

Academic research on power management for CMPs is still in its infancy. Some examples include Li and Martinez [51] who optimize a parallel workload running on a CMP by dynamically changing the number of active processors and the voltage and frequency levels at which the CMP runs. They apply DVFS chip-wide rather than independently per core, which reduces the flexibility and impact of the optimization.

Isci *et al.* [35] consider a 4-core CMP with core-level DVFS (i.e., the voltage and frequency levels are changed independently in each core). They examine different DVFS policies for high performance and for power efficiency. Their solutions are primarily based on the exhaustive search of the solution space. Because of this, the solutions are not scalable to large systems. In this work, we consider a large CMP with process variation. As a result, both application scheduling and power management have to be variation-aware. Moreover, given the large design space, we need to use an intelligent way to prune the design space.

5.6 DVFS Granularity and Implementation

State-of-the-art processor chips often have multiple voltages — for example, giving memory arrays a slightly higher voltage than the core for reliability reasons. Herbert and Marculescu [29] examine the tradeoffs of using different DVFS granularities (i.e., the number of cores in the same voltage-frequency domain). They find that having a small number of cores per domain produces the most complexity-effective design.

The first general-purpose CMP to support a form of core-level DVFS is the AMD Quad-Core Opteron [21]. In this chip, the frequency of each core can be set independently, although all cores have the same voltage. Currently, multiple on-chip voltages are provided by off-chip voltage regulators, which are bulky and costly. Recent work by Kim *et al.* [45] describes designs of on-chip regulators. They are able to perform voltage changes in nanoseconds rather than in microseconds, and consume little energy. Designs similar to these will enable wide use of core-level DVFS.

5.7 Other Dynamic Power Management Algorithms

There has been significant research on reducing power and energy consumption using dynamic adaptation. Generally the adaptation is local, within a core and the tradeoff is between the amount of hardware resources that gets allocated to a task and the performance impact. In other cases the tradeoff is between DVFS and performance, or the two approaches are combined. Most previous work has used heuristic techniques to save power with the minimum impact on performance [1,3,22,24]. These techniques generally require a significant amount of tuning and it is hard to provide any optimality guarantees.

Other previous work has taken more formal approaches. Huang et al. [30] proposed an algorithm for architectural-level adaptation, assigning different processor configurations to different subroutines. The search space of possible configurations is relatively small, allowing them to always perform an exhaustive search and come up with the optimal solution in that space. Hughes and Adve [32,33] proposed an adaptive control algorithm for processors running multimedia applications. They define the adaptation as a constrained optimization problem that maximizes the energy saved per unit of work subject to timing constraints. They solve the problem using the method of Lagrange multipliers.

CHAPTER 6

Conclusions and Future Work

Parameter variation is a major challenge for processor designers. To address this challenge, we will likely need a combination of solutions at different layers, such as lithography, layout, circuits, and microarchitecture.

The work presented in this thesis makes several contributions in this space. First, we developed a parametrized model of process variation within the chip. The model was used throughout the thesis to estimate the effects of variation on current and future chip multiprocessors and to test the effectiveness of our proposed solutions for mitigating these effects.

In this work I proposed two solutions for dealing with variation. The first solution, D-FGBB attacks the problem at a lower level. Our results showed that D-FGBB is very versatile and effective. I outlined three uses of D-FGBB: (i) reducing the leakage power at constant frequency in normal processor operation, (ii) increasing the processor frequency in a high-performance mode, and (iii) reducing the leakage power at constant frequency in a low power mode.

In its first use, D-FGBB reduces the leakage power of the chip by an average of 28–42% compared to S-FGBB. The higher savings correspond to the cases with more BB cells per chip. If, in addition, we combine D-FGBB with DVS, we save both leakage and dynamic power. In the high-performance mode, D-FGBB increases the processor frequency by an average of 7–9% compared to S-FGBB and by 7–16% compared to no BB. Finally, in the low-power mode, D-FGBB reduces the leakage power of the chip by an average of 10–51% compared to S-FGBB and by 12–69% compared to no BB.

I also showed that D-FGBB can be synergistically combined with DVFS. While DVFS mostly controls dynamic power, D-FGBB controls leakage power. Overall, like DVFS, D-FGBB is a versatile control hook that can be managed in hardware or in software, and that can be used at different time and area granularities.

In the second part of this thesis I examined the effects of variation on large chip multiprocessors. What I found was that as a result of within-die process variation, individual cores in a CMP differ substantially in both static power consumed and maximum frequency supported. I proposed leveraging this core-to-core variation with variation-aware algorithms for application scheduling to save power or improve throughput, and variation-aware power-management algorithms to maximize throughput at a given power budget.

One such power-management algorithm, called *LinOpt*, uses linear programming to find the voltage and frequency levels for each of the cores in the CMP. *LinOpt* runs on-line periodically, using profile information provided by the chip manufacturer and by on-chip power and IPC sensors. In a 20-core CMP, the combination of variation-aware application scheduling and *LinOpt* increased the average throughput by 12–17$\%$ and reduced the average ED^2 by 30–38$\%$ — all relative to using variation-aware scheduling together with a simple extension to Foxton's power management algorithm. Moreover, *LinOpt*'s throughput was within 2$\%$ of that of a simulated annealing algorithm, which had a computation time orders of magnitude higher.

There are several possible avenues for future extensions to this work. One is to enhance the scheduling and power management algorithms with the additional goal of keeping the temperature of the CMP as uniform as possible. This can be achieved through aggressive migration of applications from active to inactive cores as in [28], and through temperature-aware mapping of applications to cores and assignment of (V,f) pairs. The result is likely to be fewer hot spots and lower power consumption, but it comes at the cost of increased complexity of the algorithms. Other possible extensions include understanding how our variation-aware algorithms affect CMP wearout, or analyzing the impact of the algorithms on parallel applications.

Dynamic fine-grain body biasing proved to be a very useful knob for trading off leakage power for frequency. An interesting extension to this work is exposing this knob to software. This will give the system another mechanism for managing power in addition to DVFS. Another interesting research topic would be to examine global power management policies that can co-optimize core-level dynamic body biasing and DVFS to achieve higher performance with lower power consumption.

REFERENCES

[1] D. H. Albonesi. Selective cache ways: on-demand cache resource allocation. In *International Symposium on Microarchitecture*, 1999.

[2] N. Azizi and F. Najm. Compensation for within-die variations in dynamic logic by using body-bias. In *International IEEE-NEWCAS Conference*, 2005.

[3] R. I. Bahar and S. Manne. Power and energy reduction via pipeline balancing. In *International Symposium on Computer Architecture*, 2001.

[4] S. Balakrishnan, R. Rajwar, M. Upton, and K. Lai. The impact of performance asymmetry in emerging multicore architectures. In *International Symposium on Computer Architecture*, June 2005.

[5] S. Borkar, T. Karnik, and V. De. Design and reliability challenges in nanometer technologies. In *Design Automation Conference*, June 2004.

[6] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De. Parameter variations and impact on circuits and microarchitecture. In *Design Automation Conference*, June 2003.

[7] K. A. Bowman, S. G. Duvall, and J. D. Meindl. Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration. *Journal of Solid-State Circuits*, February 2002.

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

[9] Y. Cao and L. Clark. Mapping statistical process variation toward circuit performance variability: An analytical approach. In *Design Automation Conference*, June 2005.

[10] H. Chang and S. Sapatnekar. Full-chip analysis of leakage power under process variations, including spatial correlations. In *Design Automation Conference*, June 2005.

[11] R. Chau, S. Datta, M. Doczy, J. Kavalieros, and M. Metz. Gate dielectric scaling for high-performance CMOS: from SiO2 to High-K. In *IWGI*, 2003.

[12] T. Chen and J. Gregg. A low cost individual-well adaptive body bias (IWABB) scheme for leakage power reduction and performance enhancement in the presence of intra-die variations. In *Design, Automation and Test in Europe*, February 2004.

[13] T. Chen and S. Naffziger. Comparison of adaptive body bias (ABB) and adaptive supply voltage (ASV) for improving delay and leakage under the presence of process variation. *Transactions on VLSI Systems*, October 2003.

[14] L. Clark, E. Hoffman, J. Miller, M. Biyani, L. Liao, S. Strazdus, M. Morrow, K. Velarde, and M. Yarch. An embedded 32-b microprocessor core for low-power and high-performance applications. In *Journal of Solid-State Circuits*, volume 36, pages 1599–1608, November 2001.

[15] L. Clark, E. Hoffman, J. Miller, M. Biyani, Y. Liao, S. Strazdus, M. Morrow, K. Velarde, and M. Yarch. An embedded 32-b microprocessor core for low-power and high-performance applications. *Journal of Solid-State Circuits*, November 2001.

[16] N. Cressie. *Statistics for Spatial Data*. John Wiley & Sons, 1993.

[17] A. Datta, S. Bhunia, S. Mukhopadhyay, N. Banerjee, and K. Roy. Statistical modeling of pipeline delay and design of pipeline under process variation to enhance yield in sub 100nm technologies. In *Design, Automation and Test in Europe*, March 2005.

[18] R. Datta, A. Sebastine, A. Raghunathan, and J. A. Abraham. On-chip delay measurement for silicon debug. In *Great Lakes Symposium on VLSI*, April 2004.

[19] D. Ditzel. Power reduction using LongRun2 in Transmeta's Efficeon processor. In *Spring Processor Forum Presentation*, 2006.

[20] J. Donald and M. Martonosi. Power efficiency for variation-tolerant multicore processors. In *International Symposium on Low Power Electronics and Design*, pages 304–309, October 2006.

[21] J. Dorsey, S. Searles, M. Ciraula, S. Johnson, N. Bujanos, D. Wu, M. Braganza, S. Meyers, E. Fang, and R. Kumar. An integrated quad-core Opteron processor. In *International Solid State Circuits Conference*, pages 102–103, February 2007.

[22] S. Dropsho, A. Buyuktosunoglu, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, G. Semeraro, G. Magklis, and M. L. Scott. Integrating adaptive on-chip storage structures for reduced dynamic power. In *International Conference on Parallel Architectures and Compilation Techniques*, 2002.

[23] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. Razor: A low-power pipeline based on circuit-level timing speculation. In *International Symposium on Microarchitecture*, December 2003.

[24] D. Folegnani and A. González. Energy-effective issue logic. In *International Symposium on Computer Architecture*, 2001.

[25] P. Friedberg, Y. Cao, J. Cain, R. Wang, J. Rabaey, and C. Spanos. Modeling within-die spatial correlation effects for process-design co-optimization. In *International Symposium on Quality Electronic Design*, March 2005.

[26] A. Ghosh, R. Rao, R. Brown, and C. Chuang. On-chip process variation detection and compensation for parametric yield enhancement in sub-100nm CMOS technology. IBM Austin Center for Advanced Studies, 2007.

[27] M. Hatzilambrou, A. Neureuther, and C. Spanos. Ring oscillator sensitivity to spatial process variation. In *First International Workshop on Statistical Metrology*, June 1996.

[28] S. Heo, K. Barr, and K. Asanovic. Reducing power density through activity migration. In *International Symposium on Low Power Electronics and Design*, August 2003.

[29] S. Herbert and D. Marculescu. Analysis of voltage/frequency island granularity in chip-multiprocessors. In *International Symposium on Low Power Electronics and Design*, August 2007.

[30] M. C. Huang, J. Renau, and J. Torrellas. Positional adaptation of processors: Application to energy reduction. In *International Symposium on Computer Architecture*, 2003.

[31] Z. Huang and M. Ercegovac. Effect of wire delay on the design of prefix adders in deep-submicron technology. In *Asilomar Conference on Signals, Systems, and Computers*, October 2000.

[32] C. J. Hughes. *General-Purpose Processors for Multimedia Applications: Predictability and Energy Efficiency*. PhD thesis, University of Illinois at Urbana-Champaign, 2003.

[33] C. J. Hughes and S. V. Adve. A formal approach to frequent energy adaptations for multimedia applications. In *International Symposium on Computer Architecture*, 2004.

[34] E. Humenay, D. Tarjan, and K. Skadron. The impact of systematic process variations on symmetrical performance in chip multi-processors. In *Design, Automation and Test in Europe*, April 2007.

[35] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *International Symposium on Microarchitecture*, pages 347–358, December 2006.

[36] International Technology Roadmap for Semiconductors (1999).

[37] International Technology Roadmap for Semiconductors (2006 Update).

[38] A. Journel and C. Huijbregts. *Mining Geostatistics*. Academic Press, 1978.

[39] I. Kadayif, M. Kandemir, and I. Kolcu. Exploiting processor workload heterogeneity for reducing energy consumption in chip multiprocessors. In *Design, Automation and Test in Europe*, February 2004.

[40] A. Kahng. The road ahead: Variability. *Design & Test of Computers*, May-June 2002.

[41] A. Kahng. How much variability can designers tolerate? *Design & Test of Computers*, November-December 2003.

[42] K. Kanda, K. Nose, H. Kawaguchi, and T. Sakura. Design impact of positive temperature dependence on drain current in sub-1-V CMOS VLSIs. *JSSC*, 36(10), 2001.

[43] T. Karnik, S. Borkar, and V. De. Probabilistic and variation-tolerant design: Key to continued Moore's Law. In *Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, February 2004.

[44] A. Keshavarzi, G. Schrom, S. Tang, S. Ma, K. Bowman, S. Tyagi, K. Zhang, T. Linton, N. Hakim, S. Duvall, J. Brews, and V. De. Measurements and modeling of intrinsic fluctuations in MOSFET threshold voltage. In *ISLPED*, 2005.

[45] W. Kim, M. Gupta, G.-Y. Wei, and D. Brooks. System level analysis of fast, per-core DVFS using on-chip switching regulators. In *International Symposium on High-Performance Computer Architecture*, February 2008.

[46] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. In *Science*, volume 220, pages 671–680, May 1983.

[47] S. Krishnamurthy, S. Paul, and S. Bhunia. Adaptation to temperature-induced delay variations in logic circuits using low-overhead online delay calibration. In *International Symposium on Quality Electronic Design*, March 2007.

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

[49] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar. Mathematically assisted adaptive body bias (ABB) for temperature compensation in gigascale LSI systems. In *Asia South Pacific Design Automation Conference*, January 2006.

[50] T. Kuroda and T. Sakurai. Body biasing. In S. Narendra and A. Chandrakasan, editors, *Leakage in Nanometer CMOS Technologies*. Springer US, 2006.

[51] J. Li and J. Martínez. Dynamic power-performance adaptation of parallel computation on chip multiprocessors. In *International Symposium on High-Performance Computer Architecture*, 2006.

[52] X. Liang and D. Brooks. Latency adaptation of multiported register files to mitigate variations. In *ASGI*, 2006.

[53] X. Liang and D. Brooks. Mitigating the impact of process variations on processor register files and execution units. In *International Symposium on Microarchitecture*, December 2006.

[54] S. M. Martin, K. Flautner, T. Mudge, and D. Blaauw. Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads. In *International Conference on Computer Aided Design*, November 2002.

[55] R. McGowen, C. A. Poirier, C. Bostak, J. Ignowski, M. Millican, W. H. Parks, and S. Naffziger. Power and temperature control on a 90-nm Itanium family processor. *Journal of Solid-State Circuits*, January 2006.

[56] S. Mukhopadhyay, H. Mahmoodi, and K. Roy. Modeling of failure probability and statistical design of SRAM array for yield enhancement in nanoscaled CMOS. *Transactions on Computer-Aided Design*, 24(12), 2005.

[57] J. Nakano. *Techniques to address unreliability and variability of computing systems*. PhD thesis, University of Illinois at Urbana-Champaign, 2006.

[58] S. Narendra, M. Haycock, V. Govindarajulu, V. Erraguntla, H. Wilson, S. Vangal, A. Pangal, E. Seligman, R. Nair, A. Keshavarzi, B. Bloechel, G. Dermer, R. Mooney, N. Borkar, S. Borkar, and V. De. 1.1V 1GHz communications router with on-chip body bias in 150 nm CMOS. In *International Solid-State Circuits Conference*, February 2002.

[59] G. Ono and M. Miyazaki. Threshold-voltage balance for minimum supply operation. In *International Solid-State Circuits Conference*, February 2003.

[60] M. Orshansky, L. Milor, and C. Hu. Characterization of spatial intrafield gate CD variability, its impact on circuit performance, and spatial mask-level correction. *Transactions on Semiconductor Manufacturing*, February 2004.

[61] S. Ozdemir, D. Sinha, G. Memik, J. Adams, and H. Zhou. Yield-aware cache architectures. In *International Symposium on Microarchitecture*, December 2006.

[62] A. Papoulis. *Probability, Random Variables and Stochastic Process*. McGrawHill, 2002.

[63] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 1988.

[64] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, 2006. http://www.R-project.org.

[65] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, K. Strauss, S. Sarangi, P. Sack, and P. Montesinos. SESC Simulator, January 2005. http://sesc.sourceforge.net.

[66] P. Ribeiro Jr. and P. Diggle. geoR: A package for geostatistical analysis. *R-NEWS*, 1(2), 2001.

[67] T. Sakurai and R. Newton. Alpha-power law MOSFET model and its applications to CMOS inverter delay and other formulas. *Journal of Solid-State Circuits*, April 1990.

[68] S. R. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas. VARIUS: A model of parameter variation and resulting timing errors for microarchitects. In *IEEE Transactions on Semiconductor Manufacturing*, 2008.

[69] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. In *International Symposium on Computer Architecture*, June 2003.

[70] A. Snavely and D. M. Tullsen. Symbiotic jobscheduling for a simultaneous multithreaded processor. In *Architectural Support for Programming Languages and Operating Systems*, November 2000.

[71] K. Srinivasan and K. S. Chatha. Integer linear programming and heuristic techniques for system-level low power scheduling on multiprocessor architectures under throughput constraints. *Integration VLSI*, 40(3):326–354, 2007.

[72] A. Srivastava, S. Shah, K. Agarwal, D. Sylvester, D. Blaauw, and S. Director. Accurate and efficient gate-level parametric yield estimation considering correlated variations in leakage power and performance. In *Design Automation Conference*, June 2005.

[73] A. Srivastava, D. Sylvester, and D. Blaauw. *Statistical Analysis and Optimization for VLSI: Timing and Power*. Springer, 2005.

[74] K. Stavrou and P. Trancoso. Thermal-aware scheduling: A solution for future chip multiprocessors thermal problems. In *EUROMICRO Conference on Digital System Design*, pages 123–126, August, 2006.

[75] B. Stine, D. Boning, and J. Chung. Analysis and decomposition of spatial variation in integrated circuit processes and devices. *Transactions on Semiconductor Manufacturing*, February 1997.

[76] H. Su, F. Liu, A. Devgan, E. Acar, and S. Nassif. Full chip leakage estimation considering power supply and temperature variations. In *International Symposium on Low Power Electronics and Design*, August 2003.

[77] D. Tarjan, S. Thoziyoor, and N. P. Jouppi. CACTI 4.0. Technical Report HPL-2006-86, HP Labs, 2006.

[78] Y. Taur and T. H. Ning. *Fundamentals of Modern VLSI Devices*. Cambridge University Press, 1998.

[79] R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas. Mitigating parameter variation with dynamic fine-grain body biasing. In *International Symposium on Microarchitecture*, December 2007.

[80] A. Tiwari, S. R. Sarangi, and J. Torrellas. ReCycle: Pipeline adaptation to tolerate process variation. In *International Symposium on Computer Architecture*, June 2007.

[81] E. Toyoda. DFM: device and circuit design challenges. In *International Forum on Semiconductor Technology*, February 2004.

[82] J. Tschanz, J. Kao, S. Narendra, R. Nair, D. Antoniadis, A. Chandrakasan, and V. De. Adaptive body bias for reducing impacts of die-to-die and within-die parameter variations on microprocessor frequency and leakage. *Journal of Solid-State Circuits*, February 2002.

[83] J. Tschanz, S. Narendra, A. Keshavarzi, and V. De. Adaptive circuit techniques to minimize variation impacts on microprocessor performance and power. In *ISCAS*, volume 1, pages 9–12, May 2005.

[84] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar. An 80-tile 1.28TFLOPS network-on-chip in 65nm CMOS. In *International Solid-State Circuits Conference*, 2007.

[85] S. Xiong et al. Is gate line edge roughness a first-order issue in affecting the performance of deep sub-micro bulk mosfet devices? *IEEE Transactions on Semiconductor Manufacturing*, 17(3):357–361, Aug 2004.

[86] Y. Zhang, D. Parikh, K. Sankaranarayanan, K. Skadron, and M. Stan. HotLeakage: A temperature-aware model of subthreshold and gate leakage for architects. Technical Report CS-2003-05, University of Virginia, March 2003.

[87] W. Zhao and Y. Cao. New generation of predictive technology model for sub-45nm design exploration. In *International Symposium on Quality Electronic Design*, 2006.

AUTHOR'S BIOGRAPHY

Radu Teodorescu was born in Cluj-Napoca, in the heart of Transilvania, Romania. From a very young age he wanted to become a doctor — the kind that treats diseases. Everything changed in eighth grade, when he discovered his first computer. His passion for the technical side of things has only grown since.

Radu received his Engineer Diploma in Computer Science from the Technical University of Cluj-Napoca where he did research in medical imaging. He continued his studies in Urbana-Champaign where he received his M.S. and Ph.D. degrees in Computer Science from University of Illinois. His graduate research has been focused on computer architecture. He developed solutions for efficiently enhancing the reliability of software and proposed techniques for ensuring the continued improvement in microprocessor performance and power consumption in the face of increasing technological challenges.

Radu has co-authored over 15 research papers and has received awards for his thesis work and for his research in computer architecture. He has also held an Intel Foundation Fellowship during part of his graduate studies. After receiving his PhD he joined the Department of Computer Science and Engineering at Ohio State University as an Assistant Professor.