2011

Aggressive and reliable high-performance architectures - techniques for thermal control, energy efficiency, and performance augmentation

Prem Kumar Ramesh
Iowa State University

Aggressive and reliable high-performance architectures - techniques for thermal control, energy efficiency, and performance augmentation

by

Prem Kumar Ramesh

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Engineering

Program of Study Committee:
Arun K Somani, Major Professor
Akhilesh Tyagi
David Fernandez-Baca
Zhao Zhang
Joseph Zambreno

Iowa State University

Ames, Iowa

2011

To my dear mom, Chandra, to my dear late dad, Ramesh, and to my dearest brother Vasanth

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

This is a great opportunity to express my gratitude to those, who turned out to be rather helpful during my dissertation writing process, and supported me during the course of my doctoral program. I would like to acknowledge the particular debts I owe to those who taught me research skills and values, and who play an important role in my life.

First of all, I would like to thank my major professor Dr. Arun K Somani for his guidance and support during this course of study. I am grateful to have him as my advisor for his invaluable guidance, and I am thankful to him for giving me the freedom of choice to pursue a research direction that I was really interested in. I would also like to thank my committee members, Dr. Akhilesh Tyagi, Dr. Joseph Zambreno, Dr. Zhao Zhang and Dr. David Fernandez-Baca, for their constructive feedbacks and suggestions for this research project. I would like to acknowledge the professors at Iowa State, who trained me with sufficient skill set to pursue research in this field. I am also thankful to the ECpE department and Iowa State University for providing me with such a wonderful atmosphere for carrying out quality research.

I am very grateful to have a close-knit research group at Dependable Computing and Networking Lab. The innumerable number of discussions and seminars have always kept me motivated, and helped me to keep my graduate student life exciting. I thank all of my peers whom I collaborated with in developing various aspects of this thesis. I would like to specially thank Viswanathan Subramanian, Prasad Avirneni, Pavan Gorti, and Shubha Kher for their collaborations in various projects. I am pleased to express my gratitude to Viswanathan and Prasad for their support in a number of ways. I am thankful to my other research group colleagues Srivatsan, Mike, Nathan, Koray, Kamna, Jinxu, Ramon, Kritanjali, David,

Nishanth, Pavan, Lizandro, Zachary, Suresh and Harini for the many wonderful interactions we had.

Words cannot express my emotions and it is almost impossible for me to verbalize my friendship to Neeraj, Srivani, Vishwa and Abhijit. If not for their company, I would have found it extremely difficult to wade through the highs and lows of my stay in Ames. I owe my deepest gratitude to Arunkumar and Kabhilan for their amazing friendship and understanding. I am forever grateful to these awesome folks - Vichu, Shibi, Niranjan, Nikhil, Rokkam, Prasad, Sriram Aaleti, Harish Narayanappa, Rakesh, Arun, Visu, Kamala, Hari, Lavanya, Atul, Kamna, Puvi, Kavitha, Kritanjali, Muthu, KK, Misha, Satya Dev, Vasanth Balaramudu, Allada, Nishanth, Nikhil Ranade, Anupreet, Siva Sudani, Bharath, Rakesh Chandra, Sambit, Siva Konduri, Swami, Sankalp volunteers, coffee room chit chatters, and Friday evening volleyball gang. I would also like to take this opportunity to thank my friends back in India (whom I have known since my school days) - Logesh Raja, Rajasekar, Harish, Mukund, Achuthan, and Aswin Srinivasan, for their devoted and lasting friendship.

I express my heartfelt gratitude to my mother - a strong lady and my first Guru, for her love, affection, constant encouragement and care throughout. She has been my moral support, always giving me confidence through her golden words and short stories. I would like to thank my brother Vasanth, for his love and concern. He has been guiding me through his footsteps ever since I have known. He will continue to be my role model in my life. I am forever indebted to my mom and my brother, without whose support I would have never achieved what I have. Finally, I would like to show my gratitude to Payya mama (my uncle), who supported a part of my undergraduate education, and one of the very few good souls I have come across in my life.

ABSTRACT

As more and more transistors fit in a single chip, consumers of the electronics industry continue to expect decline in cost-per-function. Advancements in process technology offer steady improvements in system performance. The improvements manifest themselves as shrinking area, faster circuits and improved battery life. However, this migration toward sub-micro/nanometer technologies present a new set of challenges as the system becomes extremely sensitive to any voltage, temperature or process variations. One approach to immunize the system from the adverse effects of these variations is to add sufficient safety margins to the operating clock frequency of the system. Clearly, this approach is overly conservative because these worst case scenarios rarely occur. But, process technology in nanoscale era has already hit the power and frequency walls. Regardless of any of these challenges, the present processors not only need to run faster, but also cooler and use lesser energy. At a juncture where there is no further improvement in clock frequency is possible, data dependent latching through Timing Speculation (TS) provides a silver lining. Timing speculation is a widely known method for realizing better-than-worst-case systems.

TS is aggressive in nature, where the mechanism is to dynamically tune the system frequency beyond the worst-case limits obtained from application characteristics to enhance the performance of system-on-chips (SoCs). However, such aggressive tuning have adverse consequences that need to be overcome. Power dissipation, on-chip temperature and reliability are key issues that cannot be ignored. A carefully designed power management technique combined with a reliable, controlled, aggressive clocking not only attempts to constrain power dissipation within a limit, but also improves performance whenever possible.

In this dissertation, we present a novel power level switching mechanism by redefining the

existing voltage-frequency pairs. We introduce an aggressive yet reliable framework for energy efficient thermal control. We were able to achieve up to 40% speed-up compared to a base scheme without overclocking. We compare our method against different schemes. We observe that up to 75% Energy-Delay squared product (ED^2) savings relative to base architecture is possible. We showcase the loss of efficiency in present chip multiprocessor systems due to excess power supplied, and propose Utilization-aware Task Scheduling (UTS) - a power management scheme that increases energy efficiency of chip multiprocessors. Our experiments demonstrate that UTS along with aggressive timing speculation squeezes out maximum performance from the system without loss of efficiency, and breaching power & thermal constraints. From our evaluation we infer that UTS improves performance by up to 12% due to aggressive power level switching and over 50% in ED^2 savings compared to traditional power management techniques.

Aggressive clocking systems having TS as their central theme operate at a clock frequency range beyond specified safe limits, exploiting the data dependence on circuit critical paths. However, the margin for performance enhancement is restricted due to extreme difference between short paths and critical paths. In this thesis, we show that increasing the lengths of short paths of the circuit increases the margin of TS, leading to performance improvement in aggressively designed systems. We develop Min-arc algorithm to efficiently add delay buffers to selected short paths while keeping down the area penalty. We show that by using our algorithm, it is possible to increase the circuit contamination delay by up to 30% without affecting the propagation delay, with moderate area overhead. We also explore the possibility of increasing short path delays further by relaxing the constraint on propagation delay, and achieve even higher performance.

Overall, we bring out the inter-relationship between power, temperature and reliability of aggressively clocked systems. Our main objective is to achieve maximal performance benefits and improved energy efficiency within thermal constraints by effectively combining dynamic frequency scaling, dynamic voltage scaling and reliable overclocking. We provide solutions to improve the existing power management in chip multiprocessors to dynamically maximize system utilization and satisfy the power constraints within safe thermal limits.

CHAPTER 1. INTRODUCTION

Rapid advancements in process technology have revolutionized the way in which computing systems are built over the past several years. Conventionally, operating frequency has been the measure of choice to evaluate the performance of processors and system-on-chips (SoC). Power continues to be a first-class design constraint and the major limiter to the growth of system performance in the nanoscale era. Manufacturers are required to add guard-bands to the system clock frequency to guarantee reliable execution of digital systems.

In a pipelined processor, the clock frequency is determined based on the circuit critical path across all stages, under adverse operating conditions. However, the circuit propagation delay may change, as process, voltage and temperature variations are introduced during circuit fabrication. Traditional design methodologies for the worst-case operating conditions are too conservative as the critical timing delays rarely occur in tandem, during typical circuit operation. Moreover, circuit delay has a strong association with the data being processed and hence, not all instructions in a program under execution induce the worst-case delay. For instance, in a carry-propagate adder, the worst-case delay occurs only when the carry is to be propagated through each bit-slice, which occurs only for a specific input data set [1]. Such infrequent occurrence of critical timing delays has opened a new domain of study that allows improvement of processor performance to a greater extent through overclocking. Impressive results can be achieved using this technique. For example, a $2.8GHz$ $45nm$ AMD Phenom II processor running at speeds of up to $4GHz$ on air cooling alone have been reported [2]. Such is the interest with overclocking enthusiasts that chipset manufacturers are introducing technologies that support overclocking. AMD's Overdrive and Advance Clock Calibration technologies are cases in point. However, overclocking leads to system instability (i.e. system

crashes) and overheating which ultimately lead to unreliable systems. Reliable overclocking mechanisms strive to guarantee functional correctness by employing mechanisms to detect and recover from timing errors. A host of recent works explore the viability of reliable overclocking schemes to improve system performance beyond worst-case limits [3, 4, 5].

Although reliable overclocking mechanisms facilitate in improving performance, a major hurdle in realizing them is their impact on on-chip temperature. These techniques necessitate additional circuitry and also consume more power. Higher clock speeds and power densities invariably escalate on-chip temperature over a period of time. The problem becomes much more intensified due to the high performance requirements placed on the chip by the running applications. Moreover, higher power dissipation curtails the battery life in portable systems. In the case of high-end servers and high performance clusters, the effect is reflected in the cost of providing the cooling solutions. As systems operate faster, on-chip temperatures quickly reach and exceed the safe limits, causing localized hot spots in the chip that lead to system crashes and possibly causing device failures. This poses a serious threat to the reliability of these systems over the long run.

Dynamic Voltage and Frequency Scaling (DVFS) is a well studied system level on-line power and thermal management technique. Current products from both the leading microprocessor vendors, Intel and AMD, have dynamic thermal monitoring techniques that take necessary corrective actions to maintain on-chip temperature. Industry standards such as Intel SpeedStep, AMD PowerNow, and Transmeta Longrun technologies alternate between a set of predefined voltage and frequency pairs, and choose the best pair based on environmental conditions and processor workload. However, the reduction of frequency and the time taken for transition from one operating voltage-frequency set to another to maintain system temperature causes significant performance loss when executing applications that demand high performance. Techniques such as Razor [6] provides a design based on aggressive design methodologies that impose Dynamic Voltage Scaling (DVS) without altering frequency. It is imperative to mention here that such schemes still suffer moderate to significant performance degradation during voltage transition. Moreover, these techniques do not fully exploit the data dependence while adopting

timing speculation.

In this dissertation, we present a review of the current technology, what it offers to solve the above mentioned problems and discuss the various challenges it faces. The goal of this proposal is to render new ideas that overcome these challenges. Power and temperature play an important role on the lifetime of a reliably overclocked system. This proposal investigates the intricate inter-relationship between these parameters to guarantee the failure-free operation of the system.

The failure of a system can be classified as transient and permanent. Transient failures are temporary (e.g. glitches), however, permanent failures occur when the devices actually fail. Increase in temperature is an undesirable but unavoidable side-effect that results from manipulating frequency to enhance performance. This increase in temperature is one of the main contributors of system failures. Achieving maximal performance benefits within the thermal constraints by effectively combining DFS, DVS and reliable overclocking is one of the goals of this thesis. Our solution embraces aggressive design methodologies allowing errors to occur for performance benefits, while maintaining temperature within acceptable limits.

Driven by unending need for high performance and remarkable evolution in process technology, more than one processing core per chip has now become viable, thus paving the way for chip multiprocessors (CMP). It is clear that power management (and hence temperature regulation) is of utmost importance in CMP. DVARFS can easily be extended to cover CMPs as well. *Can DVARFS do anything at all to assist existing power management to improve system efficiency further?*. This research work answers this very question by developing an efficient task scheduler for maximal utilization within critical thermal limits.

1.1 Thesis Organization

The report opens with a brief introduction to all the research issues addressed in the dissertation; mainly it covers the need for temperature regulation and importance of energy efficiency in aggressively overclocked systems. Chapter 2 presents an overview of recent related researches published in literature. A brief treatment of the necessary background and detailed

technical reviews of some related works are subsequently provided. In Chapter 3, we introduce DVARFS and explain in detail about its functionality. We evaluate DVARFS with a variety of metrics and show how the technique is suitable for a spectrum of systems, from handheld devices to high performance processor systems. Following this is Chapter 4, where we explore theoretical limits and possibility of increasing the short-path delays for possible increase in performance enhancement through aggressive overclocking. In this chapter, we propose an algorithm that efficiently and controllably increases the contamination delay of the circuit. We also show our evaluation for benchmark circuits. Chapter 5 stresses the importance of efficiency of processor systems and introduces utilization as a metric for power constrained, high performance systems. In Chapter 6 we introduce our utilization-aware task scheduling in chip multiprocessors and explain in detail its working and evaluation. In this chapter, we also show how we built our chip multiprocessor simulator from a single core environment. Finally, in Chapter 7, the report closes with concluding remarks and presents brief ideas to possible further extensions and future works.

1.2 Thesis Contributions

The main goal of this dissertation is to investigate different system level techniques for thermal control, energy efficiency, and performance augmentation. Our motive is to provide viable techniques applicable to mainstream processors. In this work, we explore the potential solutions to overcome key challenges faced by the nanoscale technology era.

The seminal contributions of this thesis are as follows:

- Existing techniques to speed up the circuit operation rely on faster clock rates. Reliable overclocking is the concept of aggressively clocking the processor, allowing timing errors to occur, and recovering all those errors through timing speculation. Although aggressive clocking methodologies achieve the speed-up, they neglect the issue of overheating the chips. These techniques invariably rely on powerful cooling solutions that are quite expensive. This creates the need for an efficient thermal management scheme at the micro-architecture level to scrutinize on-chip temperature yet achieve as much high

performance. In this thesis, we study the impact of on-chip temperature on processor lifetime and we explore the possibility of achieving a desired lifetime by controlling the operating temperature.

- Traditional power management schemes work based on switching back and forth between different power levels that are determined offline. As we point out in our study, this poses as a huge overhead to achieve high performance. Hence, the power management schemes have to be re-defined to overcome this overhead. Contrast to traditional DVFS, where operating frequency is tuned to match circuit delay, we propose multiple predefined frequency levels, beyond conservative limits at any voltage level. We adopt a reliable and aggressive framework for our study. The system power level is now selected from a spectrum of frequencies (instead of one) for every voltage level. However, this incurs a cost due to timing error recovery. We develop *Dynamic Voltage, Aggressive and Reliable Frequency Scaling (DVARFS)* - an energy efficient thermal control mechanism using the redefined power level switching.

- Aggressive clocking systems have timing speculation as their central theme and operate at a clock frequency range beyond specified safe limits, exploiting the data dependence on circuit critical paths. One of the major factors limiting the degree of timing speculation is the contamination delay of the circuit. The margin for performance enhancement is restricted due to extreme difference between short paths and critical paths. This has been one of the major factors that restrict realizing timing speculation architectures in practical circuits. This requires a fast and efficient method during the synthesis process to manage the short paths by increasing their delay up to a threshold. In this thesis, we show that increasing the lengths of short paths of the circuit increases the margin of timing speculation, leading to performance improvement in aggressively designed systems. We develop *Min-arc* algorithm to efficiently add delay buffers to selected short paths while keeping down the area penalty. We explore the possibility of increasing short path delays further by relaxing the constraint on propagation delay, and achieve even higher performance.

- Multiprocessors in a chip have become a common commercial commodity in the last few years. Proficient power assignment across different cores is intriguing. Understanding thread power requirements and assigning power to the cores running these threads accordingly is the problem of our interest. We perform a study on limitations of existing power management solutions due to excess power supply. We bring out utilization as a factor for choosing a power level. We develop *Utilization-aware Task Scheduling (UTS)* an energy-efficient power management solution for CMPs. We add utilization metric constraint to the existing power constraints to schedule threads to the cores. We show that this model, along with aggressive, reliable framework will perform very differently than any of the existing power management techniques in improving the overall system efficiency provided timing errors are harnessed.

CHAPTER 2. BACKGROUND

The persistent CMOS scaling conforming to the 'Moore's law' has provided a steady improvement in cost-per-function. This has led to significant advancements in industrial and consumer electronics, and economic productivity. For instance, the current 45nm Intel i-7 quad core processor supports up to 3.3GHz clock frequency and is available in PC market [7]. Nevertheless, the process technology, beyond 45 and 32nm, is facing new challenges under the conventional path of technology scaling [8]. The present processors not only need to run faster, but also cooler and use less energy.

2.1 Clock Frequency in Nanoscale Technology

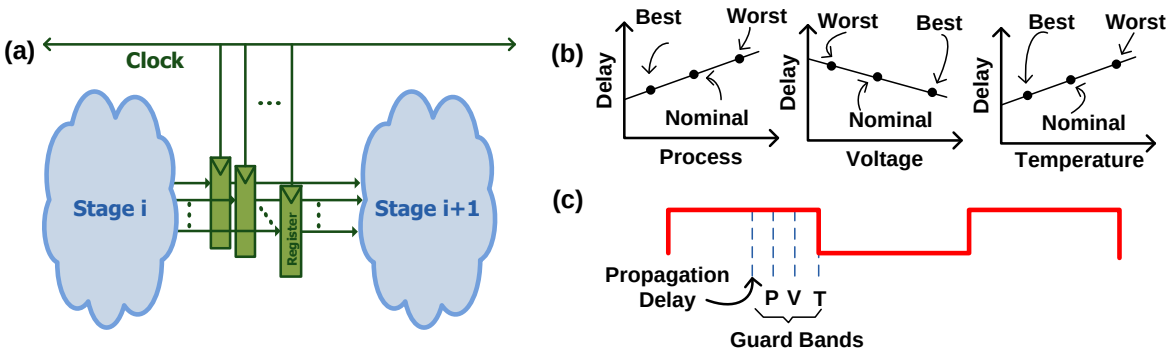


Figure 2.1 (a) Typical pipeline stage (b) Circuit delay with process, voltage and temperature variations (c) Clock period with guard bands

In a pipelined processor, the datapath consists of several stages, comprising dependent and independent blocks of combinational logic interleaved by set of register elements, as illustrated in Figure 2.1(a). Ideally, the clock frequency is determined by the circuit critical path across all

the stages. However, as device size shrinks further in the nanoscale era, it becomes increasingly complex to control the manufacturing process. This eventually leads to non-uniform circuit delay distribution across the chip.

The deviations are mainly reflected in three forms, namely, process, voltage and temperature variations. Figure 2.1(b) shows a general trend how the increase in variation of each of the category affects delay. From this, we observe that the best case design occurs with the least variations in the device dimensions, maximum voltage and lowest temperature. And, the worst case design is the one with maximum dimensional variation, lowest voltage and maximum temperature. During manufacturing, in order to guarantee correctness even at the worst scenario guard bands are added to the clock period, as shown in Figure 2.1(c).

The vendor-specified frequency includes a safety margin to provide tolerance for process variations, voltage fluctuations, extreme temperatures and power densities. However, such worst-case operating conditions and timing delays rarely occur in tandem during typical circuit operation. This had shifted the paradigm from worst case design methodology to design for common case. Thus, breaching the worst case design limits while guaranteeing correctness became the problem of interest in order to extract maximum performance out of the processor. Reliable overclocking allows embedded systems and processors to run at higher frequencies than the manufacturer specified worst-case frequency. For systems operating in typical operating environments, significant benefits can be achieved through overclocking, if reliable execution can be guaranteed.

2.2 Better-than-worst-case Designs

One of the earliest works on aggressive clocking, TEATIME [5] scales the frequency of a pipeline using dynamic timing error avoidance. This technique attempts to achieve better-than-worst-case performance by realizing typical delay operation rather than assuming worst-case delays and operating conditions. TEATIME achieves this by modeling a one-bit wide delay chain that reflects the worst-case critical path of the system, plus a safety margin. A prior work to this called TIMERTOL [9] exists in which, timing error tolerance is achieved by multiple

special copies of the pipeline logic. Similar architectures include CTV [10] and X-Pipe [11] that propose timing speculation at pipeline stage level.

The most significant aspect that can be exploited by reliable overclocking is the input data dependency of the worst-case delays. The worst-case delay paths are sensitized only for specific input combinations and data sequences [12]. Typically, the propagation delay of the digital system is much less than the worst-case delay and this can be exploited by overclocking. The benefits of overclocking can be furthered by allowing a tolerable number of errors to occur, and have an efficient mechanism to detect and recover from those errors. In addition to this, systems have different design restrictions, such as power, energy or area constraints. Based on all this, there are numerous architectures that have been proposed over the years.

Architectures without logic replication have been proposed at stage level. The basic idea is to duplicate latching; using shadow latches that always guarantees correctness. When a timing error is detected, it is recovered the following cycle. This technique along with dynamic voltage scaling has been used to improve energy efficiency [6]. Along with adaptive clocking mechanisms, reliable overclocking improves performance drastically [3]. In [13], the trade-off between reliability and performance is studied, and overclocking is used to improve the performance of register files.

Timing speculation has been well studied in the chip multiprocessors as well. Generally, these techniques couple two cores such that one of them is sped-up with the help of the other. The acceleration may be due to the execution hints provided by the advanced stream as in Slipstream [14], or in addition the advanced stream may be overclocked as in Paceline [15]. Here, the checker compares the results at checkpoints regularly. If there is a mismatch, the checker copies its current state to the leader.

Other works in the domain seek to improve common case performance through functionally incorrect design [16, 17]. The Selective Series Duplex architecture [18] consists of an integrity checking architecture for superscalar processors that can achieve fault tolerance capability of a duplex system at much less cost than the traditional duplication approach. DIVA [17] uses spatial redundancy by providing a separate, slower pipeline processor alongside the fast

processor. The desire for better than worst case designs is much more serious in nanoscale technology. PVT variations within and across the die are causing the bottleneck while selecting the worst-case frequency. ReCycle [19] uses additional registers and clock buffers to apply cycle time stealing in the pipeline, from faster stages to the slower ones. Another technique, EVAL [20] has been proposed to maximize performance with low power overhead in the presence of timing induced errors.

Apart from these run-rime schemes, there are static methods that are specifically developed for better than worst case architectures. The effect of parameter variations and its impact on timing errors has been studied in [21]. BlueShift [22] proposes a design methodology from ground up. The main idea is to identify and optimize the frequently used critical paths, called the 'overshooters' at the expense of the lesser frequent ones.

2.3 Impact of Scaling

Reliable overclocking methodologies address the timing error problem albeit other crucial factors exist that confine overclocking such as power and on-chip temperature.

2.3.1 Power Dissipation

The power consumed by a VLSI chip consists of two parts: dynamic and static. **Dynamic power** is consumed due to the clock switching activity. Dynamic power is dependent on capacitance (C), voltage (V), and frequency (f), and is given by Equation (2.1). The node transition activity factor, α is the effective power consuming transitions per clock cycle ($\alpha = 0.5$ for 50% duty cycle). Since power is directly proportional to frequency at which the circuit operates, overclocked systems consume more power than non-overclocked systems.

$$P_{dyn} = \alpha C V^2 F \tag{2.1}$$

Static power or leakage power is the inherent power consumed by the circuit even when the clock is stopped. The leakage increases proportionally with temperature, as given by Equation

(2.2) [23]. Here, β is a technology dependent constant (β is 0.036 and 0.017 for $180nm$ and $70nm$ respectively), T_0 is the temperature of a reference point and T_i is the temperature at i^{th} instant with respect to the reference point. Note that Equation (2.2) has a positive feedback: increase in temperature leads to higher leakage and total power, which in turn increases temperature.

$$P_{leak} \propto e^{\beta(T_i - T_0)} \tag{2.2}$$

Earlier, technologists considered dynamic power to be the major component of the total power consumed and ignored the static component, as static power dissipated had been far less significant. But now, it has been realized that, in deep sub-micron technology, this assumption is no longer valid, as leakage power has become a substantial constituent of total power dissipated. As power dissipation is proportional to quadratic of the operating voltage, scaling down voltage is an effective way of cutting down total power. However, scaling voltage slows down the circuit demanding increased clock period. This Dynamic Voltage and Frequency Scaling (DVFS) technique is a widely accepted method for power management, however, inevitably accompanied by significant performance overhead.

As mentioned above, increased power dissipation leads to escalated on-chip temperatures. Since cooling mechanisms are not cost effective, the necessity for a control mechanism built within the processor chips emerged as an economically viable approach. Designs began to include thermal sensors in various locations on a processor chip [24]. DVFS mechanisms were employed to manage temperature. As dynamic energy scales in quadratics with supply voltage, significant energy reduction is possible by lowering the supply voltage [25]. However, the resulting slow processor narrows the gap between high performance and low power [26]. Follow on research started to focus on design of thermally aware high performance processors aiming for minimal performance impact for specific applications [27, 28, 29]. Clock gating and voltage gating were developed to lower power dissipation during processor idle times and does not affect the performance.

Offline methods have the capability to deliver almost the same effect as dynamic schemes in thermal management. HotFloorplan involves thermal aware floor-planning, based on simulated

annealing [30]. Profile based static approach reduce peak processor temperature with relatively lower performance overhead. Another static approach proposes temperature aware design for low-power systems-on chip [31]. The design divides multiprocessor system on-chip into blocks using 3-D finite element analysis.

The need for low power architectures that deliver high performance while consuming as less power as possible is increasingly being felt by embedded system designers as they try to pack more and more power intensive computational tasks while curtailing their power budgets. Dynamic Voltage Scaling (DVS) is an approach that aims to bring down power without altering clock frequency. Razor architecture [6, 32] proposes such a design methodology. Here, the timing errors due to DVS are detected and corrected by additional checking circuitry. The voltage is dynamically scaled from the worst case settings, keeping track of the timing errors, until the number of timing errors exceeds a target error set point. Razor suffers a moderate performance cost because of reduction in voltage.

Industry standards such as Intel SpeedStep, AMD PowerNow, Transmeta Longrun technologies alternate between a set of predefined voltage and frequency pairs and choose the best pair based on worst-case voltage, temperature and process conditions. Correlating voltage controlled oscillator approaches have been proposed wherein the oscillator speed automatically adapts based on the supply voltage and generates the fastest safe clock speed [33, 34]. More aggressive power reduction can be achieved by tuning the supply voltage of individual processor chip using embedded inverter delay chains [35].

2.3.2 Thermal Impact on Lifetime Reliability

Higher temperatures not only increase power budget, but also affect the lifetime reliability of the devices. Several factors such as, rapid heating and cooling of processor chips create thermal cycles and localized heating, leading to hot spots, ultimately wearing out the circuits. To improve the overall reliability and lifetime of systems, the thermal performance of system should be monitored and the average degradation of the transistors should be managed [36]. RAMP [37] provides an architectural solution to the lifetime reliability problem. The

dynamic reliability management (DRM) presented in the paper ensures that target lifetime reliability is achieved. RAMP relates mean time to failure due to various wear out factors and brings the importance of on-chip thermal balance. Table 2.1 summarizes five critical failure mechanisms, namely, electromigration, stress migration, time dependent dielectric breakdown, thermal cycling and negative bias temperature instability as specified in RAMP, with their respective mean time to failure (MTTF). Here, k is Boltzmann's constant and T is temperature in Kelvin. These wear out phenomena create impedance in the circuits gradually leading to permanent device failures.

Electromigration occurs due to transport of material due to gradual movement of the ions in a conductor caused by the momentum transfer between electrons and the diffusing metal. Here, J is the interconnect current density. Activation energy, E_{aEM} and n are constants that depend on the interconnect metal used.

Stress Migration is a phenomenon that creates voids in the circuit, as a result of hydrostatic stress gradient. These voids may lead to high impedance or even break the circuit. This occurs due to difference in thermal expansion rates of materials. Again, E_{aSM}, m and the metal deposition temperature, T_{metal} are metal dependent constants. T_{metal} generally assumes a value far higher than circuit operating temperature. This means, $|T_{metal} - T|^{-m}$ increases with T. This is the reason why improving lifetime reliability is not as obvious as bringing temperature down.

Time dependent dielectric breakdown, also known as oxide breakdown occurs as a result of destruction of the gate oxide layer, and gradually leads to permanent transistor failure. Here, a, b, X, Y and Z are fitting parameters.

Sudden rise or fall in temperature causes **thermal cycles** which ultimately lead to device failure. Thermal cycles are caused by differences in thermal expansion rates across metal layers. Thermal cycling is proportional to the difference between current temperature and the ambient temperature $T_{ambient}$. Here, q refers to the Coffin-Mason exponent, which is empirically determined material dependent constant. From this definition, one could observe that sudden cooling of devices below $T_{ambient}$ worsens the lifetime reliability.

Wearout Mechanism	Proportional Model (MTTF)	Fitting Parameters
Electromigration (EM)[38]	$(J)^{-n}e^{\frac{E_{aEM}}{kT}}$; J=Current Density	n=1.1, E_{aEM}=0.9eV
Stress Migration (SM)[38]	$\lvert T_{metal}-T\rvert^{-m}\, e^{\frac{E_{aSM}}{kT}}$	m=2.5, E_{aSM}=0.9
Time dependent dielectric breakdown (TDDB)[39]	$(\frac{1}{V})^{(a-bT)}e^{\frac{[X+(Y/T)+ZT]}{kT}}$	a=78, b=-0.081, X=0.759eV, Y=-66.8eV/K, Z=-8.37e-4eV/K
Thermal Cycling (TC)[38]	$(\frac{1}{T-T_{ambient}})^{q}$; $T_{ambient}$=Ambient Temperature	q=2.35
Negative Bias Temperature Instability (NBTI)[40]	$\left[\left\{ln\left(\frac{A}{1+2e^{B/kT}}\right)-ln\left(\frac{A}{1+2e^{B/kT}}-C\right)\right\}\times\frac{T}{e^{-D/kT}}\right]^{\beta_1}$	A=1.6328, B=0.07377, C=0.01, D=0.06852, β_1=0.3

Table 2.1 MTTF for critical wearout models

Figure 2.2 shows how the increase in steady state temperature affects the processor lifetime. The proportionality constants are chosen assuming the baseline MTTF at $337K$ to be 30 years [37]. We use the reliability model to determine the critical temperature, for a target lifetime.

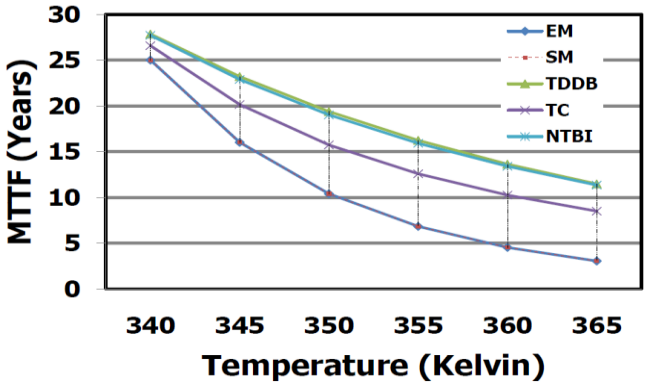


Figure 2.2 MTTF for different steady state temperatures

2.4 Evolution of Chip Multiprocessors and their Current Problems

The quest for higher performance requirements had paved the way for the micro-architectural innovations. Instruction Level Parallelism (ILP) is a key factor that decides the microprocessor performance. Wide issue superscalar processors extract the last ounce of performance from the

single threaded applications. Increasing the issue width beyond certain limit diminishes yield and also results in low power-performance efficiency. Several architectural innovations, such as Very Long Instruction Word (VLIW), multi-cluster superscalar and Simultaneous Multi-threaded (SMT) processors were proposed as an extension to the existing processor designs. Deeper pipelines offer faster clock frequency by dividing complex stages into number of sub-stages. For instance, Intel Pentium 4 processor has a twenty stage pipeline [41]. However, studies show that increasing pipeline depths may no longer hold for improving clock frequency [42].

Due to high power constraints, processor technology generations are unable to scale clock frequency as desired. Evolution of Chip Multiprocessors (CMP) eased power problems compared to other options. IBM introduced POWER4 [43] and POWER5 [44] architectures, which were the initial industry attempts of CMPs. Recently, there are many designs that have emerged in this direction, including Intel Montecito [45] and Sun Niagara [46], making CMPs the natural choice for low power and performance scalable architectures.

2.4.1 Thermal Management in CMP

Although multicore processors were designed to ameliorate the power related problems, due to technology's strict adherence to Moore's law, power density continues to increase. In addition to this, the temperature is not uniformly spread across the chip due to unbalanced workload across cores. This leads to localized hot spots at particular portions of the chip.

The single-core power management techniques, such as gating and DVFS cannot be directly applied to the multiple core scenarios due to the associated overhead. Also, a multicore scenario offers additional options such as thread migration that may reduce performance loss to some extent. It has been shown that independent per-core DVFS combined with thread migration improves performance up to 2.6X over a per-core gating [47]. Nevertheless, the effectiveness of DVFS is hampered by the slow voltage transitions. Incorporating on-chip regulators enables nanosecond scale voltage switching and can lead to significant energy savings [48].

As the number of cores increase in the CMPs, thermal management becomes non-trivial.

Workload characteristics, neighborhood temperature, and core locality play a vital role in deciding the core temperature. Based on this, many thermal aware task schedulers for CMPs have been proposed [49, 50, 51]. Recently, a thermally constrained power model has been proposed that maximizes the DVFS state in any interval, within a given power/thermal budget [52]. Although the resulting system outperforms reactive and ad-hoc voltage switching, maintaining the system at the maximum power state may not result in maximum efficiency.

Industry standards such as Intel SpeedStep, AMD PowerNow, and Transmeta Longrun technologies alternate between a set of predefined voltage and frequency pairs and choose the best pair based on environmental conditions and processor workload. However, the present day DVFS schemes involve a large overhead at the time of transition from one operating voltage-frequency set to another. This creates the necessity for a low-overhead solution, maximizing energy efficiency and working within the thermal constraints.

2.5 Timing Speculation Architectures

In this part of the chapter, we introduce some of the timing speculating architectures, explain their working and point out their positive and negative impact. We start by introducing briefly to an existing timing speculation framework for a pipelined processor. Processors that use reliable overclocking have this in-built error detection and recovery mechanism to deal with timing errors that may occur. We assume this error detection and recovery framework in the context of our proposed work. We discuss the application of timing speculation to boost single threaded performance. In the later sections of the chapter, we discuss how the most recent works handle these challenges in CMPs. We also bring out the fallouts of these schemes.

2.6 Local Fault Detection and Recovery

Reliable overclocking enables processor to work at frequencies past the worst-case limit, thereby causing timing errors to occur. Therefore, it is necessary to add built-in error detection and recovery mechanism. Local fault detection and recovery (LFDR) circuit we describe here is one such mechanism.

In a pipelined architecture, timing errors occur during overclocking when data takes longer time to propagate through the combinational logic stage, but the register at the end of the stage has already been clocked to latch the data. For a given combinational circuit, time for data to propagate through the stage depends on the data themselves and the supply voltage. Timing error occurs due to the mismatch between the circuit delay and the provided clock frequency. The LFDR circuit detects and corrects such errors with the help of an additional backup register controlled by a phase shifted (PS) clock. Thus, the circuit uses two clocks working at the same overclocked frequency.

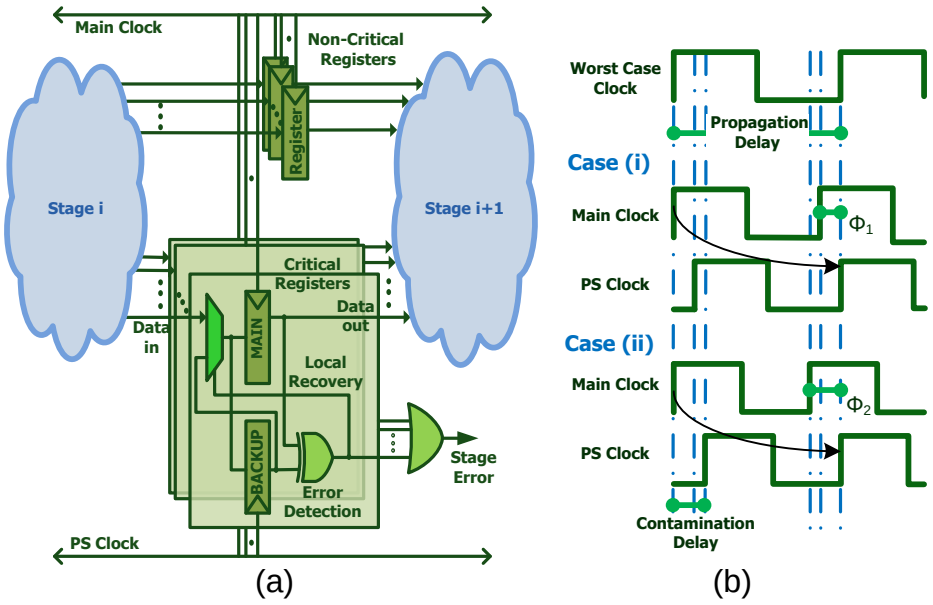


Figure 2.3 (a) Typical pipeline stage in a reliably overclocked processor (b) Illustration of aggressive main and PS clocks

A brief description of how these circuits work is presented here. The MAIN register is controlled by the *Main Clock* and the BACKUP register is clocked by the Phase Shifted *PS Clock*. Both *Main* and *PS Clocks* have the same frequency at all times, but are phase shifted as needed. The dense combinational logic has multiple inputs and outputs, and possibly with more than one path from each input to output. The circuit operates without timing errors, unless the time period of the tuned clock is insufficient for the active paths in the combinational logic

to reach the MAIN register. In that case, the MAIN register would have latched an erroneous value. However, the BACKUP register always latches the correct value, as long as the *PS Clock* is provided with the necessary phase shift. This is indicated by the arrow going from the *Main Clock* to the *PS Clock* in Figure 2.3(b). The amount of phase shift is such that, the time delay from the first rising edge of *Main Clock* to the second rising edge of *PS Clock* is not less than the propagation delay of the circuit. Also, it should be noted from Figure 2.3(b) Case (ii) that the maximum phase shift, and hence overclocking is limited by the contamination delay, which is the minimum amount of time beginning from when the input to a logic becomes stable and valid to the time that the output of that logic begins to change, of the circuit. In case, if the system is overclocked further, the BACKUP register is no longer guaranteed to latch the correct value.

When data latched in the MAIN register and the BACKUP register do not match, a local error signal is raised. In case of an error, a local recovery measure is taken by changing the control of multiplexer to select data from BACKUP register during the next cycle. The stage error signal is raised by performing logical OR of all local errors. All pipeline stages preceding this stage are stalled for a cycle, which is achieved through a global recovery mechanism. Moreover, all the stages following this stage process a bubble in a pipelined fashion. We discuss two applications of the LFDR circuit in the following subsections.

2.6.1 SPRIT^{3}E

The Superscalar PeRformance Improvement Through Tolerating Timing Errors (SPRIT^{3}E) was designed to dynamically tune superscalar processors beyond the worst case limit for enhancing their performance [3]. The number of timing errors that occur is directly proportional to the amount of frequency scaling. Therefore, by fixing the maximum number of errors in a time window, SPRIT^{3}E limits the timing errors under a budget. The framework is evaluated for an 18×18 multiplier implemented in FPGA. Limiting the timing error budget to a reasonable number, the LFDR implemented circuit can enhance the performance up to 44%. Similar experiment is performed for DLX superscalar processor, which is also synthesized for

the FPGA. By limiting the target error rate to 1%, SPRIT^{3}E achieves on an average 43%, and a maximum of 57% over the worst case settings.

2.6.2 Razor

Razor uses timing error tolerance in the context of Dynamic Voltage Scaling (DVS) [6, 32]. Razor explores the extent to which voltage can be scaled down at a given frequency. The goal of this work is to achieve increased energy reduction by eliminating the voltage margins. The proposed Razor technique was implemented in a 64-bit Alpha processor prototype at $0.18\mu m$ technology, operating at $200MHz$. The prototype was verified using simple programs. Analysis show that for the critical stages (Decode and Execute), only 192 Razor flip-flops out of a total of 2048 were used. Simulations were performed to analyze performance and power characteristics. In a 64-bit Alpha processor only 192 flip-flops out of 2048 flip-flops required Razor augmentation. Results show an average of 40% power reduction compared to traditional design that includes 3.1% energy overhead due to additional circuitry. There is a moderate performance overhead, around 3%, due to the voltage switching activity.

2.7 Thermal Consequences of Overclocking

Reliable dynamic clock frequency tuning for performance enhancement is incomplete without considering the thermal effects. Processors cannot be overclocked indefinitely, as this intensifies on-chip temperature. Thermal plots shown in Figure 2.4 compares a non-overclocked Alpha EV6 processor, running at $1GHz$ and an overclocked one, running at $2GHz$. We observed that steady state for dynamic reliable overclocking reached $380K$, while the non-overclocked settles at around $330K$. This necessitates an efficient scheme for thermal balance in reliably overclocked processors, which is part of the proposal's goal.

2.8 Facelift

Continuous workload activity causes wearing out of the devices in microprocessors. The device aging gradually results in slower circuits. Facelift attempts to hide and slow down aging

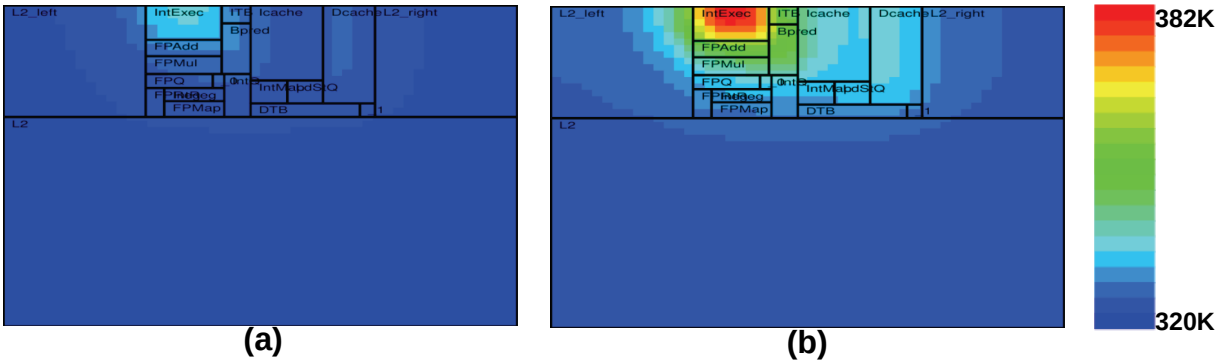


Figure 2.4 Steady state analysis (a) Non-overclocked (b) Reliably overclocked

in multicore processors through aging-driven application scheduling and appropriate voltage changes during the service life [53].

Facelift mainly focuses on the impact of aging on circuit critical paths. It takes into consideration two of the wear out phenomena, namely Negative Bias Temperature Instability (NBTI) and Hot Carrier Injection (HCI) that primarily affects PMOS and NMOS transistors respectively. The circuit slow down is directly proportional to the elevation of threshold voltage (V_t) of the transistors. The paper uses the alpha power law to model this. This is incorporated in the critical paths of the processor and the cache.

Facelift categorizes tasks into high-T and low-T jobs. Since cores do not age uniformly, the effect of aging is hidden by assigning high-T jobs to the faster cores and low-T jobs to slower ones. This aging-driven scheduling enables the chip to appear age less. Figure 2.5(A) shows the trends comparing slowest and fastest cores with traditional and aging-driven scheduling.

The impact of aging is slowed down via chip-wide Adaptive Supply Voltage (ASV) and Adaptive Body Bias (ABB). ABB is further classified into Forward Body Biasing (FBB) and Reverse Body Biasing (RBB) based on the voltage polarity. Similarly, ASV is classified into ASV+ and ASV- depending on V_{dd} value. Hence, combing RBB and ASV- results in slower circuits and hence causes slower aging. This technique called *SlowAge*. The second option is to combine FBB and ASV+ for faster circuits. Hence called *HighSpeed*. Figure 2.5(B) shows

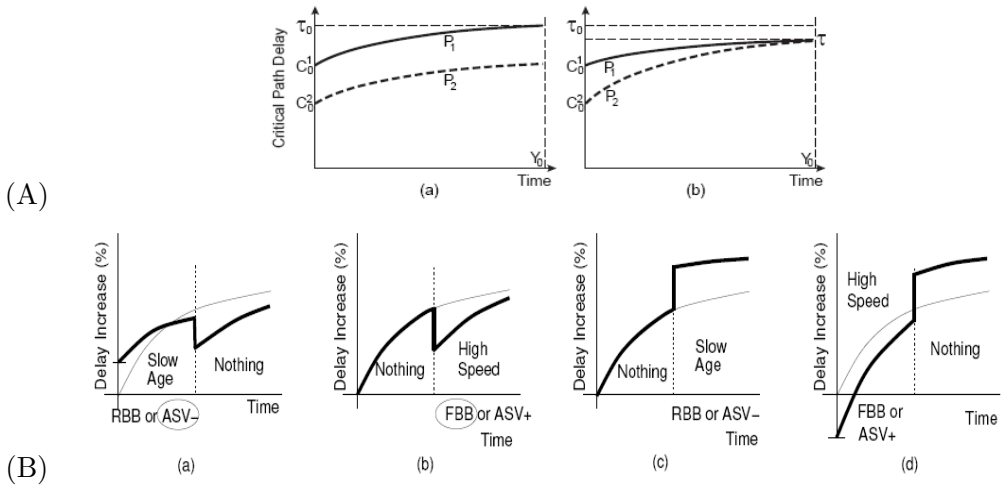


Figure 2.5 (A) Application scheduling effects (a) Traditional (b) Aging–driven (B) Applying techniques that change the aging rate

application combination of these techniques and expected service life. From the empirical model used, it is consequently observed that it is best to apply *SlowAge* at the beginning of the service life and *HighSpeed* towards the end.

The evaluation architecture is modeled for a CMP at $32nm$ with 16 cores, running at $4GHz$. Each core models a 4-issue out-of-order Alpha 212624 processor. Simulation results show that by hiding and slowing down aging, a 7 year service life processor can run 14-15% higher frequency. Alternatively, Facelift also enables processors designed for 5-7 months service life and still use it for 7 years. Implementation of Facelift is quite simple. It only involves re-calculation guard bands for hiding aging. Current technology also supports the ABB and ASV techniques that are used for the slowing down aging.

The basic idea behind aging-driven scheduling is to make slowest core age slowest and faster core age fastest. This suggests scheduling hot tasks to fast cores and cold tasks to slow cores. This approach may not be efficient from thermal point of view. In other words, this static scheduling may lead to hot spots. Further, Facelift does not take into consideration the neighborhood thermal impact on core aging. This cannot be done unless there is a dynamic scheme that keeps track of temperature online. The technique can be further improved as

a dynamic scheme to control aging, considering input dependence and making use of PVT guards bands.

2.9 Thread Motion

Thread Motion (TM) is an alternative technique to DVFS for fine grained power management in multi-core systems [54]. The aim is to increase system throughput by extracting the maximum out of a given set of applications at a given power budget.

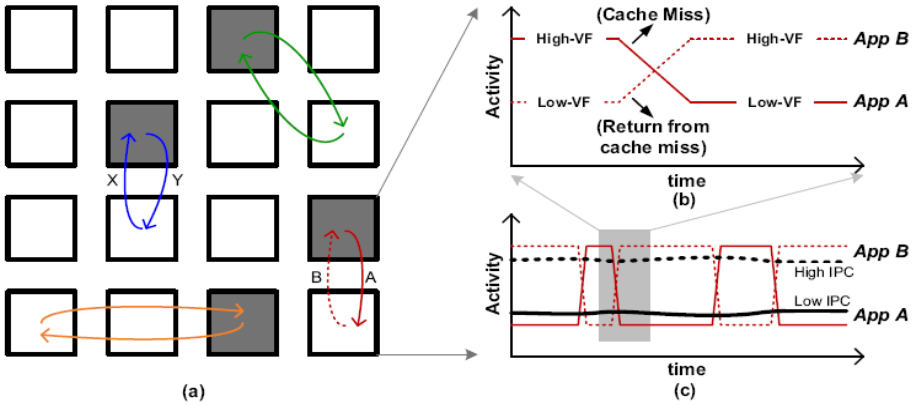


Figure 2.6 (a) Thread motion in a multicore system. (b) Exploiting fine–grained application variability in two running threads. (c) Duty cycling between 2 VF levels to match IPC

Traditional DVFS are OS driven and are hence too slow to fine variations in the program behavior. Thread Motion works under the basic premise that individual cores work at different voltage-frequency (VF) levels. Thread Motion allows threads to migrate between cores according to the current state of threads and cores. Figure 2.6 illustrates the Thread Motion phenomenon. It uses variability per instructions as a measure of application variability. It is calculated from the difference in IPC between sampling intervals.

Thread Motion involves two approaches: $time-driven$ and $miss-driven$. As their name suggest, TM is invoked on a regular time (cycles) and number of cache misses respectively. TM assumes a clustered multicore architecture, where inter-cluster TM is more expensive and less

frequently used compared to intra-cluster TM. The TM manager runs in a separate embedded microcontroller and runs the TM algorithm. The algorithm is a simple, cost-benefit analysis performed in irregular intervals. Implementing TM involves additional cost for inter-cluster cache penalty, prediction and register file latency. In spite of all these, TM with two VF levels performs in par with traditional DVFS schemes. Further for a given power budget, TM provides up to 20% better performance than coarse coarse-grained DVFS.

Although the potential performance benefits of TM are quite obvious, it suffers some limitations. First, irregular VF domains cause unbalanced heat distribution across cores. This difference in VF levels across cores may affect the life time of the cores in the long run. One way to overcome this effect is to combine DVFS with TM instead of using it as an alternative. Secondly, implementing TM for non-shared caches and complex cores are quite complex. It would involve an additional engine that might pose as an overhead from performance and power point of view.

CHAPTER 3. DYNAMIC VOLTAGE, AGGRESSIVE AND RELIABLE FREQUENCY SCALING

Persistent CMOS scaling has led to significant progress in industrial and consumer electronics, and economic productivity [7]. Nevertheless, the process technology, beyond 45 and 32nm, is facing new challenges under the conventional path of technology scaling [55]. As IC chips get denser, consumers of the electronics industry can expect continuous decline in cost-per-function. Process technology in nanoscale era has already hit the power and frequency walls. In spite of all these hurdles, the processor industries not only aim to build faster circuits, but also cooler and energy efficient one. At a juncture where there is no further improvement in clock frequency is possible, data dependent latching through timing speculation provides a silver lining. A carefully designed power management technique combined with a reliable, controlled, aggressive clocking not only attempts to constrain power dissipation within a limit, but also improves performance whenever possible.

In this chapter, we present a novel power level switching mechanism by redefining the existing voltage-frequency pairs. We introduce an aggressive yet reliable framework for energy efficient thermal control. We were able to achieve vast improvements in performance compared to a base scheme without overclocking. We compare our method against different schemes using other metrics. We observe that our solution provides huge Energy-Delay squared product (ED^2) savings, with controlled on-chip temperature. In short, we develop and evaluate a thermally constrained, reliable and energy efficient high performance system.

3.1 Design and Implementation of DVARFS

In the previous chapter, we emphasized the consequences of overclocked circuits. It is necessary to throttle overclocking based on the on-chip temperature. We developed a thermal constrained reliable overclocking technique that ameliorates system power and lifetime reliability [56]. This initial exploration has opened up a direction towards developing a powerful thermal management scheme that enhances performance as much as possible while operating well within the thermal limits, guaranteeing an extended system lifetime. We alter the existing DVFS technique to support reliable overclocking, and unify them by a common framework. We call scheme, DVARFS - *Dynamic Voltage - Adaptive and Reliable Frequency Scaling.*

3.1.1 Voltage-Frequency Feedback Control System

The control system encompasses two global feedback systems, one for controlling reliable overclocking and the other to regulate voltage. The two feedback systems and their interplay is illustrated in Figure 3.1. The major components of the control system are the voltage controller (VC), voltage regulator (VR), clock controller (CC) and clock generator (CG). VC works based on the readings from a thermal sensor. Voltage is lowered to bring down the temperature when sensor temperature exceeds critical limits, and when the system is below critical temperature, voltage is scaled up. VC assigns new voltage, V, to VR and corresponding base frequency, F to the controller CC.

At every voltage level, CC dynamically tunes clock frequency based on the number of timing errors reported by the error counter. CG provides the *Main* and *PS clocks* to the enhanced pipeline for timing error recovery. CG receives two inputs from the clock controller, namely, the new frequency ($F + \Delta F$), and the corresponding amount of phase shift (Φ). ΔF can be positive or negative depending upon whether the base frequency needs to be increased or decreased. The phase shift, Φ is calculated based on F and ΔF. Additional control is necessary to freeze the pipeline stage during frequency scaling and flushing the pipeline during voltage scaling (to save power during stall cycles).

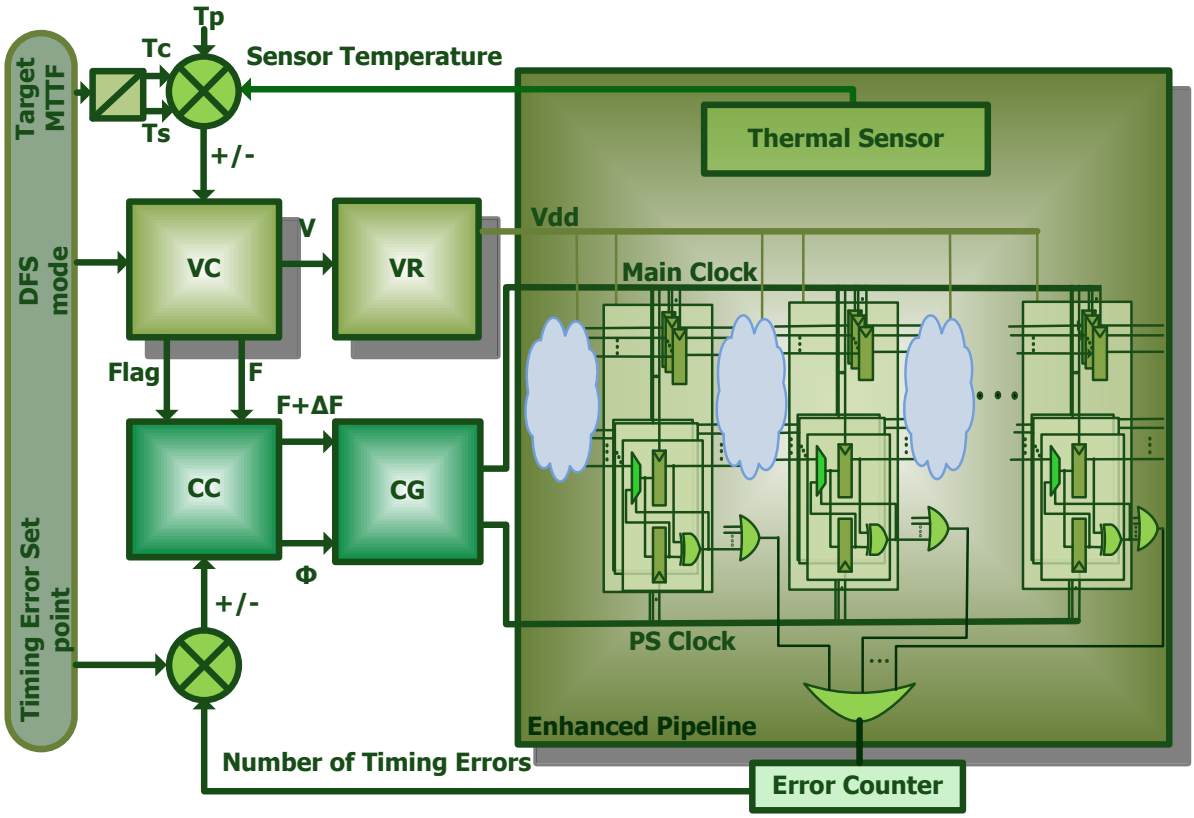


Figure 3.1 Temperature and timing error control loop

3.1.2 Analyzing Aggressive Clocking Systems

3.1.2.1 Error Rate

Aggressive clocking comes with the price of recovering timing errors during typical circuit operation. For any practical benefits, it is necessary to fix a bound for overclocking, as every error induced imparts overhead in terms of additional recovery cycles. Let t_{no} denote the current time period and t_{ov} denote the time period after overclocking. Let t_{diff} be the difference in time between original and next time period. Then, to execute n clock cycles, the total execution time is reduced by $t_{diff} \times n$, when there is no error. Let S_e, k and t_{pll} denote the fraction of clock cycles affected by errors, error recovery cycles and time for PLL to lock next frequency respectively. Then, equation 3.1 gives the bound on the timing errors that can be

tolerated without adding overhead.

$$S_e < \frac{t_{diff}}{t_{ov} \times k} - \frac{t_{pll}}{n \times t_{ov} \times k} \tag{3.1}$$

We dynamically switch between different discrete voltage settings and vary frequency in a given range at the current voltage setting. The two independent feedbacks allow us to set different sampling intervals based on their respective switching penalties. Current products, such as IBM PowerPC 750GX processors use dual PLL scheme for clock generation to perform dynamic power-performance scaling [57]. This allows instant frequency switching, when frequency sampling interval is greater than t_{pll}.

3.1.2.2 Speed-up

During overclocking, the clock frequency of the memory is not scaled, thereby increasing the total number of execution cycles. Let each memory operation take C_m cycles at t_{no} and q be the factor by which the frequency is scaled i.e., ($q = \frac{t_{no}}{t_{ov}}$). Now, after overclocking each memory operation takes $q.C_m$ cycles.

Let us assume that the system takes n clock cycles without considering memory cycles. If α denotes the factor of memory accesses that happen when the system executes n cycles. Then, the new execution time due to reliable overclocking is given by:

$$Ex_{ov} = n.t_{ov} + n.\alpha.q.C_m.t_{ov} + n.S_e.k.t_{ov} \tag{3.2}$$

To express original runtime (Ex_{no}) from Eqn 3.2, we replace t_{ov} by t_{no} and substitute $q = 1$ & $S_e = 0$. The overall speed up is calculated as given by Eqn 3.3.

$$Speedup = \frac{Ex_{no}}{Ex_{ov}} = \frac{q \times (1 + \alpha.1.C_m)}{(1 + \alpha.q.C_m + S_e.k)} \tag{3.3}$$

From Eqn 3.3, it is clear that computational density of workloads has a direct impact on speed up. Detailed analysis on how workload attributes affect performance enhancement

margin is provided in Chapter 5.

3.1.2.3 Voltage-Frequency Pairs

The voltage controller supports a set of predefined discrete voltage levels. The supply voltage has a strong association with circuit delay. For our understanding, let us consider the empirical model for this as given by Eqn (3.4).

$$Delay = \frac{C.V^2}{2v_{SAT}C_{OX}W(V - V_T)^2} \tag{3.4}$$

Here, v_{SAT}, C_{OX} and W are technology dependent constants; C specifies the load the circuit drives; V and V_T are the system voltage and threshold voltage respectively ($V_T = 0.2398V$ for $45nm$ technology) [58]. Eqn (3.4) suggests that the time period provided should match this *Delay*.

In conventional DVFS, the frequency is reduced corresponding to the circuit delay at each voltage level. The voltage-frequency (VF) pairs are determined off-line during design phase for the worst-case settings. However, in our case it is necessary for us to determine the VF pair dynamically. One way to do this is to relate the number of timing errors with the circuit slow down, thereby relating to the capacitive load that can be driven for that time period. Rearranging Eqn (3.4) for V_{no}, t_{no} and V_{ov}, t_{ov} yield the following loads that can be driven respectively.

$$K_{no} = \frac{t_{no}(V_{no}-V_T)^2}{V_{no}^2},\ K_{ov} = \frac{t_{ov}(V_{ov}-V_T)^2}{V_{ov}^2}$$

Thus, the percentage slow down for the new VF pair with respect to the current one is given by Eqn 3.5.

$$\%SlowDown = \frac{K_{no} - K_{ov}}{K_{no}} \times 100 \tag{3.5}$$

Let us assume that at a given voltage, the system had settled down on a frequency that yields maximum performance under controlled error rate. During DVFS, the VF pair that has the same *%SlowDown* would bear the same error rate. As opposed to the traditional DVFS,

where the frequency for the corresponding voltage is set a priori (and cannot be changed), here the clock controller is still prone to alter the frequency subject to the occurrence of timing errors.

3.1.3 Temperature Throttling

For thermal control, we define a predefined temperature set point based on which, the voltage feedback control functions. The power down temperature, T_p is the maximum temperature the circuits can withstand (typically, $T_p = 105°C$). The critical temperature, T_c, is a preset temperature below T_p, for reliable operation of the circuit. We desire the system to stay within this limit, by switching back and forth the operating voltage.

3.2 Experimental Framework

To validate DVARFS technique, we use SimpleScalar simulator [59] for Alpha EV6 processor. Table 4.1 provides the baseline values for the simulator. Thermal sensor implementation is done using HotSpot 4.0 [60]. The instantaneous power trace to calculate temperature, is provided by Wattch power model [61] integrated within the SimpleScalar tool. The entire framework is shown in Figure 3.2.

3.2.1 Wattch-HotSpot Integration

Wattch is an accurate, architecture level power tool that is embedded within sim-outorder of the SimpleScalar simulator [61]. Wattch calculates the energy accumulated over the cycles. We modified the tool to track the instantaneous power for each functional block. We use HotSpot model to calculate temperature, an efficient, architecture level thermal modeling tool [60]. HotSpot requires the floorplan of the underlying processor, from which the temperature for each functional block is calculated, based on the instantaneous power values of corresponding blocks. We take the temperature output from HotSpot to calculate leakage power using Equation 2.2. The additional power due to error recovery circuits are included appropriately as reported by [6]. We sample instantaneous power from Wattch every cycle and track temperature variations

Parameter	Value
Fetch width	4 inst/cycle
Decode width	4 inst/cycle
Issue width	4 inst/cycle (ooo)
Commit width	4 inst/cycle
Functional units	4 INT ALUs 1 INT MUL/DIV 4 FP ALUs 1 FP MUL/DIV
L1 D-cache L1 I-cache L2 Unified	128K 512K 1024K
Technology node	$45nm$
Voltage	$1.25, 1.15, 1.00, 0.95V$
Base frequencies w.r.t voltages	$2536, 2475, 2402, 2316MHz$
No. of freq levels per voltage	32
Critical Temperature	$T_c = 363K$
Initial Temperature	$333K$
Temperature sampling	$1ms$
Freq sampling	$10\mu s$
Voltage penalty	$100\mu s$
Freq penalty	Single PLL: $10\mu s$ Dual PLL: $0\mu s$

Table 3.1 Simulator parameters

in HotSpot. We make use of dynamic thermal variations for our feedback control. We use transient analysis for our experiments rather than steady state analysis. We establish our experiments with current state of the art by designing our simulations for the $45nm$ technology. We adopted the scaling parameters from the parallel multicore version of the SimpleScalar PowerPC simulator [62, 63].

3.2.2 Incorporating Timing Errors

We bring in the aspects of timing speculation in the SimpleScalar simulator from a hardware model. We used the Illinois Verilog Model (IVM) - a Verilog RTL implementation of the Alpha microprocessor. Since the IVM is not fully synthesizable, we had to synthesize and evaluate the individual pipeline stages. We used the $45nm$ OSU standard cell library for timing estimation

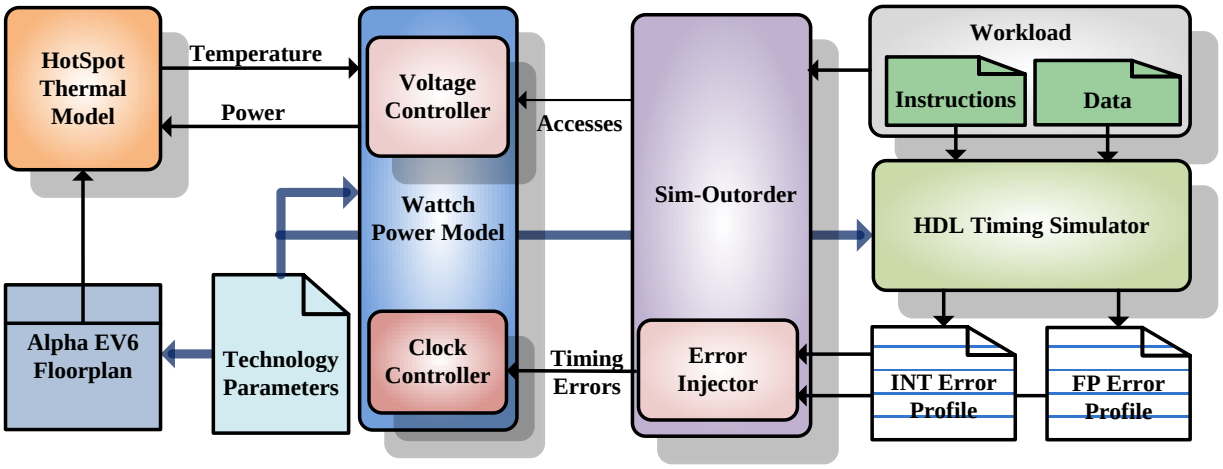


Figure 3.2 Simulation framework depicting feedback control of timing errors and temperature for clock tuning

[64]. Figure 3.4 shows the cumulative error rate for $SPECINT2000$ suite. We noticed around 89.17% of the paths fail in the issue stage at $3.5ns$, which causes a sudden rise in error rate. As IVM does not support float, we instrumented $SPECFP2000$ instructions and performed timing analysis for FP ALU obtained from opencores.org. We incorporate these reported error rate values in our functional simulator.

3.2.3 Incorporating Feedback Control System

With this extensive simulation framework environment, we propose to experimentally validate the claims of DVARFS scheme. Our goal here is to show that controlled reliable overclocking is indeed a beneficial way to enhance performance taking into consideration about thermal constraints.

3.3 Evaluation and Results

In this section we present the results of our simulation studies. Our goals in this evaluation are as follows:

1. To analyze the performance of the DVARFS with processor having no thermal control

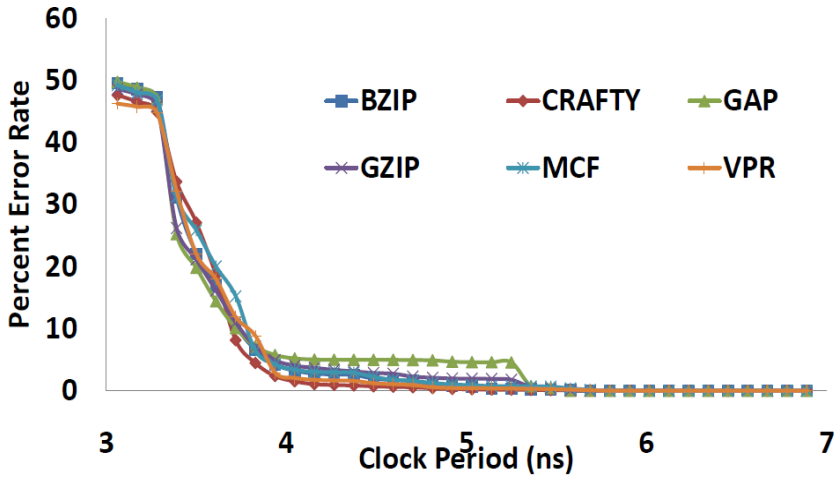


Figure 3.3 Timing error analysis for selected SPEC workloads

(*Simple*), traditional DVFS (*DVFS*), Reliably Overclocked Processor (*rop*), and Thermally throttled ROP (*trop*) schemes.

2. To examine the effectiveness of the thermal control in each of the above mentioned schemes (henceforth called modes). And,

3. To perform a comparative study of the average power and energy dissipation for different modes. In addition to this, we also measure energy delay product (EDP) and EDP delay product (ED^2).

For analyzing the thermal impact, we make a fair assumption that if the control scheme keeps the peak temperature of the processor below critical limits, then it achieves the target lifetime pertaining to it. We assume the critical temperature, $T_c = 363K$ corresponding to 10 year lifetime. All the normalized measures are relative to *Simple* mode.

3.3.1 Algorithms for Various Feedback Control Mechanisms

3.3.1.1 Base Case: *Simple*

We evaluate all the feedback control mechanisms relative to the base case of no thermal control scheme. We run sim-outorder without any modifications and compare all the other

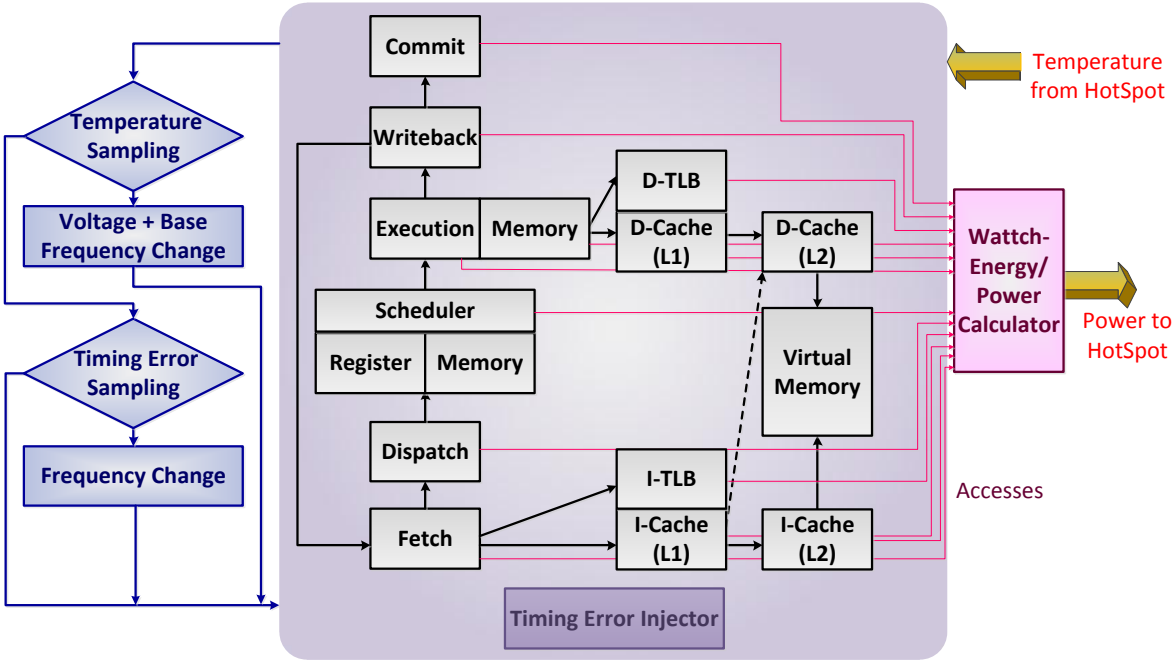


Figure 3.4 Illustration of feedback control system flow diagram and the main simulator loop in the framework. NOTE: The pipe stages are illustrated in reverse order as it is modeled in sim-outorder

mode performances with respect to this run. This allows us to normalize the results with a base system. We maintain the same processor configurations and technology parameters for all the modes. During any thermal emergencies the system is allowed to work beyond the critical limits. For the purposes of this study, we allow such scenarios to occur to get to know how effective the other thermal control methods are.

3.3.1.2 Dynamic Voltage and Frequency Scaling: *DVFS*

We implemented DVFS mechanism for thermal control in our simulation. The pseudocode to implement this scheme is given in Algorithm 1. When executing in this mode, the temperature is sampled once in several cycles, depending on the interval length. During temperature sampling intervals, this module checks if the maximum processor temperature (maximum temperature among all the functional blocks) exceeds the predefined critical temperature. If so,

the voltage controller is called to reduce the operating voltage one level below the present level. The voltage controller correspondingly re-assigns the clock frequency to the base level. If the processor is already running at the lowest possible voltage level, a 'Panic' signal is raised. During other times, when temperature is below the critical limits, the voltage level is increased one level up. Again, the frequency is increased to the new base frequency.

Algorithm 1 Traditional DVFS; can be modified into Razor by changing $VF\_PAIR[]$

if $((sim\_cycle - sim\_cycle\_oldv) > Vcycles)$ **then**
  $sim\_cycle\_oldv = sim\_cycle$;
  if $((max\_current\_temperature > Tcritical))$ **then**

    if $(vlevel > 0)$ **then**
      $vlevel - -$;
      $new\_voltage = VOLTAGE[vlevel]$;
      $new\_frequency = VF\_PAIR[vlevel]$;
      $dvfs(new\_voltage, new\_frequency)$;
      $Vpenalty(new\_frequency)$;
    else
      signal(PANIC); // Temperature exceeded at minimum voltage level; Nothing Can be done!
    end if
  end if
else
  if $(vlevel < (LEVELS - 1))$ **then**
    $vlevel + +$;
    $new\_voltage = VOLTAGE[vlevel]$;
    $new\_frequency = VF\_PAIR[vlevel]$;
    $dvfs(new\_voltage, new\_frequency)$;
    $Vpenalty(new\_frequency)$;
  else
    $signal(PANIC)$; // Already at Max Voltage; Nothing Can be done!
  end if
end if

3.3.1.3 Reliably Overclocked Processor: *rop*

In this mode we do not incorporate any thermal control mechanism. We replicate the same functionalities as SPRIT^{3}E through controlled overlocking. We keep track of the predefined,

programmable timing error set point. If the error rate reaches the set point frequency is increased. Otherwise, it is decreased. This mechanism is shown in Algorithm 2. The algorithm calls the subroutine, $FREQUENCY\_TUNING()$. This module takes care of calculating the next frequency level depending on the error rate. The pseudocode for this is described in Algorithm 3. Frequency is tuned up if the parameter passed is $+1$, and tuned down if it is -1. Note that although the algorithm involves three modes ($ARFSMODE$), we only implement mode $ARFSMODE = 1$ for *rop*.

Algorithm 2 DFS without thermal control; SPRIT3E like

```
if ((sim_cycle − sim_cycle_old) > Fcycles) then
  sim_cycle_old = sim_cycle;
  if (timing_error_counter > ErrorLimit) then
    FREQUENCY_TUNING(−1)
  else
    FREQUENCY_TUNING(+1)
  end if
end if
```

Algorithm 3 Dynamic frequency tuning: FREQUENCY_TUNING(tune)
// Takes parameter ±1

```
if (ARFSMODE == 1||ARFSMODE == 2) then
  if (x == +1)&&(flevel < FLEVEL − 1) then
    flevel + +;
  else if (x == −1)&&(flevel > 0) then
    flevel − −;
  end if
else if (ARFSMODE == 3) then
  if x == +1 then
    flevel = BINSRCH(flevel, FLEVEL);
  else if x == −1 then
    flevel = BINSRCH(0, flevel);
  end if
end if
Fpenalty(new_frequency); timing_error_counter = 0;
```

3.3.1.4 Thermally Throttled ROP: *trop*

In order to bring in thermal control in reliably overclocked processor, the most straightforward solution is to introduce frequency throttling during thermal emergencies, which is what we adopt. From Algorithm 4, it can be observed that the only point where this method differs from *rop* is while checking the condition. Here, there is an additional checking condition for maximum processor temperature. Further, to keep the control scheme simple, we assumed same sampling interval for thermal sensors and error counter.

Algorithm 4 Thermal control with only DFS (NO DVS/DVFS); Thermal control in SPRIT3E

---

if $((sim\_cycle - sim\_cycle\_old) > Fcycles)$ **then**
  $sim\_cycle\_old = sim\_cycle$;
  if $((max\_current\_temperature > Tcritical)||(timing\_error\_counter > ErrorLimit))$ **then**
    $FREQUENCY\_TUNING(-1)$
  else
    $FREQUENCY\_TUNING(+1)$
  end if
end if

3.3.1.5 Dynamic Voltage, Aggressive and Reliable Frequency Scaling: $dvarfs$

In DVARFS, there are mainly two control mechanisms involved, as mentioned before. The procedure is described in Algorithm 5. The thermal control loop checks for the sensor temperature during every sampling interval. If the temperature exceeds critical temperature, the voltage is stepped down. Otherwise, it is stepped up similar to DVFS. If the temperature exceeds the critical limits even at the lowest voltage level, then we start to step down frequency to the lowest level, disabling overclocking. This is a distinguishing feature of DVARFS compared to DVFS. In the cases where the temperature is below critical limits, we still have opportunity to improve performance through overclocking. In those cases, we keep track of the timing error counter separately and call the $FREQUENCY\_TUNING()$ sub-routine accordingly. We implement three modes ($ARFSMODE$) while doing frequency scaling, namely,

$ARFSMODE = 1$ or low-to-high frequency scaling ($lohi$), $ARFSMODE = 2$ or high-to-low frequency scaling ($hilo$), and $ARFSMODE = 3$ or binary search frequency scaling (bin), based on how we switch from one frequency level to the next. In $lohi$, the DFS is similar to that in rop, where we start at the lowest frequency level and progressively move to higher frequency if timing error occurrences are under the control limit. Mode $hilo$ is just opposite of $lohi$. $hilo$ is a very optimistic approach where we start at the highest frequency level and progressively step down until error rate is under threshold. In bin, we employ a binary search algorithm to find the optimal frequency level.

Algorithm 5 Dynamic voltage, aggressive and reliable frequency scaling

if $((sim\_cycle - sim\_cycle\_oldv) > Vcycles)$ **then**
  $sim\_cycle\_oldv = sim\_cycle$;
  if $((max\_current\_temperature >= Tstepdown))$ **then**

    if $(vlevel > 0)$ **then**
      $vlevel - -$;
      $Vpenalty(new_frequency)$;
    else
      $signal(PANIC)$; //Temperature Exceeded at Minimum Voltage Level; Bringing Frequency Down!
      $FREQUENCY\_TUNING(-1)$
    end if
  else

    if $(vlevel < (LEVELS - 1))$ **then**
      $vlevel + +$;
      $Vpenalty(new\_frequency)$;
    end if
  end if
else if $((sim\_cycle - sim\_cycle\_old) > Fcycles)$ **then**
  if $(timing\_error\_counter > ErrorLimit)$ **then**
    $FREQUENCY\_TUNING(-1)$
  else
    $FREQUENCY\_TUNING(+1)$
  end if
end if

We simulated six SPEC INT 2000 workloads, namely $bzip2, crafty, gap, gzip, mcf$ and

vpr and seven SPEC FP 2000 workloads, namely $applu, apsi, equake, galgel, lucas, mesa$ and *mgrid*, to analyze the experiments with the system deploying the DVARFS scheme. For all the modes that involves timing speculation, we assume the timing error set point to be 5%. That is, we do not allow more than 5 timing errors in 100 cycles. The initial steady state thermal states are assumed to be same, and equal to *Simple* scheme. All other simulator parameters are illustrated in Table 4.1.

3.3.2 Performance

From Equation 3.3, we measured the speed up for all the modes with *simple* mode as base. The speed-up charts are illustrated in Figure 4.3. In general, integer workloads perform better compared to FP workloads due to lesser computational density of the former over latter. It is evident that DVARFS in general performs better than *simple* and other schemes. Especially, *hilo* outperforms all other schemes across all workloads. This is because *hilo* starts with the maximum frequency and slows down according to the error occurrences. *bin* performs moderately well, with performance gain across all workloads. *lohi* performs well for few integer workloads, viz., *bzip*2 and *gzip*. However, the loss of performance in *lohi* mode is found to be not more than 3% in both integer and floating point workloads. This suggests that typical workloads (integer and floating point) tend to have very low occurrences of timing violations that show up as errors for most of the operating frequencies (past worst-case limits). This is quite evident from *hilo* performance. It is because of the same reason that *bin* shuttles between acceptable and unacceptable frequency ranges leading to lower performance improvement compared to *hilo*. For *lohi*, by the time it reaches optimal frequency, it has already lost the much of the chance for performance enhancement. Moreover, in this time the processor will also eventually approach thermal emergency causing voltage scaling. This happens over and over throughout execution resulting in a very small opportunity for enhancement. Using DVARFS it is possible to achieve up to 40% performance improvement including the stall cycles for the 5% error rate.

The most widely accepted power management scheme, DVFS ($dvfs$), suffers a moderate

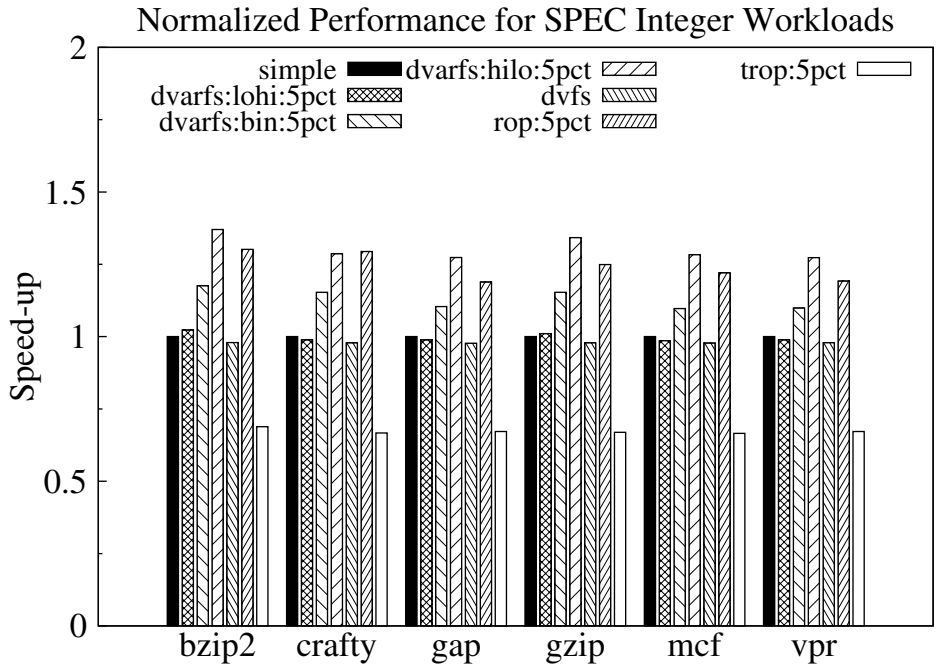
performance loss around $4-5\%$ across all workloads. This is a smaller number compared to the typical dvfs, as we consider only thermal emergencies to be the controlling factor (DVFS is generally applied for controlling power/energy dissipated within a budget).

Reliable overclocked processor (*rop*) performs best only next to *hilo*. It is normally expected that *rop* to outperform all the other modes, as it does not involve thermal control loop. The only thing to restrict its performance is the frequently occurring timing errors. Actually, the lower speed-up compared to *hilo* is majorly due to the fact that *rop* considered here is the simple ROP [65] which works similar to *lohi*. An improved version of ROP is possible and can be expected to perform better than other modes, but is beyond the scope of the thesis.

We tried to bring in thermal control in ROP (*trop*) where the frequency is throttled during both timing errors and thermal emergencies. However, results show that there is a significant performance loss by doing this. The performance loss reflects how often thermal emergencies occur during workload running. It also shows that scaling voltage down is a more effective way of controlling temperature for longer running time (cubic reduction in dynamic power as opposed to linear reduction). Note that we assumed all the modes (wherever applicable) to run for 5% target error rate [66]. *trop* has its own advantages, especially when processor lifetime is the major concern in the absense of voltage scaling, as we shall see in the later part of this section.

3.3.3 Power

The average power is normalized to *simple* mode as shown in Figure 3.6. It is quite obvious why *rop* has power consumption a lot higher compared to all other modes, as *rop* does not have a control loop for power or temperature. There is not much difference in its behavior from integer to floating point workloads. Power consumption for DVARFS is $25-30\%$ lower compared to simple scheme. This shows how DVARFS, in addition to thermal controlling scheme, is effective as a power management scheme. Among the three dvarfs mode, *bin* dissipates the least power as the mode can easily switch between lower (safe) and higher (unsafe) frequency ranges. *lohi* follows next, as it operates relatively at smaller frequencies



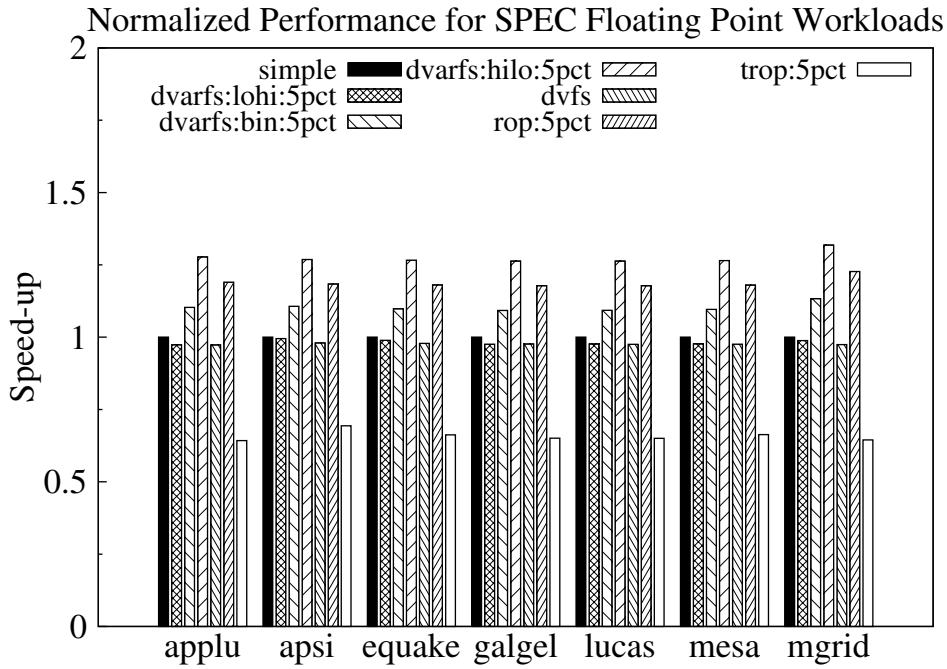


Figure 3.5 Speed-up chart for SPEC INT and FP workloads

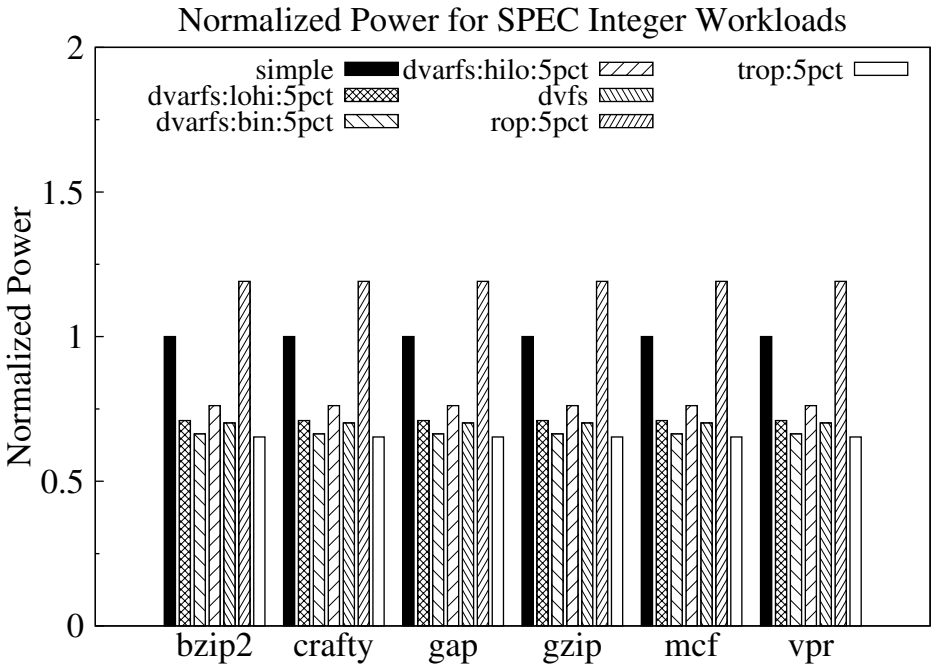
most of the running time. *hilo* consumes higher power compared to the former two as it spends most running time in highest frequency levels. However, it manages to converge to an optimal range without affecting the overall power dissipation. $dvfs$ saves up to 30% power. It is important to note here that *bin* saves much more power than $dvfs$, and *lohi* performs in par with $dvfs$. Although from speed-up perspective *trop* doesn't add much, in terms of power consumption *trop* performs as good as *bin*; in some cases saving more than what *bin* does.

3.3.4 Energy (PDP)

Power delay Product (PDP) measures the energy dissipated during execution. Again, the PDP reported is the average PDP, normalized with respect to *simple* and is depicted in Figure 3.7. A quick look back, we observed that *trop* is one of the most efficient power saving modes. However, the power delay product reveals that the delay costs much more than the power saved. In other words, the energy consumed by *trop* now performs in par with *rop*, which again does not save energy any more than *simple* mode. *trop* manages to consume $2-3\%$ more energy than *rop* itself across all workloads. It is very important to note here that DVARFS saves more energy than DVFS ($dvfs$), the latter being the most widely used scheme for energy saving. *lohi* saves as much energy as $dvfs$, while *bin* and *hilo* save much more. Quite evidently, DVARFS outperforms all the other modes in optimizing the tradeoff between power and delay.

3.3.5 EDP

Energy Delay Product (EDP) is a widely used metric when the emphasis on delay is an order higher than the power dissipated. In other words, EDP gives the tradeoff between energy and delay. The EDP for all workloads are shown in Figure 3.8. In this, *trop* expends almost 50% more than simple mode in this trade off. Specifically, for integer workloads, crafty, gap, mcf and vpr, *trop* touches the 50% mark. Similar is the case for all floating point workloads. In the case of *applu*, *trop* expends almost 60% more EDP with respect to that of *simple* mode. *rop* has lower EDP with savings around $20-25\%$ compared to *simple*. In one instance, namely for the integer workload *bzip*2, *rop* performs in par with $dvfs$. Interestingly, DVARFS modes



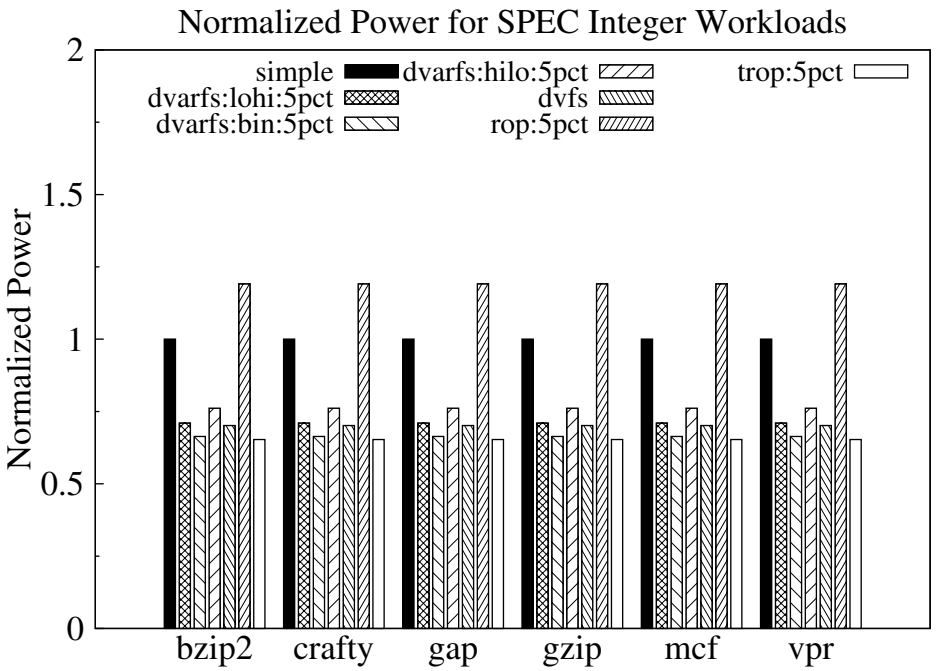


Figure 3.6 Power chart for SPEC INT and FP workloads

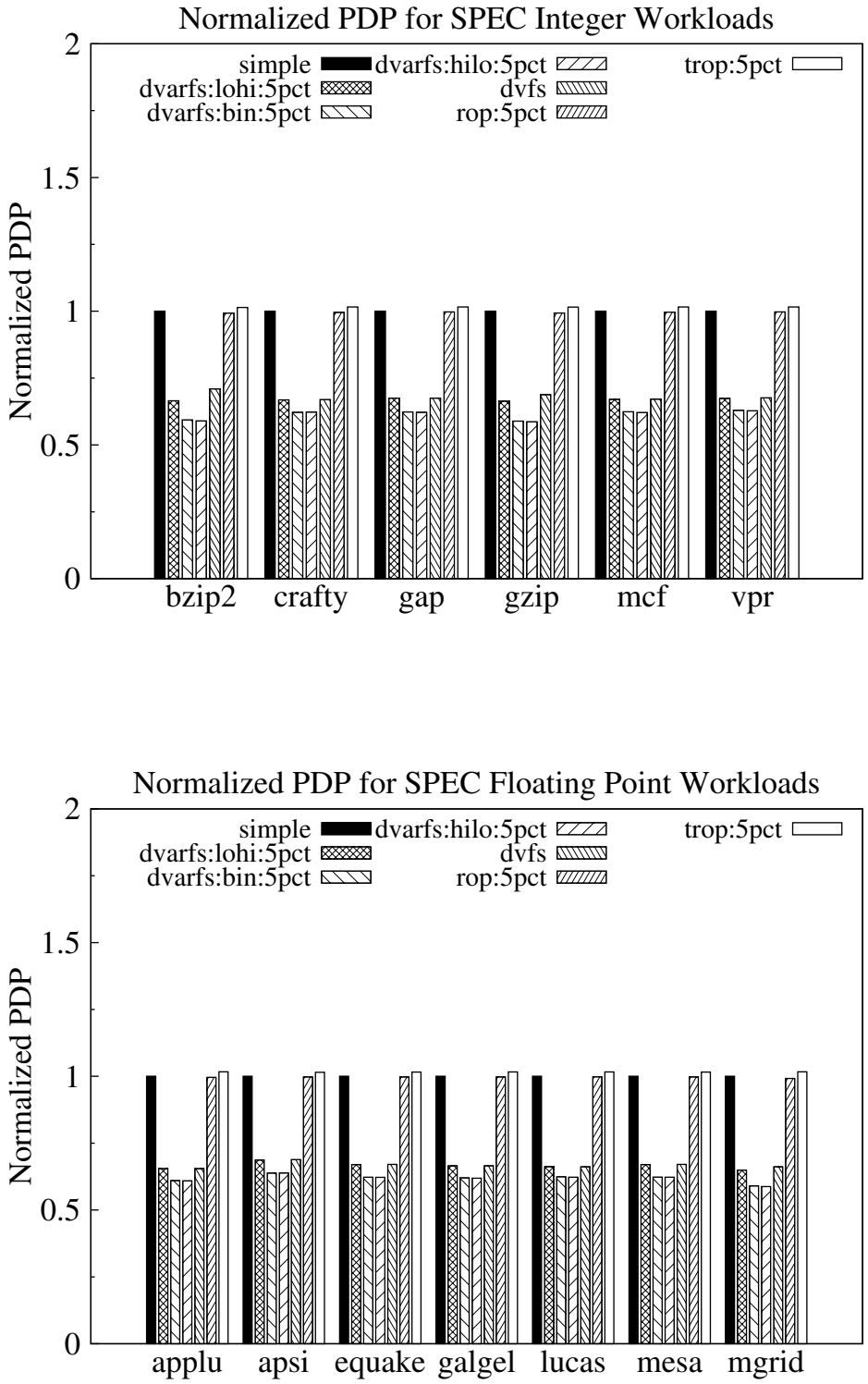


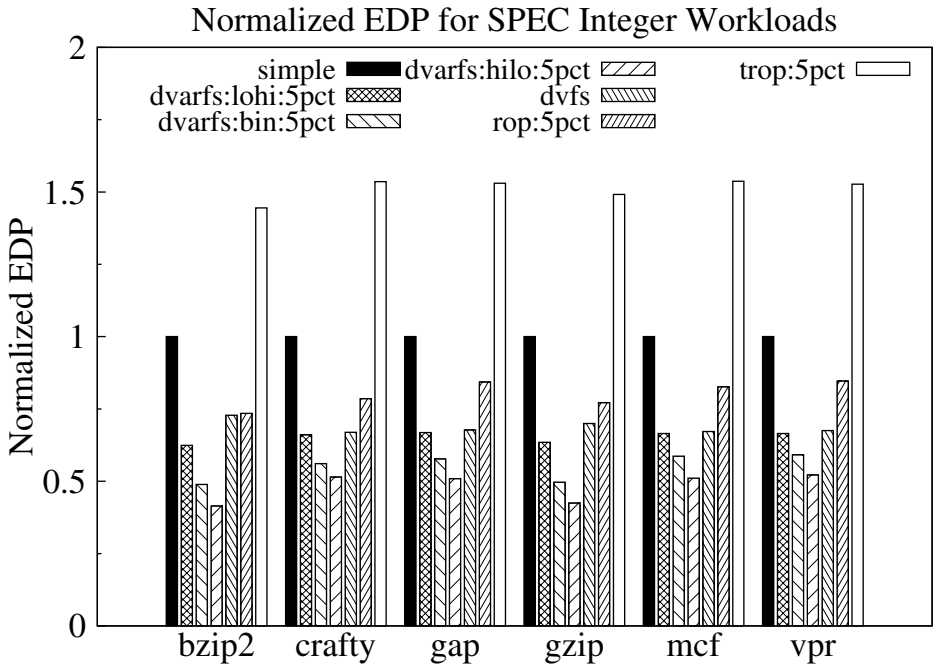
Figure 3.7 PDP chart for SPEC INT and FP workloads

have EDP savings better than DVFS. *lohi* performs better in the case of *bzip*2. For all the other workloads, *lohi* saves as much as $dvfs$ does. *bin* and *hilo* saves more than $dvfs$ in all the cases, with *hilo* outperforming *bin*. *hilo* consistently saves around 50% EDP savings for all workloads and over 60% is reported for the integer workloads *bzip*2 and *gzip*.

3.3.6 ED^2

ED^2 is a metric that has recently gained popularity, especially among handheld and battery operated devices. It is the product (trade off) of EDP and delay. The significance of this metric is that it allows a voltage independent analysis. It has been shown in literature that this is a better metric than the energy delay product, in a sense that optimal ED^2 implies optimal energy and delay [67]. Figure 3.9 illustrates the ED^2 for all workloads. As the order of the delay product increases, the percentage expended by TROP worsens. For ED2 metric, *trop* consumes over 100% (in some cases, close to 150%) in ED^2. From this we can infer TROP is not a good solution for energy constrained systems. Surprisingly, *rop* performs close to $dvfs$. In few instances, as in integer workloads, *bzip*2 and *gzip*, and, floating point workload, *mgrid*, *rop* outperforms $dvfs$ in terms of ED^2. Once again, DVARFS performs the best compared to other modes. *lohi* saves $40 - 50\%$ ED^2 compared to *simple* mode. *hilo* on the other hand reports $60 - 75\%$ ED^2 savings. *bin* performs in between *lohi* and *hilo*.

It is very interesting to note that DVARFS handles performance, power and energy based metrics exceptionally well proving it to be a powerful technique to adopt in the future systems. The most important thing to be noted here is that the trade off is well handled between different modes of DVARFS suitable to the required metric we are interested in. In simple terms, DVARFS possesses the advantages of DVFS and ROP and it is least affected by their limitations, which helps it to suit for all kinds of systems from handheld devices to high performance processors.



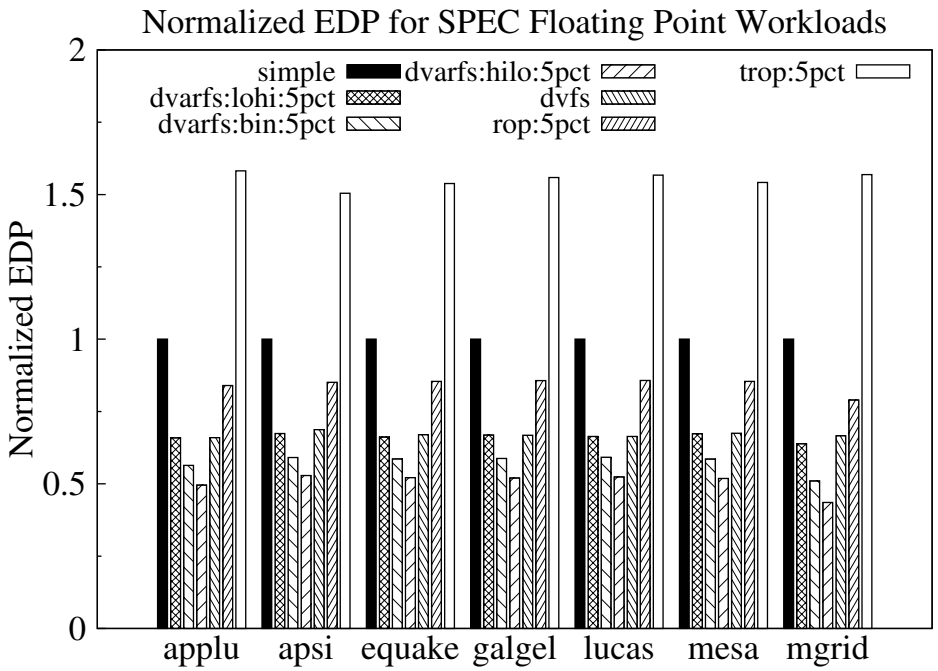
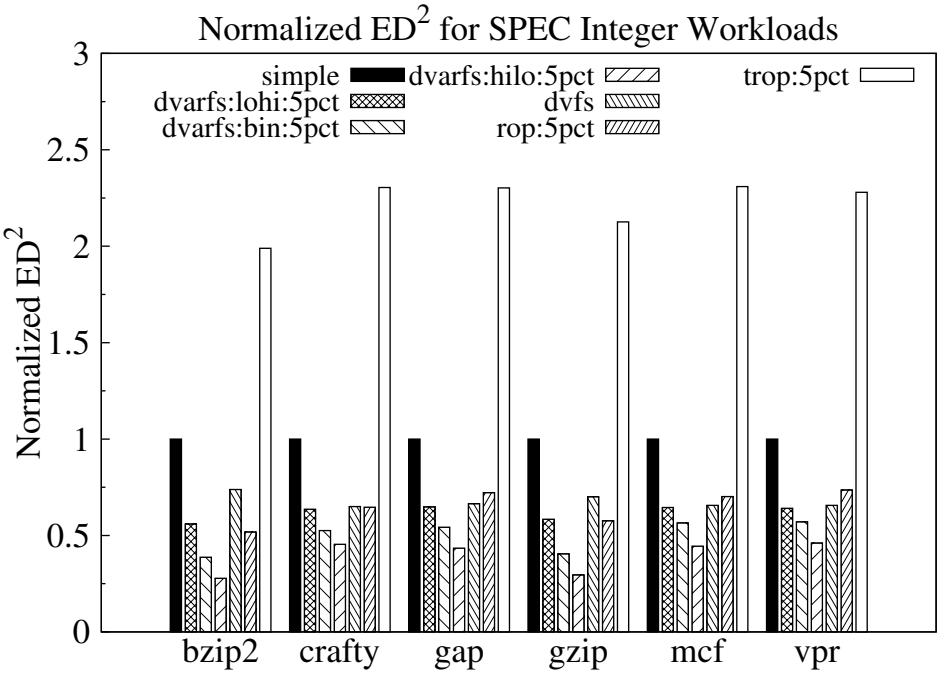


Figure 3.8 EDP chart for SPEC INT and FP workloads



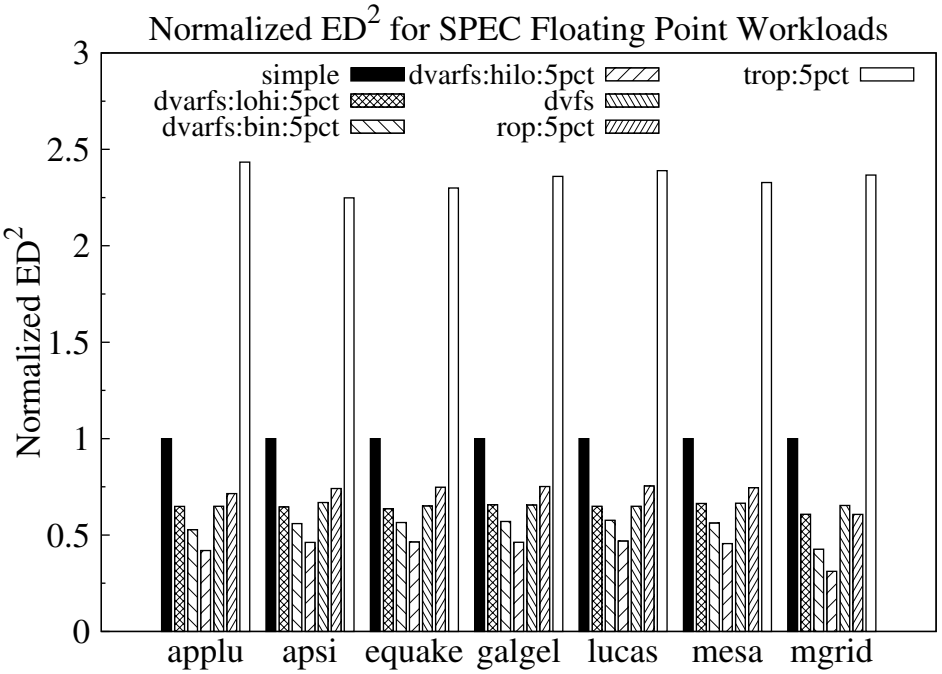


Figure 3.9 ED^2 chart for SPEC INT and FP workloads

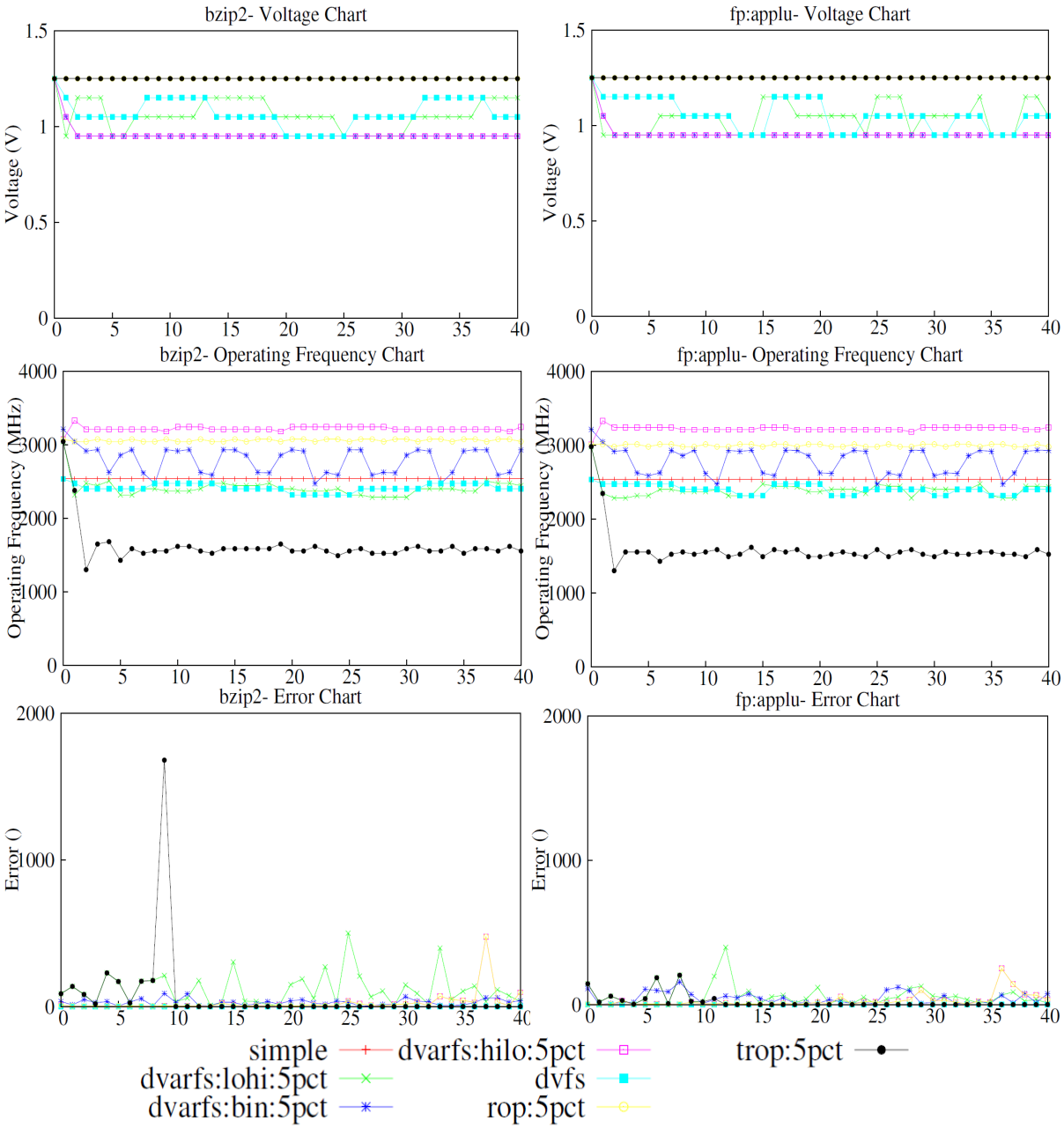


Figure 3.10 Voltage, frequency and error trace for SPEC INT (bzip2) and FP (applu) workloads

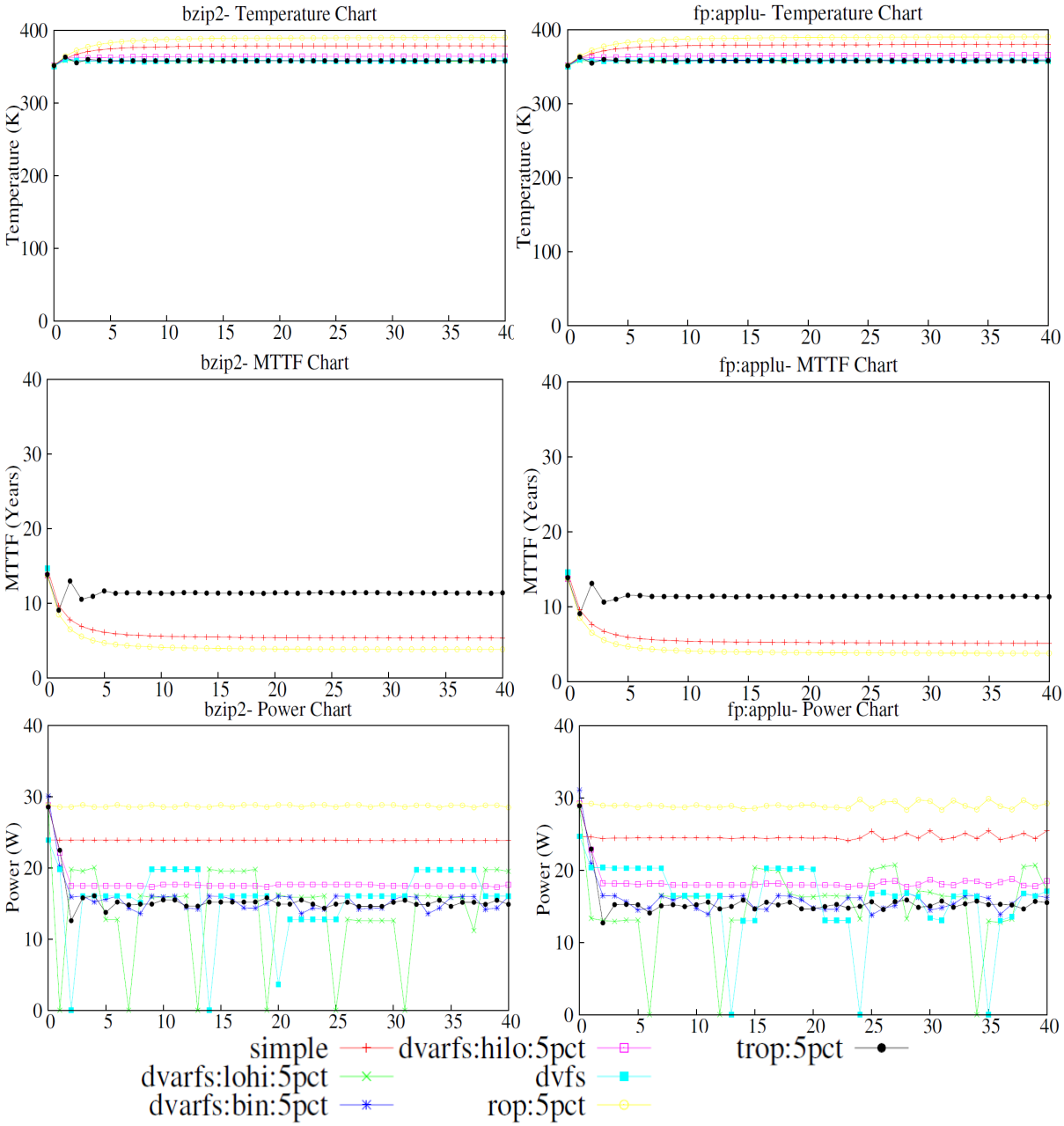


Figure 3.11 Temperature, MTTF and power trace for SPEC INT (bzip2) and FP (applu) workloads

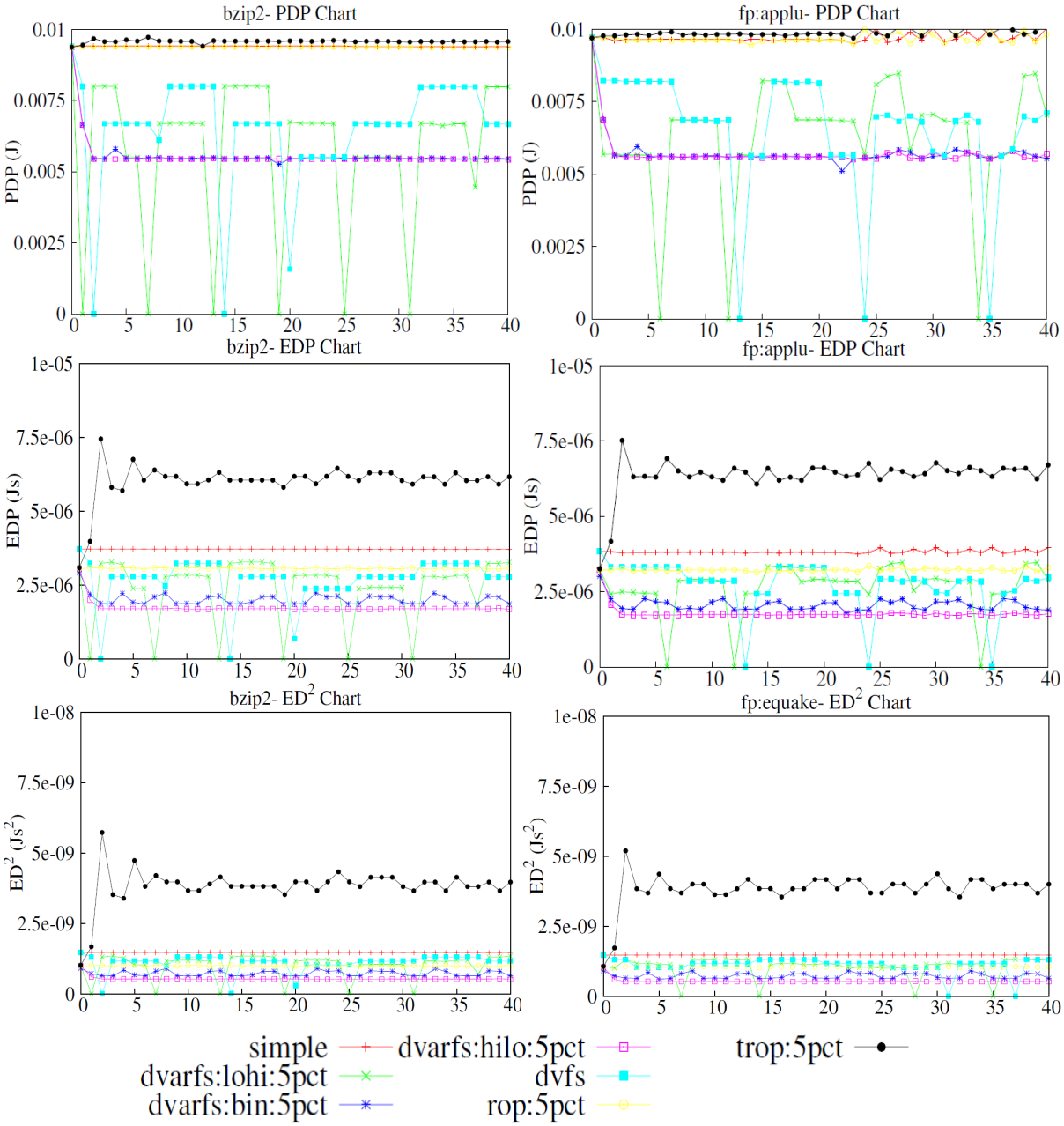


Figure 3.12 PDP, EDP and ED^2 trace for SPEC INT (bzip2) and FP (applu) workloads

3.3.7 Comparative Study of DVARFS with Base Execution

Tables 3.2 and 3.3 show the mean minimum and maximum execution for all SPEC integer and floating point workloads, respectively. In order to eliminate the bias we excluded the initial warm up phase of execution and obvious outliers in the data set. We compare the values of different metrics for each mode with that of *simple* and estimate the percentage increase or decrease. The difference is represented as positive and negative signs. Gain or loss is determined by the sign of the difference and metric itself. For instance, for a given mode, a positive difference in frequency metric is considered as a gain as it supports performance improvement, while that is considered as a loss if it had been a power related metric. We will explain the significant data points in each metric for all the modes.

We observed that for integer workloads, voltage selection range is higher in DVARFS in general than floating point workloads. Whereas, $dvfs$ has the same switching range. *lohi* and $dvfs$ have similar profile with $8-24\%$ lower voltage operating point. *bin* and *hilo* work closely with respect to voltage switching. In the case of FP workloads, both *bin* and *hilo* spend majority of time at the lowest voltage level. Evidently, *rop* and *trop* work at the same voltage level as *simple*.

There is not any difference in the operating frequency of *lohi* in the integer workloads. It gets worse in the case of FP. This explains the trivial performance contribution and overhead suffered by *lohi* in the overall speed up. *bin* shifts between $+12\%$ to -10% in the case of integer workloads. *bin* performs better for FP workloads. The gap between the maximum and minimum is smaller in the case of *hilo*. It should also be noted that *hilo* provides significant gain.

The 5% error margin is maintained throughout execution. Typically, the timing speculation modes suffer $1-5\%$ errors at the maximum. As already mentioned, temperature of *simple* is around $380K$ and *rop* reaches the maximum of $395K$, while all the remaining modes are able to control the temperature within the set limit of $353K$. As a result, there is a 23% loss of device lifetime in the case of *rop*, and up to 70% gain is possible through any of the temperature control technique.

The minimum power savings for $dvarfs$ is 16% as it is observed in the FP workloads and a maximum of 50% is observed in the integer workloads. Significant savings is observed for all the energy related metrics in the case of $dvarfs$ modes. *trop* suffers from $100 - 300\%$ loss for the energy based metrics.

Table 3.2 Comparing various performance metrics for non-overclocked, reliably overclocked processors and DVFS with DVARFS executing SPEC2000 integer benchmarks

Metric	simple		lohi		bin		hilo	
	Max, Min	%Diff	Max, Min	% Diff	Max, Min	% Diff	Max, Min	% Diff
Voltage (V)	1.25	-	1.15	-8	-1.05	-16	-1.05	-16
	1.25	-	0.95	-24	-0.95	-24	-0.95	-24
Frequency (GHz)	2.50	-	2.50	-	2.80	12	3.25	30
	2.50	-	2.50	-	2.25	-10	3.10	24
Errors	-	-	≤ 500	5	≤ 100	1	≤ 150	1.5
	-	-	≤ 50	0.5	≤ 50	0.5	≤ 50	0.5
Temperature (K)	380	-	358	-6	358	-6	360	-5
	353	-	353	-	353	-	353	-
MTTF (Years)	6.5	-	11.0	70	11.0	70	11.0	70
Power (W)	25.0	-	20.0	-20	15.0	-40	18.5	-26
	25.0	-	12.5	-50	13.0	-48	18.4	-26
PDP ($\times 10^{-3}$J)	9.0	-	8.0	-11	5.5	-39	5.5	-39
	9.0	-	5.5	-39	5.5	-39	5.5	-39
EDP ($\times 10^{-6}$Js)	2.75	-	2.68	-3	2.50	-10	2.38	-13
	2.75	-	2.50	-10	2.40	-12	2.38	-13
ED^2 ($\times 10^{-9}$ Js^2)	1.25	-	1.20	-4	1.15	-8	1.00	-20
	1.25	-	1.10	-12	1.12	-10	1.00	-20

Metric			dvfs		rop		trop	
			Max, Min	% Diff	Max, Min	% Diff	Max, Min	% Diff
Voltage (V)	-	-	1.15	-8	1.25	-	1.25	-
	-	-	0.95	-24	1.25	-	1.25	-
Frequency (GHz)	-	-	2.40	-4	3.10	24	2.25	-10
	-	-	2.25	-10	3.00	20	1.18	-52
Errors	-	-	-		≤ 150	1.5	≤ 250	2.5
	-	-	-		≤ 50	0.5	≤ 50	0.5
Temperature (K)	-	-	358	-6	395	4	358	-6
	-	-	353	-	353	-	353	-
MTTF (Years)	-	-	11.0	70	5.0	-23	11.0	70
	-	-						
Power (W)	-	-	20.0	-20	30.0	20	18.0	-28
	-	-	12.5	-50	29.0	16	12.0	-52
PDP ($\times 10^{-3}$J)	-	-	8.0	-11	9.0	-	10.0	11
	-	-	5.5	-39	9.0	-	9.5	5
EDP ($\times 10^{-6}$Js)	-	-	2.70	-2	2.65	-4	6.50	136
	-	-	2.50	-9	2.65	-4	5.00	81
ED^2 ($\times 10^{-9}$ Js^2)	-	-	1.24	-8	1.20	-4	5.00	300
	-	-	1.20	-4	1.20	-4	2.50	100

Table 3.3 Comparing various performance metrics for non-overclocked, reliably overclocked processors and DVFS with DVARFS executing SPEC2000 floating point benchmarks

Metric	simple		lohi		bin		hilo	
	Max, Min	% Diff	Max, Min	% Diff	Max, Min	% Diff	Max, Min	% Diff
Voltage (V)	1.25	-	1.15	-8	0.95	-24	0.95	-24
	1.25	-	0.95	-24	0.95	-24	0.95	-24
Frequency (GHz)	2.50	-	2.49	-1	3.00	20	3.30	32
	2.50	-	2.30	-8	2.49	-1	3.20	28
Errors	-	-	≤ 250	2.5	≤ 120	1.2	≤ 130	1.3
	-	-	≤ 50	0.5	≤ 50	0.5	≤ 50	0.5
Temperature (K)	380	-	358	-6	358	-6	360	-5
	353	-	353	-	353	-	353	-
MTTF (Years)	6.5	-	11.0	70	11.0	70	11.0	70
Power (W)	25.0	-	21.0	-16	15.0	-40	18.0	-28
	25.0	-	13.0	-48	13.0	-48	17.0	-32
PDP ($\times 10^{-3}$J)	9.0	-	8.0	-11	5.5	-38	5.5	-38
	8.8	-	5.5	-37	5.0	-43	5.5	-37
EDP ($\times 10^{-6}$Js)	2.75	-	2.78	1	2.50	-10	2.38	-13
	2.75	-	2.50	-10	2.40	-13	2.38	-13
ED^2 ($\times 10^{-9}$ Js^2)	1.25	-	1.20	-4	1.15	-8	1.05	-16
	1.25	-	1.10	-12	1.13	-10	1.00	-20

Metric			dvfs		rop		trop	
			Max, Min	% Diff	Max, Min	% Diff	Max, Min	% Diff
Voltage (V)	-	-	1.15	-8	1.25	-	1.25	-
	-	-	0.95	-24	1.25	-	1.25	-
Frequency (GHz)	-	-	2.49	-1	3.10	24	1.50	-40
	-	-	2.30	-	3.00	20	1.35	-46
Errors	-	-	-		≤ 130	1.3	≤ 125	1.25
	-	-	-		≤ 50	0.5	≤ 50	0.5
Temperature (K)	-	-	358	-6	395	4	358	-6
	-	-	353	-	353	-	353	-
MTTF (Years)	-	-	11.0	70	5.0	-23	11.0	70
	-	-						
Power (W)	-	-	20.0	-20	29.0	16	16.0	-36
	-	-	13.0	-48	28.0	12	13.0	-48
PDP ($\times 10^{-3}$J)	-	-	8.0	11	9.0	-	9.0	-
	-	-	5.5	-37	8.8	-	8.8	-
EDP ($\times 10^{-6}$Js)	-	-	2.70	-2	2.75	-	7.40	169
	-	-	2.50	-9	2.75	-	6.20	125
ED^2 ($\times 10^{-9}$ Js^2)	-	-	1.24	-1	1.20	-4	5.10	308
	-	-	1.20	-4	1.20	-4	3.75	200

Execution Traces

In the forthcoming parts of the chapter, we show how execution traces for various metrics during run time for selected workloads. Figures 3.10, 3.11 and 3.12 depict the traces cycle by cycle for half for the first half a million cycles. For explanation purpose, we illustrate only two benchmarks here. We selected one each from SPEC INT and FP, namely, *bzip*2 and *applu*, respectively. Execution trace for all the workloads are provided in Appendix A.

3.3.8 Voltage Trace

Figure 3.10 illustrates the voltage trace two instances of benchmarks. It is evident to note that *simple*, *rop* and *trop* stay at $1.25V$. All modes that have voltage control enabled switches to $1.05V$ almost as as soon as execution starts, and never goes back to $1.25V$ (the highest voltage level). $dvfs$ and *lohi* switches between the remaining three levels, while *bin* and *hilo* switches to the lowest voltage level ($0.95V$) and stays there till the end of execution.

3.3.9 Clock Frequency Trace

The second part of Figure 3.10 illustrates the frequency switching profile during execution. Again, *simple* stays at $2500MHz$, which is the base frequency. $dvfs$ fluctuates between only two levels. There are only as many frequency levels as voltage levels in the case of DVFS. Similar trend observed in the case of *lohi*. *bin* fluctuates back and forth. This is mainly due to error rate and not thermal emergencies. *hilo* stays at the maximum frequency level under tolerable error rate. The point to be noted here is that the overclocked frequency at the lowest voltage level is still over the operating frequency of *simple*. Following is inferred from this:

1. Performance enhancement is achievable even at the lowest voltage

2. Timing error occurrence is tolerated even at highest frequency levels

rop performs similar to *hilo*. In spite of overclocking capability, *trop* has to operate at lower frequencies so as to handle thermal emergencies. In due course of run time, it slowly converges to a point where the optimal frequency for both temperature and performance is reached.

3.3.10 Error Trace

Final portion of Figure 3.10 shows the error profile during execution. There are no errors for *simple* and $dvfs$ modes as they work at conservative frequency limits. Small spikes are observed at constant intervals for *rop*, *trop* and all DVARFS modes. It is more profound in *lohi* because of the frequent voltage switching. Every time voltage switches, the control loop starts the frequency search afresh from the lowest level. In the case of *bin* and *lohi* there is not much voltage switching during most part of execution. *trop* has a high spike for *bzip*2. Apart from few spikes, the error occurrence is well controlled in *trop*.

3.3.11 Temperature trace

Temperature traces for all modes across all workloads are presented in Figure 3.11. Temperature gradually climbs over $380K$ (*simple*) when left uncontrolled. It should be pointed here that the break down temperature limit for digital circuit is close to $378K$. We assumed $363K$ to be the critical temperature limit. *rop* exceeds $390K$. DVARFS performs in par with DVFS by effectively controlling temperature within critical limits. Interestingly, *trop* maintains the temperature as effectively as DVARFS and DVFS. This is an important point to be considered, as TROP proves to be effective way of controlling temperature of the processor in the absence of dynamic voltage scaling.

3.3.12 MTTF Trace

We tracked down the Mean time To Failure (MTTF) during execution using the models described in Section 2.3.2 as shown in Figure 3.11. All the modes that control temperature effectively reports MTTF over 10 years. *simple* ends up with MTTF close to 5 years, whereas *rop* is around 3 years and has the least MTTF out of all the modes.

3.3.13 Power Trace

Power trace during workload execution is shown in Figure 3.11, third row. Quite intuitively, *rop* dissipates the maximum power, close to $30W$. It is to be noted that all the DVARFS

configurations start with the same power state as *rop* since the initial voltage and frequency settings are the same. Also, *simple* and $dvfs$ start from the same level. We observed that $dvarfs$ soon adapts itself to the temperature requirements and performs in par with $dvfs$. We see that there is not much power fluctuations in *hilo*, as opposed to *lohi* or *bin*. *lohi* fluctuates the maximum. This is because, every time there is a power level switching, all the levels *lohi* had gained is lost. Power level switching in $dvfs$ and *trop* at regular intervals is also constantly observed.

3.3.14 Energy Metric Traces

Figure 3.12 illustrates the profile traces of PDP, EDP and ED^2, respectively, for the selected integer and floating point workloads. $dvfs$ and *lohi* shuttles between the high and low states for all the three metrics. However, the gap between the levels is relatively reduced as it goes from PDP, EDP to ED^2. *trop* is affected the most by the delay product. *hilo* and *bin* have more regular and stable profile throughout execution. Such is the case for *simple* and *rop* as well, however, the magnitudes are way too higher.

3.4 Summary

One of the main hurdles in realizing timing speculation in practical circuits is their barrier they pose towards harnessing power dissipation in nanoscale circuits. Higher power density escalates chip temperature, which is a serious threat. In this chapter, we presented an overview of power impact on chip temperatures and analyzed its effect on lifetime reliability. We considered a typical reliable overclocking framework and studied its thermal behavior compared to worst case design. We made the case for the need of a powerful thermal management scheme in reliably overclocked circuits.

We presented an efficient technique for enhancing performance and managing on-chip temperature by allowing dynamic voltage-frequency pairing. We built a feedback control system called DVARFS, exploring a new direction to manage on-chip thermal conditions to achieve maximal performance benefits. The DVARFS mechanism is an aggressive yet reliable frame-

work for energy efficient thermal control. DVARFS facilitates to reliably overclock the processor under thermal bounds at target lifetime with a programmable error rate. We established an extensive simulation framework environment, integrating various tools to perform our simulation studies. Using this framework we have shown that it is possible to achieve power savings in par with existing DVFS scheme despite exceeding the worst-case operating frequency. The significance of this approach is that the system operates under controlled power, under a given temperature set point and still yield performance enhancement. With this aggressive approach, we were able to achieve up to 40% speed-up compared to a base scheme with no overclocking. We also compared other metrics against different schemes and found that DVARFS invariably performs better. We observed 75% ED^2 savings relative to base architecture. In comparison, DVFS only saves only about 40%. From our investigation, it becomes evident that controlled reliable overclocking is indeed a beneficial way to enhance performance taking into consideration about thermal constraints.

CHAPTER 4. MANAGING CONTAMINATION DELAY

Timing Speculation (TS) is a widely known method for realizing better-than-worst-case systems. Aggressive clocking systems have TS as their central theme and operate at a clock frequency range beyond specified safe limits, exploiting the data dependence on circuit critical paths. However, the margin for performance enhancement is restricted due to extreme difference between short paths and critical paths. In this chapter, we show that increasing the lengths of short paths of the circuit increases the margin of TS, leading to performance improvement in aggressively designed systems. We develop an algorithm to efficiently add delay buffers to selected short paths while keeping down the area penalty. We explore the possibility of increasing short path delays further by relaxing the constraint on propagation delay, and achieve even higher performance.

4.1 Background

Microprocessors have traditionally been designed to function reliably for the worst case timing delays under adverse operating conditions. Such worst case scenarios occur rarely, allowing possible performance improvement by making common cases faster. Alternative to conventional methods, the concept of latching data speculatively is called Timing Speculation (TS) [68, 65, 6, 69, 4, 3]. Dual latch based TS is a widely accepted methodology for designing better-than-worst-case digital circuits. Timing speculation combined with timing error tolerance is a powerful technique to (1) achieve energy efficiency by under-volting, as in Razor [6], or (2) performance enhancement by overclocking, as in SPRIT^{3}E [65]. They are less expensive in terms of area and power compared to logic replication.

Dual latch based TS require additional clock routing for replicated latches (or flip-flops)

that are triggered by a phase shifted clock. Despite the area and routing overheads, the benefits achieved by dual latched TS remain immense [6, 32, 3, 70, 71, 72, 73]. However, in [3] it has been pointed out that the timing benefits realized through speculation is limited by the short paths of the circuit due to the tight timing constraints in order to guarantee correctness during error recovery. This problem is compounded when circuits have a significantly lower contamination delay. It has been shown that for a CLA adder circuit significant performance enhancement is achieved when its contamination delay is increased by adding buffers, increasing the delay of all the paths in the circuit above a desired lower bound, while not affecting the critical path of the circuit is one of the steps performed during synthesis of sequential circuits to fix hold time violations. However, increasing the contamination delay of a logic circuit significantly, sometimes as high as half the propagation delay, without affecting its propagation delay is not a trivial issue [74]. At first glance, it might appear that adding delay by inserting buffers to the shortest paths will solve the problem. However, delay of a circuit is strongly input value dependent, and the structure of the circuit plays a role in deciding the value of an output in a particular cycle. Current synthesis tools support increasing the delay of short paths through their hold violation fixing option; in a broader sense, what we essentially want to do is that to extend the hold time of the replicated register.

Traditional delay optimization approaches consider only part of the problem, viz., to ensure that the delay of each path is less than a fixed upper bound. The closest work we are aware of is presented in [75], which uses timing optimization algorithm, Sylon-Dream Level-Reduction, for speeding up multi-level networks. The non-critical paths are processed by an area reduction procedure to reduce network area without increasing the maximum depth. SDLR uses the concept of permissible functions in both level and area reduction processes. The existing techniques only attempt to confine the critical path delay under design specified threshold. For aggressive timing speculative architectures, in addition to the existing short path timing constrains free of any hold time violations, the delay optimization algorithms must make sure that the short paths must satisfy threshold requirements in order to increase the performance enhancement margin. This is the aspect that makes our work different from any of the existing

works. Our work is aimed at increasing the contamination delay of digital circuits up to a given threshold, beyond satisfying hold time violations.

We address three significant issues pertaining to short paths in timing speculation architecture. First, we investigate theoretical analysis of a dual latch TS framework and quantize the margin for performance enhancement by operating beyond worst case. Second, we study the impact of short paths on performance on Alpha processor core, where we present a sensitivity analysis of the speed-up achievable for different settings of contamination delays. In that process, we establish a case for increasing contamination delay of circuits for aggressive architectures to relax the margin for performance enhancement. Third, we develop an algorithm to achieve this goal. Specifically, we build a weighted graph model to represent a multi-level digital circuit. We showcase a new min-arc algorithm that works on the graph network to increase short path delays by adding buffers to selective interconnections. We consider each interconnection, whether it lies on the critical path, short path, or not. Depending upon how far each section of the circuit is from the maximum and minimum delayed paths, fixed delays are added. The algorithm is evaluated on ISCAS'85 benchmark suite. In our simulations, we investigate the increase in short path delays with and without relaxing critical path delays of these circuits. Also, we analyze the area overhead due to the addition of delay buffers. We were able to increase the contamination delay to 30% of the circuit critical path length without affecting its propagation delay. We further increase the contamination delay by relaxing the constraint on the propagation delay by allowing it to be increased by a small amount for a larger gain in performance.

4.2 Existing Works for Managing Circuit Path Delay

Early works on timing verification involved identification and categorization of long paths as either false paths or sensitizing paths [76]. Long paths that are false paths (paths with no activity) unnecessarily increase the circuit critical delay. Therefore, detecting false paths and mitigating them is a critical issue in digital circuits even to this day [77, 78, 79].

As already mentioned in Section 4.1, not many works are done keeping short paths in mind.

Sylon-Dream accomplishes faster multi-level networks by its level reduction technique (SDLR) [75]. The non-critical paths are processed by an area reduction procedure to reduce network area without increasing the maximum depth. SDLR uses the concept of permissible functions in both level and area reduction procedures. Gate resizing and buffer insertion are two major techniques for critical path optimization. Critical path selection instead of sensitization is suggested for performance optimization [80]. Here the objective is to select a small set of paths to ease the optimization process by guaranteeing the delay of the circuit to be no longer than a given threshold. Several optimization techniques, involving clustering, logic analysis and gate resizing are proposed in [81, 82, 83, 84, 85]. A statistical timing analysis approach is investigated in [86]. A re-timing and re-synthesis approach is presented in [87]. This work suggests re-synthesizing the circuit to expose signal dependencies. The optimization scheme tightly constrains logic re-synthesis, so that the re-synthesized circuit is guaranteed to meet the performance target.

Although there are several delay optimization approaches proposed in literature, all of them try to hold the critical path delay within a threshold. It is fundamental that all the timing optimization algorithms must consider short path timing constraints. Data latches in a pipelined architecture inherently possess set up and hold time constraints. It is necessary to make sure that the resulting circuit has no set-up or hold time violations, to guarantee correct data transfers. There are algorithms to make sure the circuit is free of any such violations considering both long and short paths [88]. However, there is hardly any consideration for short path constraints from the perspective we are dealing with. In this work, we try to alleviate the contamination delay limitation imposed on aggressive timing speculation architectures. Therefore, we differ from any of the existing works fundamentally. As far as we know, this is the first work aimed at increasing the contamination delay of digital circuits up to a given threshold. It is also important to point out that our algorithm works complementary to existing synthesis schemes.

4.3 Timing Speculation Circuits

Traditional design methodologies for the worst-case operating conditions are too conservative as the critical timing delays rarely occur in tandem, during typical circuit operation. Such infrequent occurrence of critical timing delays has opened a new domain of study that allows improvement of processor performance to a greater extent. During execution, since delay incurred by the digital circuit is much less than the worst-case delay, this can be exploited by making common cases faster. Timing speculation is a technique wherein data generated at aggressive speeds are latched and sent forward speculatively assuming error free operation. Error detection is deployed to detect a timing violation. When an error is detected, the forwarded data is voided and the computation is performed again as part of the recovery action.

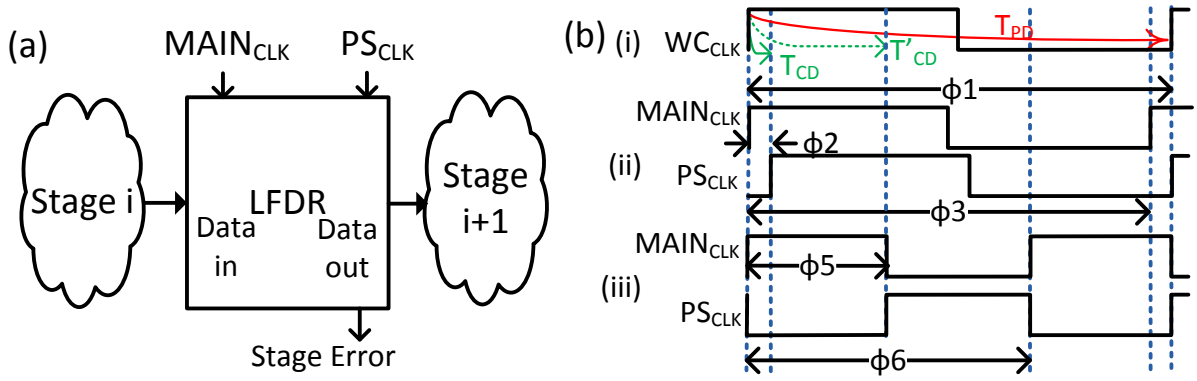


Figure 4.1 (a) Typical pipeline stage in a reliably overclocked processor (b) Illustration of aggressive MAIN and PS clocks for circuits with different contamination delays

4.3.1 Dual Latched Timing Speculation Framework

Let us recall the dual latched timing speculation framework (LFDR) from Chapter 2. Figure 4.1 (a) presents a black box view of the LFDR circuit in between two pipeline stages. As it was mentioned previously, to be able to reliably overclock a system dynamically using LFDR framework, the foremost requirement is to generate the $MAIN_{CLK}$ and PS_{CLK}. The

two clocks are governed by certain timing requirements that are to be met at all times. LFDR consists of two registers: a *main register* clocked ambitiously by $MAIN_{CLK}$ at a frequency higher than that required for error-free operation and a *backup register* clocked by PS_{CLK} at the same rate as $MAIN_{CLK}$, but phase shifted such that the worst-case propagation delay time of the combinational circuit. The timing diagram shown in Figure 4.1 (b) illustrates this. Here, case (i) shows the worst case clock, WC_{CLK}, with time period Φ_1, which covers the maximum propagation delay. Case (ii) shows TS scenario, where the clock time period is reduced to Φ_3. The key point to note is that the amount of phase shift, Φ_2, for the PS_{CLK} is limited by the contamination delay, T_{CD}, of the circuit.

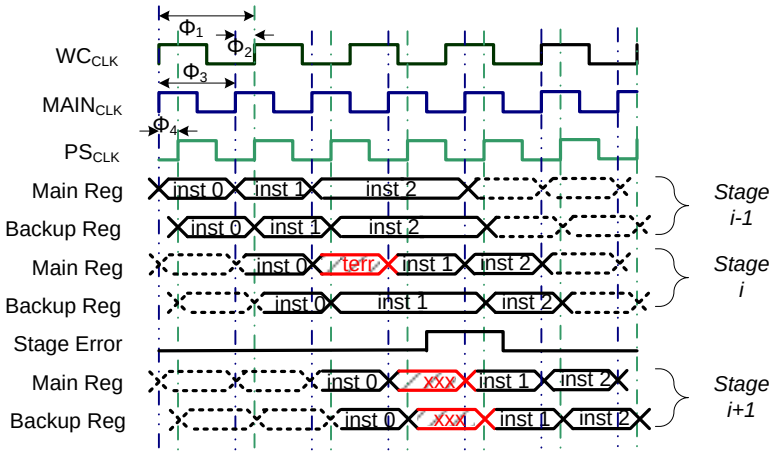


Figure 4.2 Timing diagram showing pipeline stage level timing speculation

Figure 4.2 shows timing waveforms that depict timing speculation using LFDR. In the figure, *inst*0 moves forward without any timing errors. However, *inst*1 encounters a timing error in *Stage* i, indicated by corrupted data "*terr*". This error is detected by the error detection mechanism, and the stage error signal is asserted. This stage error signal triggers a local and global recovery. Timing error recovery flushes the data sent forward speculatively, indicated in the figure as "*xxx*", and voids the computation performed by *Stage* $i+1$. Once the timing error is fixed, the pipeline execution continues normally. It is clear from the waveform that the time gained by TS is Φ_4, which is equal to Φ_2.

A balance must be maintained between the number of cycles lost to error recovery and the gains of overclocking. One important factor that needs to be addressed while phase shifting the PS_{CLK} is to limit the amount of phase shift within the fastest delay path of the circuit.

4.4 Impact of Short Paths on Performance

The cardinal factor that limits frequency scaling for LFDR frameworks is the contamination delay of the circuit. The phase shift of the delayed clock is restricted by the contamination delay to prevent incorrect result from being latched in the backup register. Reliable execution can be guaranteed only if the contents of the redundant register are considered *"golden"*. To overcome this limitation, it is important to increase the contamination delay of the circuit. From Figure 2.3 (b) case (iii) it is easy to notice that a circuit with contamination delay $T'_{CD} > T_{CD}$ gives a greater margin for TS.

Let us denote the worst-case propagation delay and minimum contamination delay of the circuit as T_{PD} and T_{CD}, respectively. Let T_{WCCLK}, $T_{MAINCLK}$ and T_{PSCLK} represent the clock periods of WC_{CLK}, $MAIN_{CLK}$ and PS_{CLK}, respectively. Let T_{PS} represent the amount of phase-shift between $MAIN_{CLK}$ and PS_{CLK}. Also we will denote T_{OV} as the overclocked time period.

At all times, the following equations hold.

$$T_{WCCLK} = T_{PD} = \frac{1}{F_{MIN}} \tag{4.1}$$

$$T_{MAINCLK} = T_{PSCLK} = T_{OV} \tag{4.2}$$

$$T_{PD} = T_{OV} + T_{PS} \tag{4.3}$$

Let F_{MIN} be the setting where there is no overclocking i.e., $T_{OV} = T_{PD}$. In this case, $T_{PS} = 0$. The maximum possible frequency, F_{MAX} permitted by reliable overclocking is governed by T_{CD}. This is because short paths in the circuit, whose delay determine T_{CD}, can corrupt the data latched in the backup register. If the phase shift T_{PS} is greater than the T_{CD},

then the data launched can corrupt the backup register at PS_{CLK} edge. If such a corruption happens, then the backup register may get incorrect result and cannot be considered *"golden"*. Hence, it is not possible to overclock further than F_{MAX}. The following equations should hold at all times to guarantee reliable overclocking.

$$T_{PS} \leq T_{CD} \tag{4.4}$$

$$F_{MAX} \leq \frac{1}{T_{PD} - T_{CD}} \tag{4.5}$$

For any intermediate overclocked frequency, F_{INT}, between F_{MIN} and F_{MAX}, $T_{PS} \leq T_{CD}$. During operation, F_{INT} is determined dynamically based on the number of timing errors being observed during a specific duration of time. The dependence of phase shift on contamination delay leads directly to the limitation of the aggressive frequency scaling. A simplistic notion of the maximum speed-up that is achievable through reliable overclocking is given by Equation 4.6.

$$\text{Maximum Speedup} = \frac{T_{PD}}{T_{PD} - T_{CD}} \tag{4.6}$$

4.4.1 Increasing Short Path Delays

It is clear from Equation 4.6 that the maximum speedup is achieved when the difference between the contamination delay and propagation delay is minimal. However, it must be noted that increasing T_{CD} too much also affects the margin for overclocking. To overcome this challenge, we develop a technique to increase the contamination delay by a moderate extent without affecting the propagation delay of the circuit. As outputs of the combinational logic depends on several inputs, and more than one path to each output exists, with both shorter and longer paths overlapping, adding buffer delays to shorter paths would increase the overall propagation delay of the circuit. The main challenge is to carefully study the delay patterns, and distribute the delay buffers across the interconnections. More importantly, the overall propagation delay must remain unchanged. However, it may not always be possible to constrain propagation delay of the critical paths due to logic/interconnection sharing in the network.

Most practical circuits have significantly lower contamination delay. For instance, we verified that an 8-bit CLA adder circuit, implemented in $0.18\mu m$ Cadence Generic Standard Cell Library (GSCLib), has a propagation delay of $1.06ns$, but an insignificant contamination delay of $0.06ns$, thus allowing almost no performance improvement through reliable overclocking. It should be noted that the outputs of CLA adder depends on more than one inputs, thus a trivial addition of delay buffers to short paths results in increased propagation delay of the circuit. However, by re-distributing the delay buffers all to one side (either input or output), it was possible to increase contamination delay, without affecting the propagation delay, by up to $0.37ns$.

Increasing circuit path delay above a desired level without affecting critical path is not uncommon in sequential circuit synthesis. In fact, it is performed as a mandatory step during synthesis operation. In a sequential circuit, for an input signal to be latched correctly by an active clock edge, it must be loaded (become stable) before a specified time. This duration is called the set up time of the latch. Again, the input signal must be stable until a specified time after the active clock edge in order to get sampled correctly. This interval is called the hold time of the latch. Any signal change in the input before set-up time or after the hold time does not affect the output until the next active clock edge. Clock skew, which is the difference in arrival times at the source and destination flip-flops, also exacerbates hold time requirements in sequential circuits. Hold time violations occur when the previous data at the input of the destination flip-flop is not held long enough to be latched properly. The data can change during the hold time window, if the contamination delay of the circuit is less than the hold time requirements at the destination flip-flop. The hold time requirement for a sequential circuit is normally a very small fraction of the propagation delay of the circuit. Hence, adding buffers to short paths that violate hold time criteria is a step that is done without too much of a concern regarding area and power overheads.

Increasing the contamination delay of a logic circuit significantly, sometimes as high as half the propagation delay, without affecting its propagation delay is not straightforward [74]. At first glance, it might appear that adding delay by inserting buffers to the shortest paths will

Parameter	Value
Fetch/ Decode/ Issue/ Commit width	4 inst/cycle
Functional units	4 INT ALUs, 1 INT MUL/DIV, 4 FP ALUs, 1 FP MUL/DIV
L1 D-cache	128K
L1 I-cache	512K
L2 Unified	1024K
Technology node	$45nm$
Base frequency	$2.5GHz$
No. of freq levels	32
Freq sampling	$10\mu s$
Freq penalty	$0\mu s$ (Assuming Dual PLL)

Table 4.1 Simulator parameters

solve the problem. However, delay of a circuit is strongly input dependent, and several inputs play a role in deciding the value of an output in a particular cycle. Current synthesis tools support increasing the delay of short paths through their hold violation fixing option; in a broader sense, what we essentially want to do is to extend the hold time of the backup register.

Though it is possible to phase shift to a maximal extent, reducing the clock period by that amount may result in higher number of errors. Having a control over the increase in contamination delay gives us an advantage to tune the circuit's frequency to the optimal value depending on the application and the frequency of occurrence of certain input combinations. Also, introducing delay to increase contamination delay increases the area of the circuit. Therefore, while judiciously increasing contamination delay we must also ensure that the increase in area is not exorbitant.

4.4.2 Performance on Alpha Processor

To demonstrate the effect of increasing short paths on performance, we conducted a simple study on Alpha processor model for different contamination delay settings. We ran selected set of SPEC 2000 benchmark workloads on SimpleScalar - a cycle accurate simulator [59]. In order to embed timing aspects in SimpleScalar, we examined a hardware model of Alpha processor and obtained the number of timing errors occurring at different clock period, for each workload.

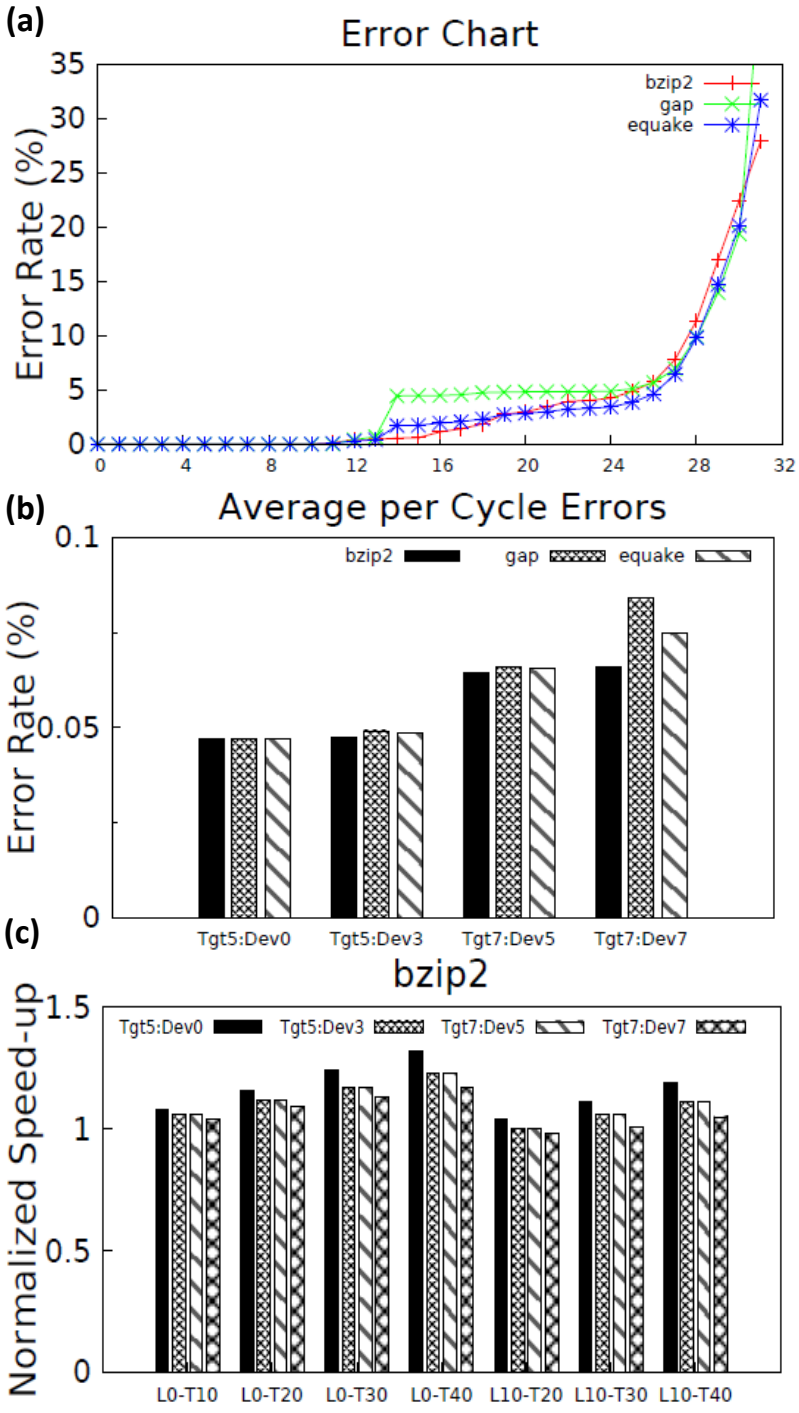


Figure 4.3 (a) Cumulative error rate at different clock periods for the IVM Alpha processor executing instructions from SPEC 2000 benchmarks (b) Average error rate per clock cycle (c) Normalized speed-up relative to reliably overclocked, unmodified circuit

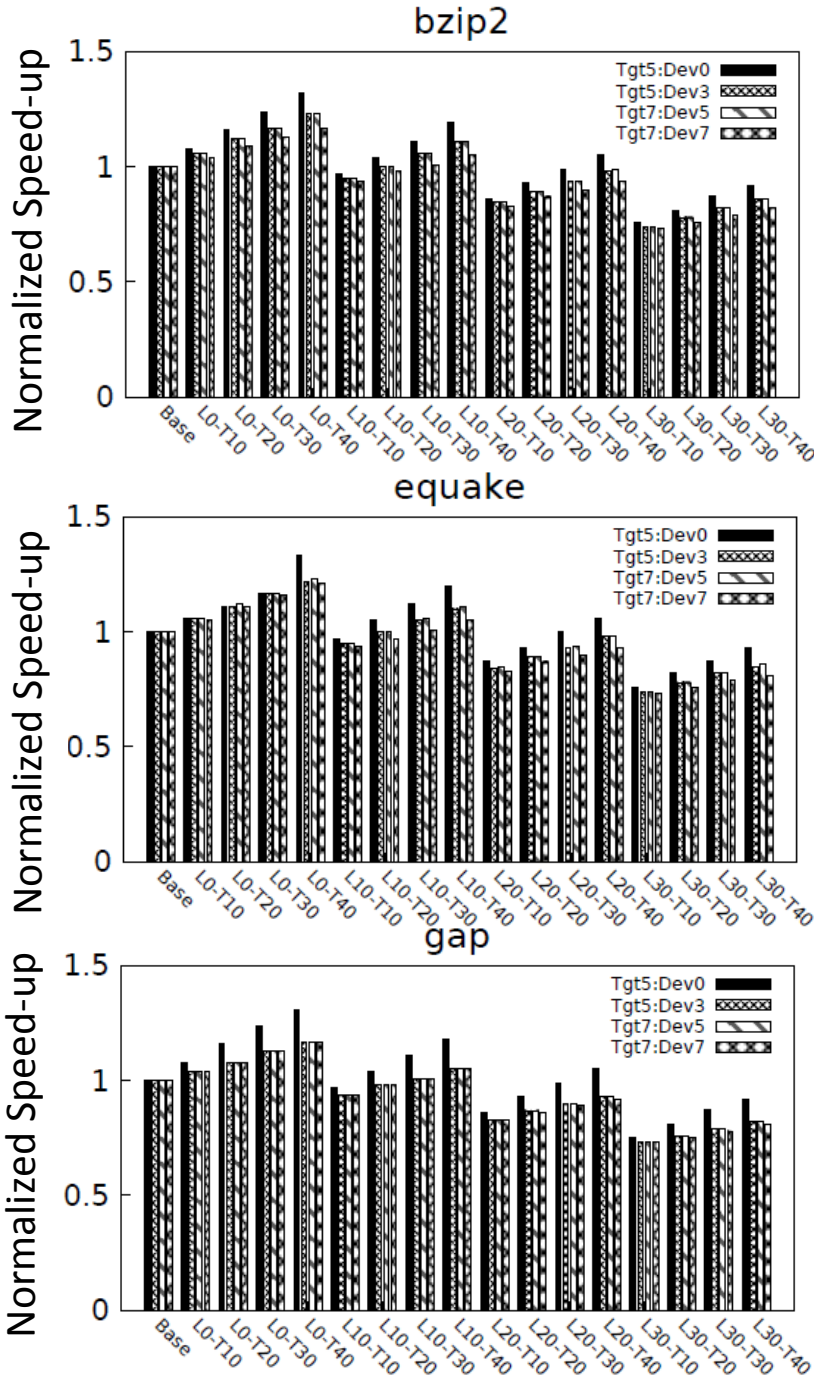


Figure 4.4 Normalized speed-up of bzip2, equake, and gap benchmarks for different L and T configurations

For this purpose we used the IVM synthesized in Chapter 3. Although we are aware of the fact that the pipeline in IVM is simplistic, it does not have any impact on our results as we are performing a comparative study of different settings for the same circuit. We adopt the same configuration for SimpleScalar simulations as well. The details of the settings are presented in Table 4.1. Figure 4.3(a) shows the cumulative error rate of three SPEC 2000 workloads for 32 equal intervals, for worst-case delay of $7ns$ and minimum contamination delay of $3.5ns$. The error profile illustrated is the average values obtained by running the experiment for $100,000$ cycles, and repeating the experiment with different sequences of $100,000$ instructions for each workload.

We incorporated a timing error injector that induces appropriate number of errors in SimpleScalar. Pipeline stall for one cycle per error occurrence is added correspondingly. As increasing the contamination delay affects path distribution of the whole circuit, it is likely that the overall error rate for each workload may go up. In our experiment, we assume uniform increase in error rate, denoted as Dev, for each workload. For our study, we typically used $Dev = 0, 3, 5$ and 7%. Further, we analyze the performance impact of varying CDs with different target error rates (Tgt). Figure 4.3(b) shows the error occurrence per cycle for *bzip*2, *equake* and *gap*. Quite evidently, we observed smaller error occurrences for small/no deviation of circuit, and the error rate tend to increase as the error rate due deviation, Dev, goes up. However, a small increase in target error rate allows more margins for performance increase. But, this may not hold true for higher error rates. In general, it was generally observed that when Dev gets closer to Tgt, there was an increase in error occurrences. This is more noticeable in the case of *gap*.

Since it may not always be possible to increase the contamination delay without affecting the critical paths, we increase the CD to a threshold limit. As a result, we may end up increasing the PD. We also experimented with increase in PD by allowing a leeway of a small percentage of increase in PD. We study the speed-up obtained for different combinations of CD threshold and PD leeway relative to the performance of aggressive clocking framework with the original circuit. $L\langle l\rangle - T\langle t\rangle$ denotes l% leeway of PD and t% minimum threshold of CD.

We performed our study for $l = 0, 10, 20$ and 30% and $t = 10, 20, 30$ and 40%.

We found that in all the cases, performance goes up with threshold values, which is in agreement with our intuition. In other words, increasing the short path delays allows more margin for reliable aggressive clocking assuming a moderate target error rate occurrence. It should also be noted that allowing a leeway on critical paths induces performance overhead. Normalized speed-up trend of *bzip*2 workload for various modes of operation is exemplified in Figure 4.3(c). We have illustrated the results for the modes that yielded performance gains. The performance of *bzip*2, *equake*, and *gap* benchmarks, for all the configurations we implemented is shown in Figure 4.4. From the point of view of leeway on PD, our investigation on relative performance is summarized as follows:

- $L = 0$ is the best case scenario for performance benefits, yielding from $10-30\%$ speed-up.
- $0 \leq L \leq 10$ is the effective range for any performance benefits at all, irrespective of T
- $L = 20\%$ gives a small increase in performance in the range $0\% \leq Dev \leq 7\%$
- $L = 30\%$ gives a little increase in performance for few cases in the range $0\% \leq Dev \leq 5\%$
- $L > 30\%$ causes performance overhead even for higher values of T and smaller Dev

Our experiments reveal that by increasing the delays of short paths up to 40%, subject to moderate increase in PD (typically 10%), yields up to 30% performance enhancement. Also, it is very important to keep the increase in error rate due to circuit deviation within 5%. This guarantees zero overhead even at maximum leeway ($L = 30\%$).

This study establishes a case for change in the existing synthesis algorithms to incorporate minimum path delay constraints. The major change in this revised algorithm is to increase the short path delays without (or minimally affecting) the critical path delays of the circuit. A secondary and passive constraint is to maintain the circuit variation (if not make it better), so that the deviation causing increase in error occurrences is kept minimal. We will discuss more on this constraint later. We provide a systematic approach to realize circuits with path delay distribution that allows greater margin for aggressive clocking for performance enhancement.

4.5 Min-arc Algorithm for Increasing Short Path Delays

After having a close look at various circuits, we understand that increasing short path delays invariably increases the area of the circuit and, if not done carefully, affects its propagation delay. An ideal solution that we would like is to have logic moved from the critical path to the non-critical paths without using the specified components at the output terminals not getting affected. This is not always possible. The next best approach would be to increase the delay of short paths as much as possible without increasing the propagation delay, and keep the area increase within a limit. As mentioned earlier, short path delays can be increased without affecting propagation delay for carry look-ahead adders and other smaller circuits. However, this is done manually, and the area overhead is very high for 64-bit adders. Minimizing short path constraints, without increasing propagation delay may not be possible for many practical circuits. In that case, we can allow a small increase in the propagation delay, if that increase can allow higher margin for TS.

We introduce Min-arc algorithm for increasing contamination delay of logic circuits up to a defined threshold. We adopt an approach closely resembling min-cut algorithm for flow networks. The basic idea of the algorithm is to identify a set of edges, from here on we refer it as the *cut-set*, such that adding a fixed amount of delay to the set does not affect the delays of any long paths. However, an important difference between this and traditional flow networks is that the cut-set for the Min-arc may not necessarily break the flow of the network. But rather, the cut-set is a subset of edges in the actual (rather traditional) min-cut. The reason why we do not consider a traditional min-cut is to not unnecessarily add delay buffers where it is not needed. However, a subset of the min-cut edges is essential to keep the addition of delays minimal. Another reason for increasing path delay in batches is to keep the structure of the logic network unaltered from the original network. Benefits of maintaining path delay distribution is explained in Section 4.6.

The basic outline of the Min-arc algorithm to increase the short path delay of the circuit up to a required value is presented in Algorithm 6. The entire procedure is divided into six basic steps, in which the first and last steps are one-time operations, converting the logic circuit to

an equivalent graph network and vice-versa. The remaining parts of the algorithm modifies the graph into a weighted graph network and iteratively updates the prepared network by adding the necessary delay to the selected interconnection using the modified min-cut procedure. The forthcoming portion of this section is devoted towards explaining each step in detail.

Algorithm 6 Steps for manipulating short path delay in logic circuits

STEP A: Convert combinational circuit to a graph
STEP B: Get minimum and maximum path through every edge
STEP C: Prepare graph for min-cut
STEP D: Do min-cut on the graph obtained in step 3
STEP E: Add delay to the edges returned by min-cut
STEP F: Update the graph and repeat Steps 2 through 6 until contamination delay is increased up to the required value
STEP G: Convert the graph back to combinational logic circuit

---

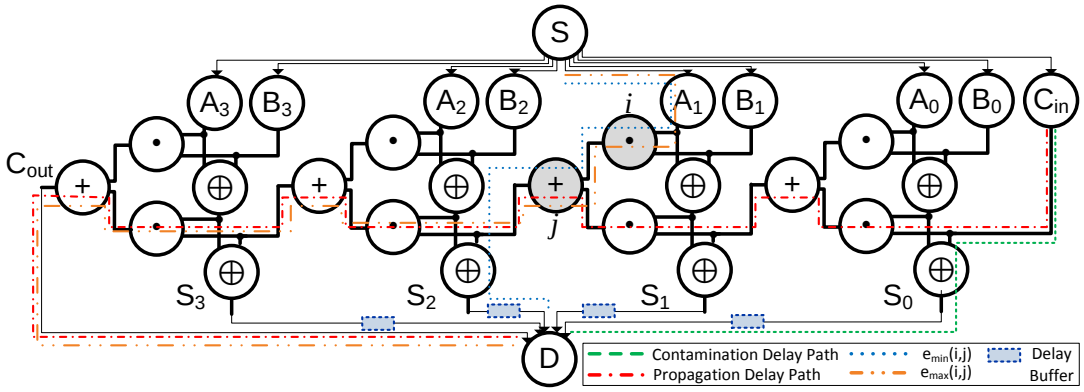


Figure 4.5 Illustration: Network model for 4-bit ripple carry adder. (Assuming unit interconnect and logic delays)

4.5.1 Construction of Weighted Graph Network

The first step is to convert the given combinational logic into a directed graph, where the logic blocks becomes the nodes, and the interconnections from each logic block to others form the directed edges. The nodes and edges may be weighted depending on their time delays. To this graph we add a source, S, from which edges connect to all the inputs, and a drain,

Terms	Definitions
$MAX(i,j)$	Maximum path from node i to j, incl. i and j
$MIN(i,j)$	Minimum path from node i to j, incl. i and j
$MAX(S,D)$	Propagation delay of the circuit, T_{PD}
$MIN(S,D)$	Contamination delay of the circuit, T_{CD}
$e(i,j)$	Edge from node i to j
$wt(i,j)$	Weight of edge from node i to j, not incl i and j
$e_{max}(i,j)$	$MAX(S,i) + wt(i,j) + MAX(j,D)$
$e_{min}(i,j)$	$MIN(S,i) + wt(i,j) + MIN(j,D)$
LWY	Percentage of leeway (0-1) on critical path while adding buffer. E.g., $LWY = x\%$ allows the target network to have $T_{PD}(1+x)$ as the final propagation delay
THD	Normalized threshold (from T_{CD} to T_{PD}) below which we do not want any short paths
INF	A very large integer value
$SCALE$	A moderate integer value, ($> T_{PD}$), to scale the weight to a new range
$func()$	A function dependent on T_{PD}, T_{CD}, $e_{max}(i,j)$ and $e_{min}(i,j)$. Returns a real number, 0-1. In this work, we define this as $\sqrt{\frac{(e_{max}(i,j)-THD)}{(T_{PD}-THD)}} \times \sqrt{\frac{(e_{min}(i,j)-T_{CD})}{(THD-T_{CD})}}$

Table 4.2 Definitions

D, to which all the outputs connect. Note that there is a zero weight for S, D and all the edges from/to them. Figure 4.5 illustrates an example network model for a 4-bit ripple carry adder with S and D added. T_{PD} and T_{CD} of the logic circuit are highlighted in the figure. It is necessary to preserve the node types whether they are logic gates, buffer delays, input or output pins. Also it is important to note the type of logic for a logic gate node. This is important in order to maintain functional correctness of the circuit.

4.5.2 Finding the Minimum and Maximum Path

Once the directed network is constructed, the next step is to mark the edge weights for generating the cut-set. Before doing so, we introduce and define several terms and symbols as illustrated in Table 4.2, which will be used in the remaining steps. We calculate the longest and shortest distances for every edge from source and drain. That is, we obtain $MAX(S,i)$, $MAX(j,D)$ $MIN(S,i)$ and $MIN(j,D)$ for every edge $e(i,j)$ in the weighted graph. We

use Djikstra's algorithm to calculate $MAX()$ and $MIN()$ functions. From this, we calculate $e_{max}(i,j)$ and $e_{min}(i,j)$ for every edge, $e(i,j)$ as described in Table 4.2, which corresponds to the longest and shortest paths of the logic network through that edge. The paths marking $e_{min}(i,j)$ and $e_{max}(i,j)$ for randomly chosen nodes i and j for the 4-bit ripple carry example is depicted in Figure 4.5. In a similar manner, the minimum and maximum weights for every edge are calculated.

4.5.3 Preparing Graph for Min-cut

From steps B through F we re-construct the prepared weighted graph network as and when we select a minimum weight interconnection to add the delay buffer. We re-construct the graph from the previous state using new edge weights. The edge weights are calculated in such a manner that the minimum weighted arc gives the most favorable interconnection where to add delay. The procedure for calculating new weights for every edge, $e(i,j)$, is described in Algorithm 7. The edge, $e(i,j)$ may fall under one of the four categories listed in the algorithm. For the first two cases, the edge weight is calculated as the sum of $e_{min}(i,j)$ and $e_{max}(i,j)$. This is the general scenario where the minimum and maximum paths are added as edge weights. The former case is the scenario of a short path, where $e_{max}(i,j)$ is smaller than the threshold for contamination delay. The latter case is when the selected edge, $e(i,j)$, has a delay such that the shortest path is more closer to the threshold than the longest path is to the propagation delay. In other words, when a delay buffer is added to any edge in the path to increase the short path delay by the given threshold, the maximum delay increase affecting a critical path is still within propagation delay of the circuit. The third scenario is when the longest path exceeds propagation delay including leeway. This edge is critical and by no means can buffers be added to this. Hence, we substitute a large number (INF) as the edge weight so that this edge is never picked as part of the min-cut. Finally, we have a case when delay buffer addition exceeds or gets very close to the propagation delay. In this case, we scale the edge weight moderately higher than the original range. This addresses the case where addition of buffer to any edge affects longer paths.

4.5.4 Finding the Min-cut

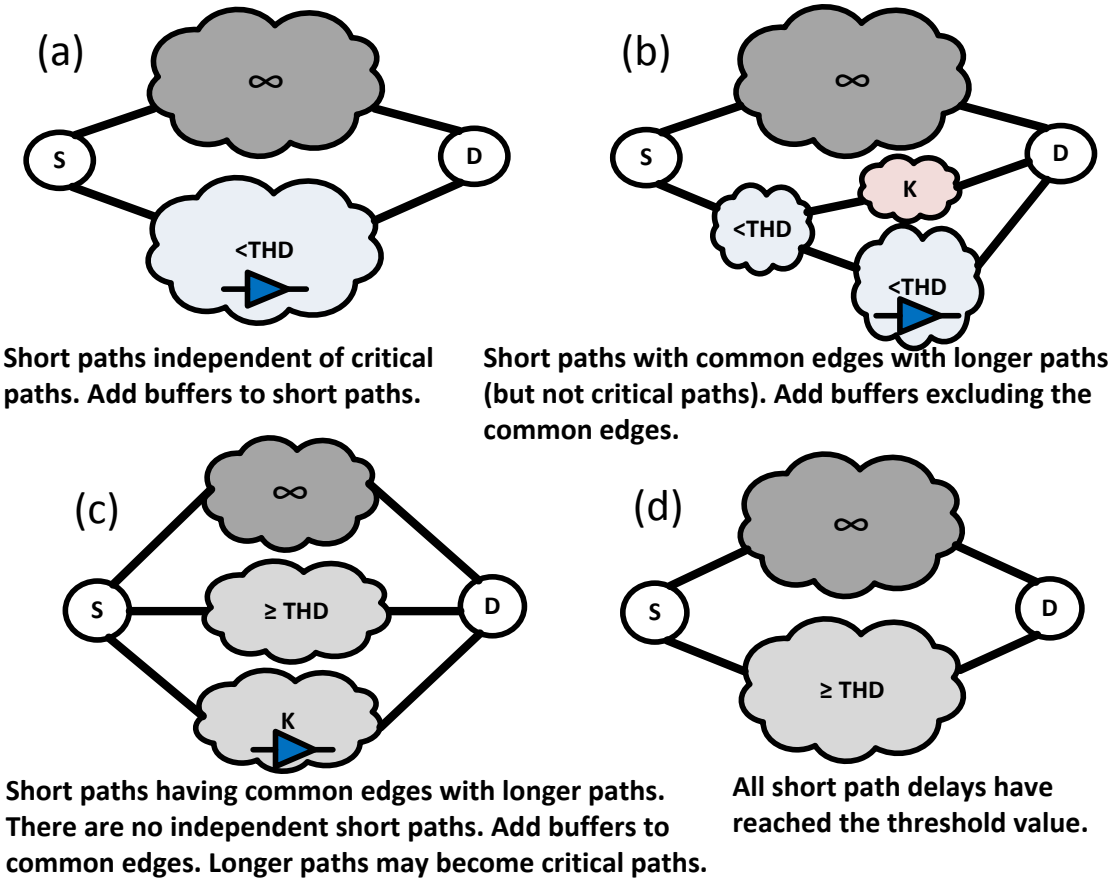


Figure 4.6 Illustration of four different scenarios finding the cut-set in Min-arc algorithm

Once the graph weights are re-assigned, the cut-set is determined. We use a variant of Edmonds−Karp min-cut algorithm of the graph network. The cut-set consists of edges with minimum weight in the new graph. Figure 4.6 illustrates the different scenarios in determining the cut-set. The cut-set re-definition is necessary because the traditional min-cut always has at least one edge in the critical path. Figure 4.6(a) shows how a logic circuit is divided into critical and non-critical paths. As long as the non-critical paths are independent of critical paths, buffer delays can be added to the former ones. In this case, the min-cut excludes all the critical paths. Generally, the scenario is not this straightforward. As illustrated in Figure

4.6(b), the short paths are intertwined with longer paths that are not critical paths. In such cases, the weights of the longer paths are scaled to a different range (in this case K). If there is a subset of short paths that exist independent of the longer paths, buffer delays are added to this subset. We noticed that this is the most common scenario in the benchmark circuits. Once all the independent short paths have been added with corresponding delays, the new circuit is left out with paths that are scaled as shown in Figure 4.6 (c). Buffer delay is added to the scaled paths, which runs the risk of modifying longer paths into critical paths. The final circuit is shown in Figure 4.6 (d), where there are only critical paths and paths that have delay meeting the threshold requirements. In the ripple carry example, the case is similar to Figure 4.6 (a). The cut-set is thus all the paths excluding the critical path. Figure 4.5 shows the min-cut where the buffers are added.

4.5.5 Adding Buffer Delays

The buffers are carefully placed on edges where it would not affect the longest paths. Thus, the amount of delay each buffer should have depends on the path connectivity, which may have major impact on the timing error occurrence. For instance, it is possible to add delays such that all the paths have delay equal to the critical path delay. In most practical circuits, pushing all the paths to a certain delay interval would result in sudden rise in the timing errors, causing overhead due to error recovery. So, it is always necessary to keep in mind while designing the algorithm that there is a gentle rise in path delays from one interval to the other. Buffers are added on the edges present in the cut-set. The amount of delay added, $delay(i,j)$, for any edge $e(i,j)$ is given by Equation 4.7.

$$delay(i,j) = min((THD - e_{min}(i,j)), (T_{PD} - e_{max}(i,j))) \tag{4.7}$$

The delays for all the edges in a cut-set, for a given iteration, are added at the same time. While adding delay, we ensure that in the same iteration, to no other edge the delay is added that are connected to paths through this edge.

4.5.6 Satisfying Conditions

We iterate steps B through F until the minimum condition for the shortest path is met or until there is no other edge where delay can be added without affecting T_{PD} of the circuit (including LWY). Step F checks if the desired value of contamination delay is reached. Once the required conditions are met, no more buffer additions are carried out and we move on to step G. From our experiments, we found that the minimum condition for contamination delay is achieved for all the circuits we evaluated.

4.5.7 Converting Graph to Logic Circuit

The final step is to revert back to the original circuit once the short paths lengths are increased to the desired level. Since we record the node types in the network graphs in step A, it is possible to re-build the circuit from the graph network with the added buffers. It should be noted that we do not optimize the logic of the circuit, whereas we only add additional buffers preserving the original logic of the circuit.

Quantifying Min-Arc Algorithm

The time complexity of Min-arc algorithm is mainly affected by Steps B and D. Let $|V|$ be the total number of logic blocks (vertices) and $|E|$ is the total number of interconnections (edges) in the logic circuit. Using Djikstra's shortest path algorithm, the worst case time to calculate $MAX()$ and $MIN()$ functions is $O(|V|^2)$. For finding the minimum weighted edge min-cut for the graph network, it takes $O(|V||E|^2)$. In the worst case every edge becomes a part of the cut-set. That is, there are at most $|E|$ iterations. Hence, the overall time complexity of the Min-arc algorithm is $O(|V|^2|E| + |V||E|^3)$.

4.6 Evaluation of Min-arc Method

Although the time complexity of Min-arc algorithm is polynomial order, it is necessary to consider its performance on practical circuits. We evaluate the algorithm on ISCAS'85 benchmark suite [89]. The suite provides a diverse class of circuits in terms of number of IOs,

Algorithm 7 Re-calculation of edge weight for edge $e_{max}(i,j)$

```
1: if e_max(i,j) ≤ THD then
2:    wt(i,j) = e_min(i,j) + e_max(i,j)
3: else
4:    if (THD − e_min(i,j) < (T_PD − e_max(i,j)) then
5:       wt(i,j) = e_min(i,j) + e_max(i,j)
6:    end if
7: else
8:    if e_max(i,j) > (T_PD(1 + LWY)) then
9:       wt(i,j) = INF
10:   end if
11: else
12:   wt(i,j) = SCALE * func()
13: end if
```

logic gates and interconnections (nets). Table 4.3 lists a brief description and other relevant details of the circuits. All the circuits were transformed into network graphs as specified in Section 4.5. The interconnect delays and logic cell delays were obtained by synthesizing the circuits for $45nm$ technology using OSU standard cell library [64]. All the configurations $(L\langle l\rangle - T\langle t\rangle)$ described in Section 4.4.2 were investigated.

Circuit	Description	Inputs	Outputs	Gates	Nets	Area(Buf)
c432	27-channel interrupt controller	36	7	205	386	5360.698
c499	32-bit SEC circuit	41	32	277	513	7821.103
c880	8-bit ALU	60	26	471	841	11791.877
c1355	32-bit SEC circuit	41	32	621	1169	17166.807
c1908	16-bit SEC/DED circuit	33	25	940	1581	24947.760
c2670	12-bit ALU and controller	233	140	1644	2665	36016.406
c3540	8-bit ALU	50	22	1743	3033	49139.693
c5315	9-bit ALU	178	123	2610	4810	71726.222
c7552	32-bit adder/comparator	207	108	3830	6568	101953.107

Table 4.3 Area increase in terms of buffer delay (ps)

Interesting results were noted in this study. First, for all the circuits, the Min-arc method was able to increase the short path delays to the desired threshold levels without any leeway on PD. Even then, we continued with all the configurations to include leeway in order to study the effect of including them. We present the results only of a few selected circuits and average

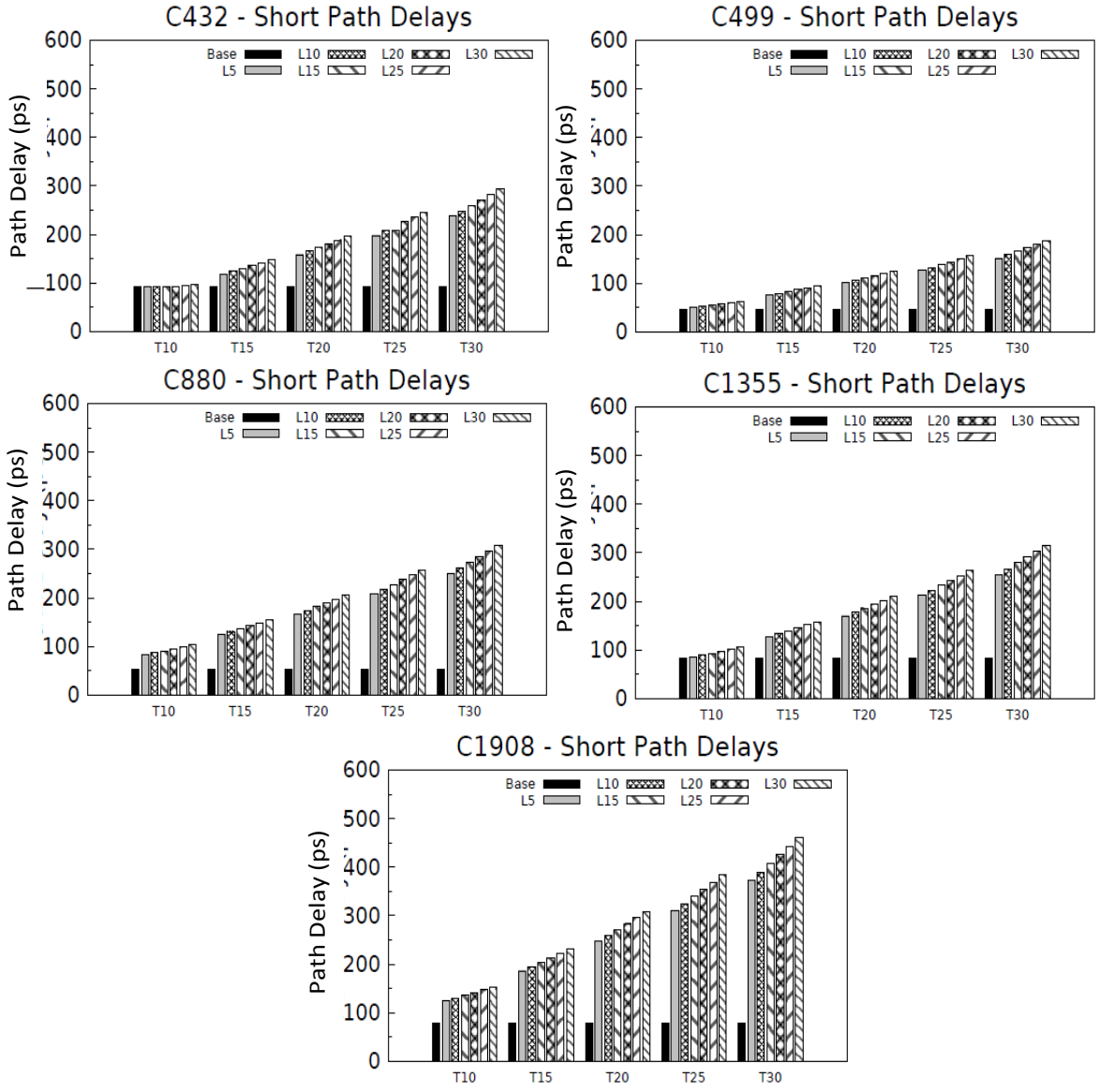


Figure 4.7 Charts showing increase in contamination (short path) delay of circuits (Part 1/2)

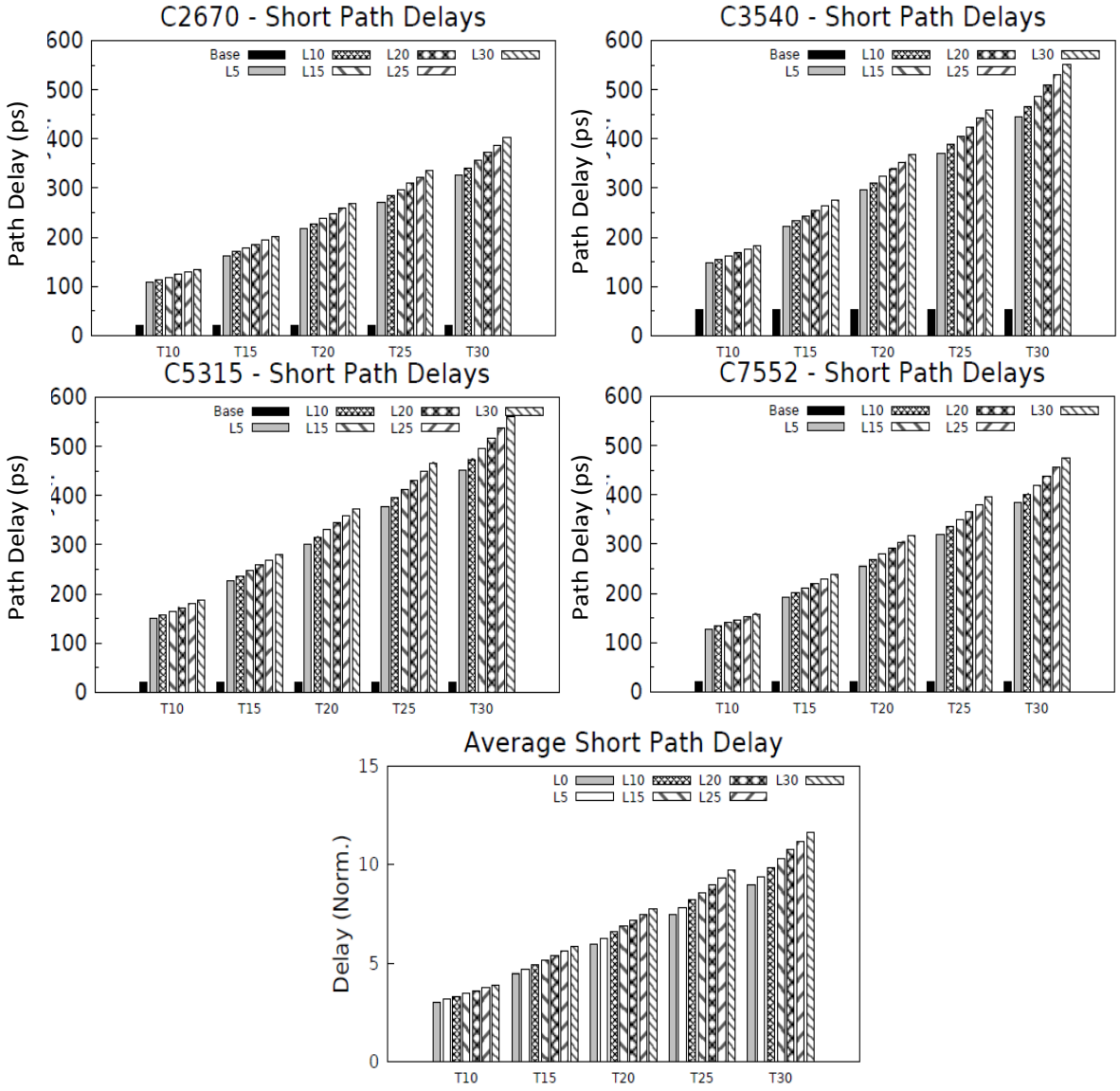


Figure 4.8 Charts showing increase in contamination (short path) delay of circuits (Part 2/2)

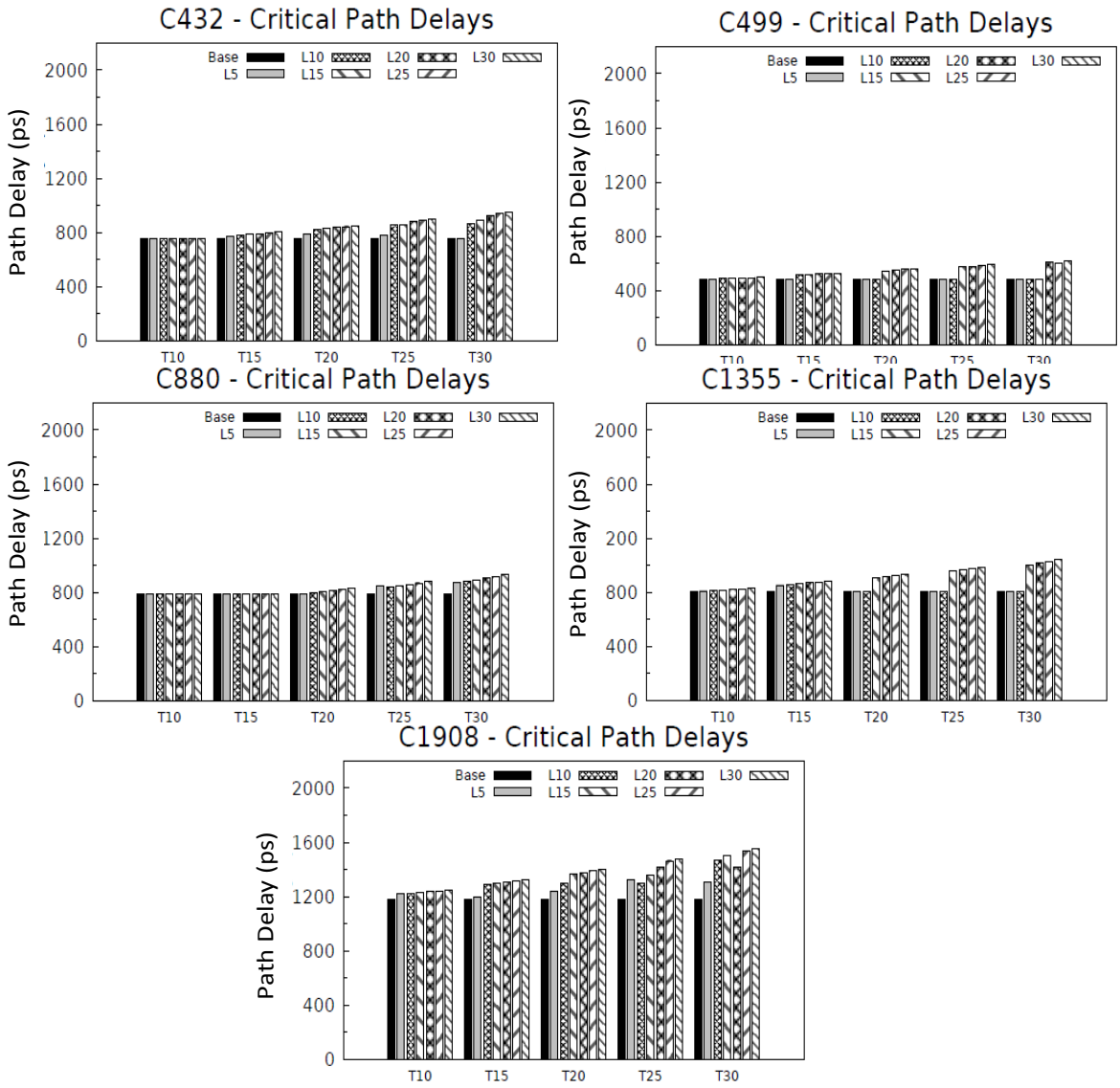


Figure 4.9 Charts showing increase in propagation (critical path) delay of circuits (Part 1/2)

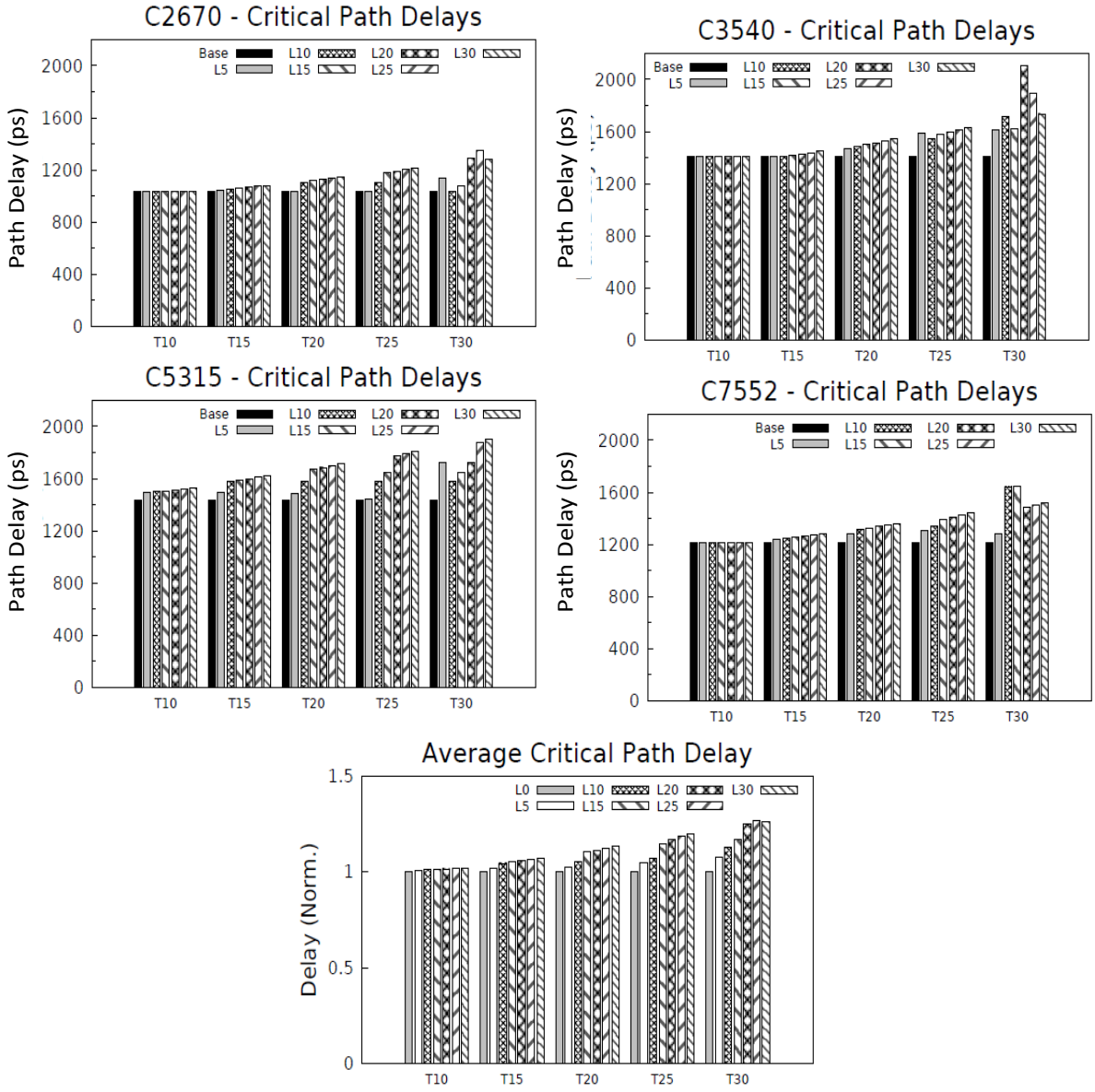


Figure 4.10 Charts showing increase in propagation (critical path) delay of circuits (Part 2/2)

of all circuits. It was found that circuit characteristics (i.e., size and connectivity) have strong effects on how the algorithm performs. Figures 4.7, 4.8, 4.9, and 4.10 illustrate the increase in the short path delays, and critical path delays, respectively, for different configuration in $c432$ and $c5315$ circuits. The charts also show the average increase of these delays for all nine circuits. For smaller circuits (as in $c432$), we notice that there is not much the algorithm could possibly do, as there is a higher chance of affecting the critical path by adding delay to any net. In $c432$, we notice that the maximum delay increase of short paths from the base circuit with $91ps$, (with 0% leeway) is around $225ps$. However, in larger circuits (as in $c5315$), delay buffers were more easily added. This is seen in $c5315$, where short path delay is increased from $20ps$ to $430ps$, again with 0% leeway. In other words, as the circuit size increases, the number of independent short paths also becomes more, allowing easy inclusion of delay buffers. It should also be noted that there is not much delay increase from $L0$ to $L5$ or other higher levels of leeway on PD. Increasing threshold on the other hand tend to have a great impact in increasing the CD. On an average, there is a $1.5\times$ factor of increase from one threshold level to the next for all configurations. Assuming $LWY = 0$, we were able to achieve $300\% - 900\%$ increase in CD, and increasing LWY steadily from 5 to 30%, we observed increase of CD in a saw tooth pattern achieving $315\% - 1165\%$ increase in CD. It should be observed from the critical path delay patterns that the algorithm strictly adheres to the critical delay limits.

One major effect of adding buffers to circuits is that it affects path delay distribution. Although our goal is to increase the CD to a threshold limit, pushing a set of paths to one side may increase the timing error rate during execution. Therefore, it is important to maintain the delay distribution of the circuit paths without much deviation. For all the circuits we tested, Min-arc algorithm was able to closely maintain the path delay distribution. In most cases, while adding delay buffers, it was possible to shift all the delay intervals to a new level, thereby maintaining the circuit structure as much as possible. Even though the structure of the circuit is maintained, the short paths are now pushed to higher delay slots, thereby increasing the possibility of error occurrences. This corresponds to Dev, mentioned in Section 4.4. However, we expect the increase in error rate to be a nominal value.

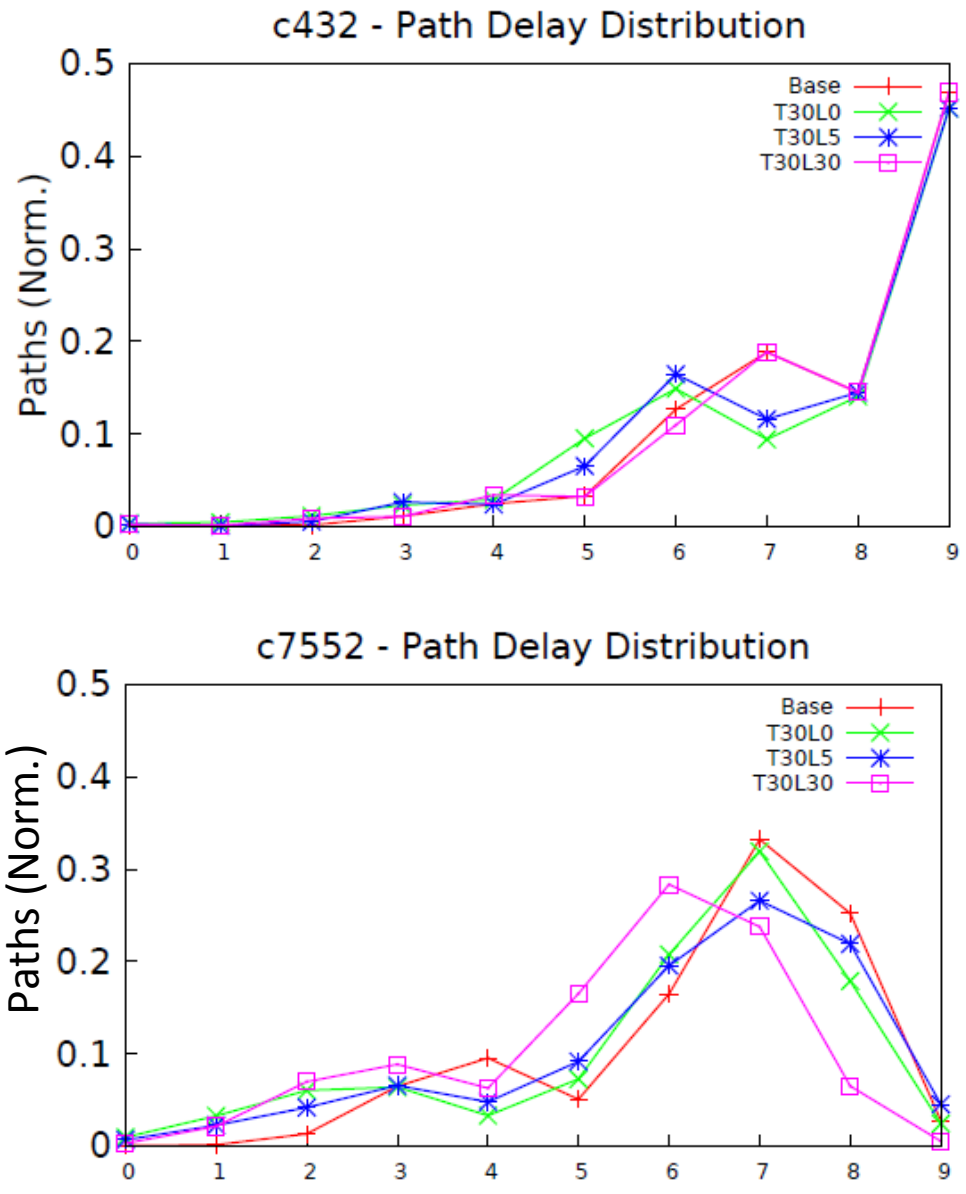


Figure 4.11 Path delay distribution from CD to PD for $c432$ and $c7552$

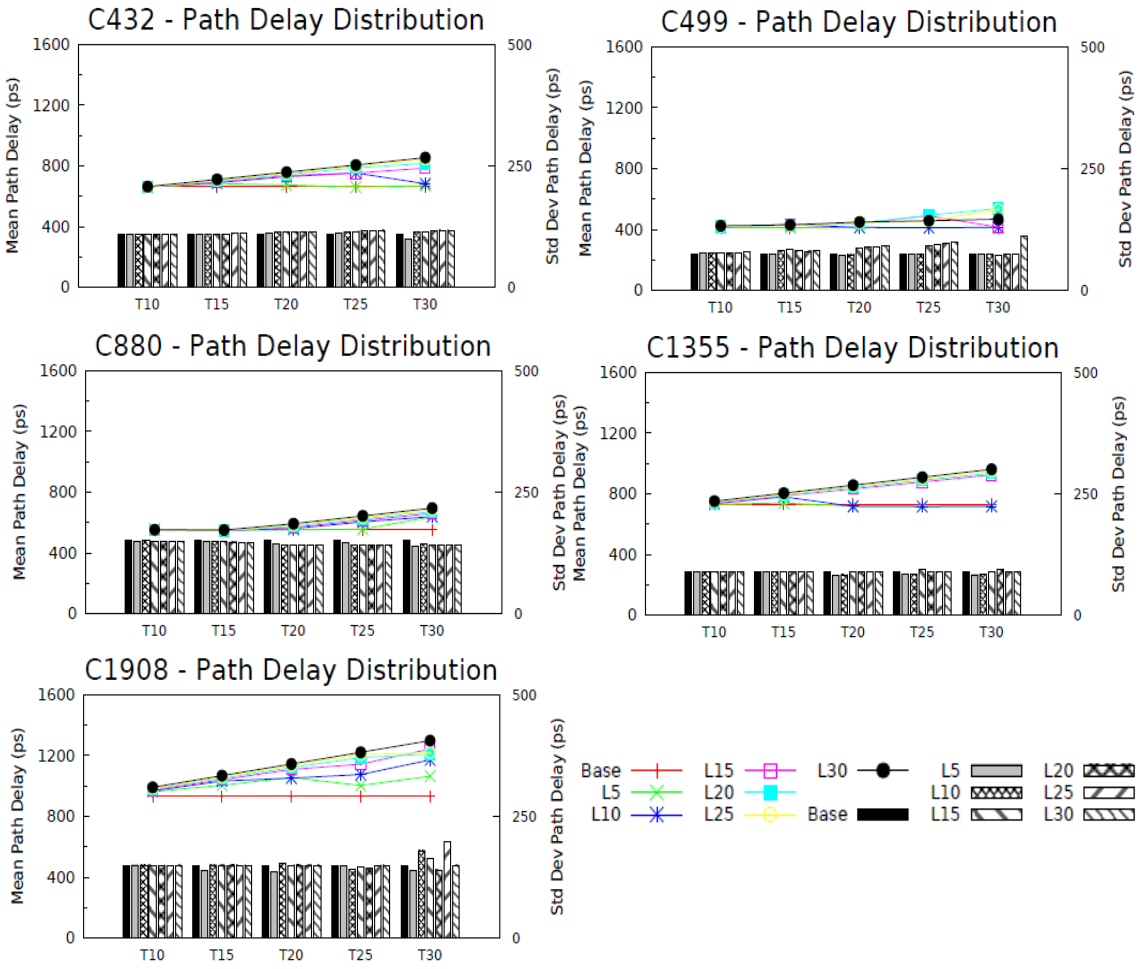


Figure 4.12 Average path delay distribution, in terms of mean and deviation (Part 1/2)

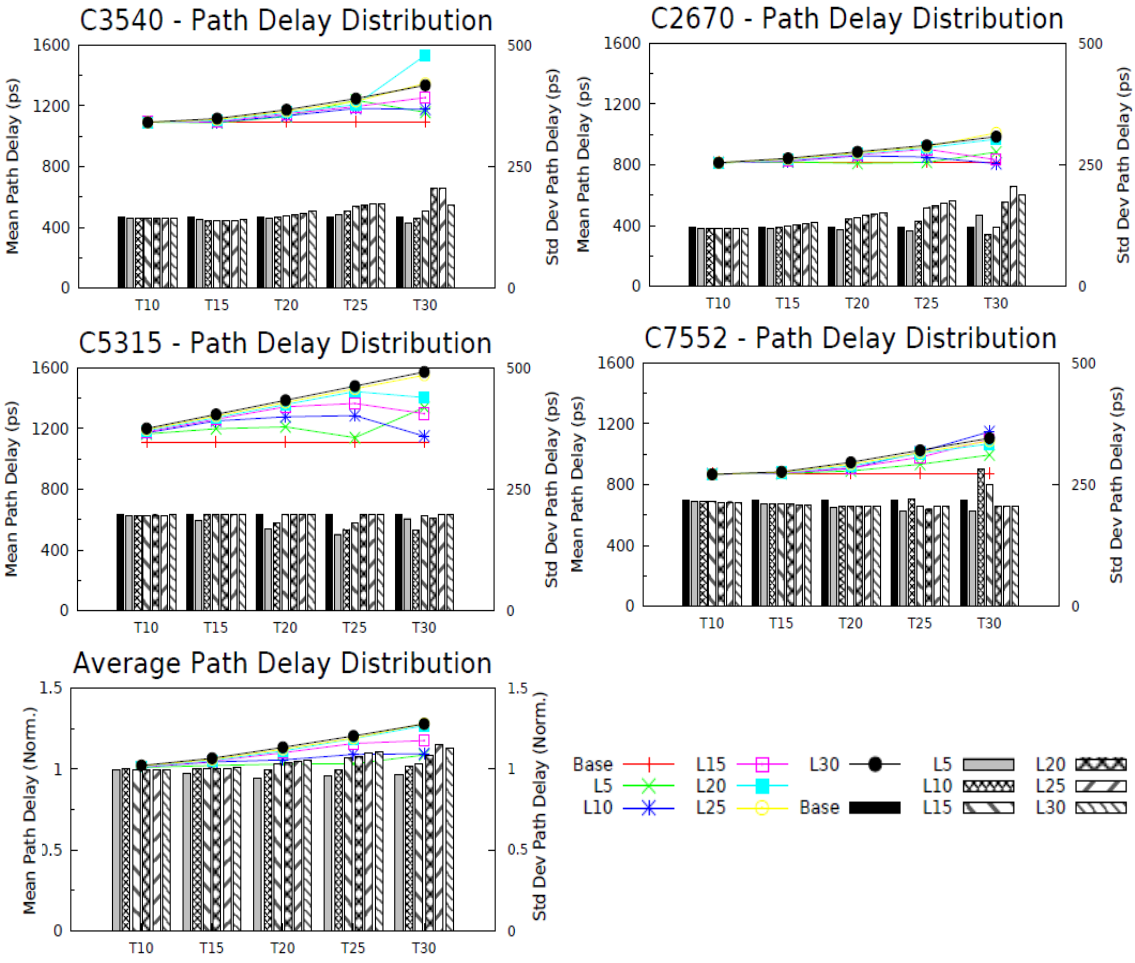


Figure 4.13 Average path delay distribution, in terms of mean and deviation (Part 2/2)

Figure 4.11 illustrates the path delay distributions for selected configurations of two circuits ($c432$ and $c7552$). We noticed that for $c432$ (and other smaller circuits), the path structure were mostly maintained. In the case of $c7552$ (and other larger circuits), the circuit structure was altered moderately. To illustrate this point further, we plotted mean and standard deviation of all the circuits. Figures 4.12 and 4.13 showcase the distribution plots of each of the circuit and average of all the circuits. We noted that the smaller circuits suffer from negligible deviation from original circuit in spite of higher mean, and the larger circuits are vulnerable to change in structure. From the average plot, it is also evident that higher leeway values cause more deviation. A maximum deviation of -12% and $+16\%$ were observed for $T30L0$ and $T30L30$ configurations, respectively.

4.6.1 Area overhead

The overhead for Min-arc algorithm is the area penalty. More the circuit allows adding buffers, more the overhead in chip real estate. We estimate the original circuit area in terms of buffer delays, and compare the area increase for each of the configurations. This study facilitates us to narrow down the choices of L and T for any given circuit. Table 4.4 enlists the percentage area increase for various L and T combinations, for all the circuits. It is important to choose the configuration that has highest increase in delay with moderate increase in area.

Without any leeway (corresponding to $L0$), with every 5% increase in T there is around 20% increase in area. This holds for most circuits, except for smaller circuits as in $c499$ and $c880$, where it is around 10%. A maximum of 100% increase is observed for $c2670$ at $T = 30\%$. For this maximum threshold, there is a wide range of area increase across the benchmark circuits. We did not see any strong relation between circuit size and the area increase. This means that it is the circuit connectivity that has a major role to play on buffer placements. For $T = 30\%$, the minimum area increase of around 10% is observed for the circuit $c3540$.

A general observation from our study is that the area increases with L or T. However, we observed quite a few configurations, where the area decreases with L or T. This reflects how the algorithm handles different input combinations independently, rather than building from

Ckt		L0	L5	L10	L15	L20	L25	L30
c432	T10	000.000	000.000	000.000	000.000	000.000	000.119	000.338
	T15	002.233	001.884	002.306	002.727	003.149	003.570	003.992
	T20	010.911	014.424	005.738	006.441	007.186	008.029	008.872
	T25	027.913	034.655	010.918	012.148	012.244	013.380	014.610
	T30	055.948	062.209	070.731	019.970	020.029	019.528	021.004
c499	T10	001.295	002.278	003.262	004.246	005.229	006.213	007.196
	T15	011.131	012.606	014.082	015.557	017.032	018.508	019.983
	T20	020.967	022.934	024.901	026.868	028.836	030.803	032.770
	T25	030.803	033.262	035.721	038.180	040.639	043.098	045.557
	T30	040.639	043.590	046.540	049.491	052.442	055.393	058.343
c880	T10	000.875	001.076	001.338	001.673	002.009	002.371	002.773
	T15	004.642	005.397	006.152	006.907	007.830	008.798	009.855
	T20	013.250	015.129	013.595	015.339	017.084	018.828	020.573
	T25	024.952	035.427	023.190	025.371	027.552	029.732	031.913
	T30	032.744	037.751	049.281	035.402	038.019	040.636	043.253
c1355	T10	000.000	000.139	000.893	001.647	002.401	003.155	003.909
	T15	007.900	009.566	009.189	010.320	011.451	012.582	013.714
	T20	022.983	026.000	029.016	018.993	020.501	022.009	023.518
	T25	038.066	041.837	045.608	027.666	029.551	031.436	033.322
	T30	053.150	057.675	062.200	036.338	039.330	040.863	043.126
c1908	T10	003.140	003.732	004.324	004.915	005.507	006.099	006.690
	T15	008.066	009.944	010.832	011.719	012.607	013.495	014.382
	T20	015.359	016.111	018.340	018.524	019.707	020.890	022.074
	T25	024.749	025.100	027.090	028.884	026.938	028.286	029.765
	T30	035.196	035.855	035.942	037.866	040.209	044.067	037.457
c2670	T10	023.403	025.084	026.806	028.584	030.362	32.140	033.982
	T15	041.617	044.526	047.441	050.387	053.334	56.280	059.227
	T20	060.493	064.606	069.049	072.991	076.950	80.936	084.922
	T25	081.137	088.340	092.171	095.884	100.867	105.849	110.832
	T30	100.542	117.577	113.181	132.332	131.624	138.105	136.851
c3540	T10	000.624	000.696	000.768	000.840	000.937	001.038	001.139
	T15	001.982	002.810	002.708	003.085	003.474	003.883	004.293
	T20	005.350	005.159	007.636	006.366	006.969	007.573	008.177
	T25	007.657	017.588	009.916	013.920	015.115	011.390	012.181
	T30	010.828	015.571	019.480	013.445	045.045	021.419	027.415
c5315	T10	007.858	011.719	012.840	013.972	015.136	016.306	017.477
	T15	024.559	025.690	025.784	027.600	029.417	031.233	033.049
	T20	039.732	042.558	041.610	041.613	044.074	046.536	048.997
	T25	063.509	066.116	058.342	056.139	058.843	061.920	064.996
	T30	074.338	102.908	085.120	107.317	095.169	077.304	080.996
c7552	T10	005.393	05.500	005.871	006.265	006.663	007.068	007.505
	T15	009.431	010.237	011.069	011.919	012.795	013.698	014.611
	T20	015.342	016.454	017.846	018.954	020.243	021.532	022.822
	T25	022.605	025.980	026.880	026.368	029.263	029.592	031.204
	T30	040.318	039.191	043.577	041.745	035.718	037.652	039.586

Table 4.4 Area increase in terms of buffer delay (ps)

previous level output. We noticed several places where area would decrease with L (shaded blue in Table 4.4). As illustrated, there is at least one place where this occurs in each circuit, with the exception of $c499$ and 2670. In the case of T, we see that there are only a couple of configurations where this occurs, namely in $c3540$ (underlined in Table 4.4). This explains how target threshold for short paths affect increase in area. In most cases we noticed around 2% increase in area for every 5% in L. In majority of the cases, we noted only moderate increase in area ($< 50\%$). We observed 12 cases where the area increase was more than 100%, in which 10 of them are from the same circuit, $c2670$. This is a 12-bit ALU with controller ($c2670$) that has a lot of parallel paths with few common edges. Similar but less intense effect is seen in the case of the 9-bit ALU ($c5315$). The configurations where the area increase exceeds 100% is highlighted orange in the table.

4.7 Summary

Contamination delay is one of the major bottlenecks for achieving higher performance in timing speculation architectures. In this paper, we investigated the theoretical margins for improving performance for the dual latch framework. We brought forward the limits to performance enhancements in timing speculation. Using our analysis, we demonstrated how much performance improvement is achievable by increasing the contamination delay of the circuit without affecting the critical path delays. Performance gains were attained even for the cases affecting propagation delay by up to 10%. We studied further how these gains vary with target timing error rate.

The main goal of this paper is to increase the short path delays to a specified threshold, without (or minimally) affecting the critical path delays. We proposed the Min-Arc algorithm to achieve this goal. We presented the results for ISCAS-85 circuits, where we have shown that the Min-Arc is able to increase the contamination delay of all the circuits without affecting propagation delay. We analyzed further as to how much these short paths increase while allowing a small leeway to critical path delay. We observed moderate area increase in the circuits implementing the Min-arc algorithm. Finally, we discuss how the algorithm preserves

the path delay distributions of the circuits and therefore, closely maintaining the rate of timing error occurrences from the original circuit.

To conclude, Min-arc algorithm successfully increases the contamination delay of logic circuits with moderate area penalty. The results we have obtained are very promising, opening up different directions for the near future. Managing short paths leads to different error rates and power dissipation. Studying the interdependencies between different parameters certainly helps us understand timing speculation architectures better. Comparative study of synthesized circuits with and without Min-arc algorithm, and realizing it in hardware, like FPGAs or ASICs, will make the case stronger for timing speculation in commercial circuits.

CHAPTER 5. POWER BUDGETING USING DYNAMIC V-F PAIRING

Power management techniques have been well studied in the recent few years. Existing methods choose one among the different available power states and can precisely control the dynamic power of the system, to achieve the desired power/energy set point. The aim of an efficient power management scheme is not only be to reduce power as much as possible, but also to allow the processor to dissipate only as much as the budget allows. More importantly, the challenge is to curtail the loss incurred due to power level transitions and circuit slow down to minimal. As it was illustrated in Chapter 3, the predetermined power states in traditional DVFS are set at the worst-case level, which leads to significant performance loss. It was also shown how effectively DVARFS controls on-chip temperature, and minimizes power dissipation, while enhancing performance. Although by intuition it is apparent that DVARFS is apt for energy-efficient power management, it is necessary for us to do a comparative evaluation against the existing schemes.

In this chapter, we implement a power management strategy similar to Intel SpeedStep. Our goal is to show the effectiveness of adaptive frequency tuning beyond worst-case boundaries in dynamic power management within a power budget as opposed to dynamic power level shifting. We improvise the control mechanism proposed in the earlier chapter, thus making the system more adaptable, while meeting the power constraints within safe thermal limits.

5.1 Power Dissipation in Aggressively Clocked Systems

In Section 3.1.2, we re-framed the basic performance equations making them suitable for aggressively clocked processors. In this section we delve further into some of the important metrics learn more about these systems. From our analysis, we devise an energy-efficient,

power management scheme. By this we mean that our intention here is to sustain a power constrained system, while keeping up to the performance as much as one can.

5.1.1 Computation Bounded Workloads

The speed-up expression presented in Equation 3.2 includes three parts. The first part corresponds to the compute cycles in pipeline without IO/memory access. The final part corresponds to the timing error recovery associated with overclocking. The remaining part of the equation represents the fraction of time spent in IO/memory. In DVARFS, the amount of time spent for recovery is typically one cycle, that is $k = 1$.

From Figure 5.1(a) it is evident that during traditional DVFS, when frequency is brought down ($q < 1$), the total execution cycles reduce. This is because of the relatively fewer cycles spent for memory. The figure shows the effect of frequency scale down on performance for various memory access factors. For instance, a memory bound workload (say $\alpha = 25\%$) offers close to 1.5 times more voltage scale down compared to a CPU bound workload (say $\alpha = 5\%$), for the same performance loss of 15%.

On the other hand, the benefits of reliable overclocking surpass the memory penalties under controlled error rate. This is clearly understood from the series of charts (b), (c) and (d) of Figure 5.1. Here, we depict the speed-up for a spectrum of memory access factors relative to target error rate, S_e, for different values of q.

For performance enhancement, the system must tolerate $20\%, 50\%, 70\%$ and 100% of timing exceptions at the overclocking rates $q = 1.2, 1.5, 1.7$ and 2.0, respectively. In the forthcoming sections, we show that for practical workloads the number of timing errors produced is quite low for smaller values of q, but quickly reaches 100% for higher values.

Without loss of generality, we assume core activity to be directly proportional to IPC of the processor core. In order to analyze the effect of frequency scaling on CPU threads, we ran all the SPEC workloads in two different configurations in *simple* mode. In our study, we implemented two different frequency settings, $2.5GHz$ and $3.4GHz$. The core activity for 12 SPEC benchmarks for the two settings are illustrated in Figures 5.2 and 5.3. Each time

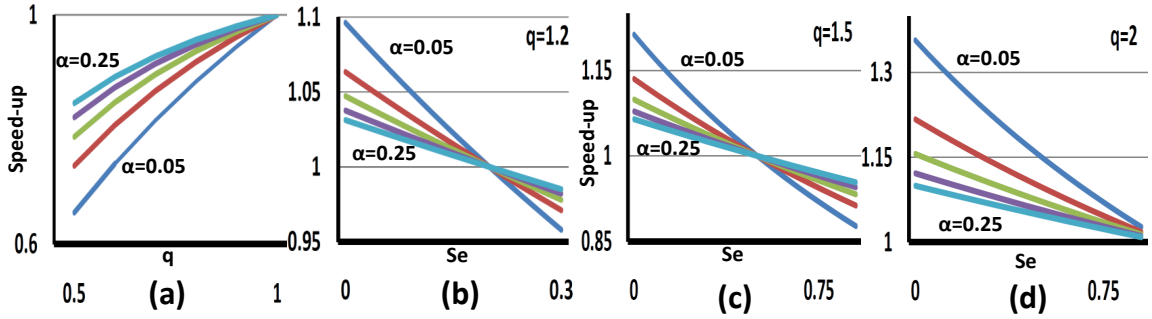


Figure 5.1 Performance analysis of workloads with varied computation boundedness

SPEC INT	bzip2	crafty	gap	gzip	mgrid	vpr
%Load/Store	35.24%	32.94%	32.25%	22.21%	36.66%	40.78%
SPEC FP	apsi	equake	galgel	lucas	mcf	mesa
%Load/Store	35.83%	42.53%	43.34%	21.76%	34.30%	35.79%

Table 5.1 Percentage of load-store instructions in SPEC 2000 INT and FP workloads

stamp depicts 10,000 cycles. We noticed that the workloads have a slow start for the initial few time stamps in spite of fast forwarding. From the two traces, we observed that increasing frequency for certain workloads results in increased activity. In other words, frequency scaling has positive impact on IPC of the system running computation bound threads. As a result, these threads while running on a $3.4GHz$ processor complete execution several time stamps ahead of those running at $2.5GHz$.

Table 5.1 enlists the percentage of load-store instructions in SPEC 2000 benchmark suite. The data is obtained from [90]. After careful observation, we inferred that all the threads with the smaller percentage of load-store instructions tend to finish faster than those with higher fraction of load-store. For instance, *bzip*2 and *lucas* are cases in point, where clock frequency increases core activity resulting in faster execution.

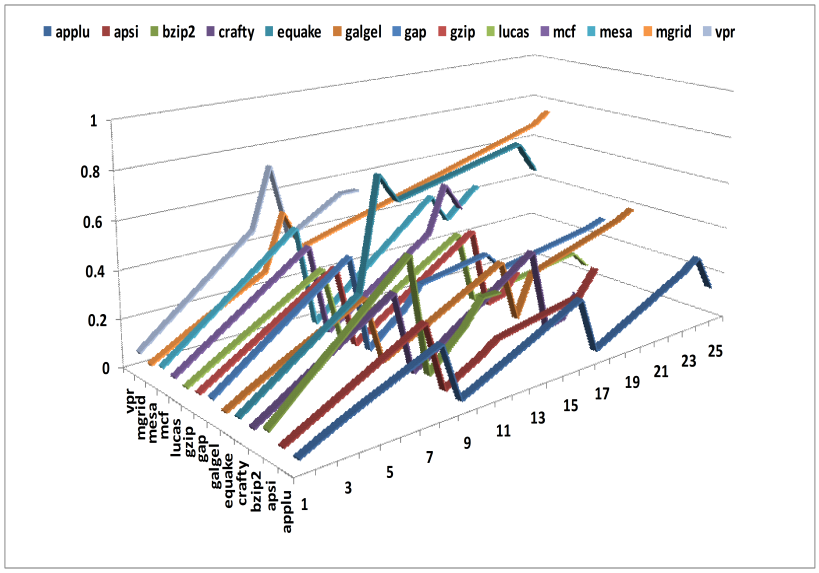


Figure 5.2 Thread activity trace for SPEC 2000 workloads running at $2.5GHz$

5.1.2 Dynamic Power Dissipation

As mentioned earlier, overclocking increases the switching activity of the circuits causing more dynamic power dissipation. Our intention here is to seek an upper bound on voltage to save power in the overclocked processor. Eqns (5.1) and (5.2) illustrate the dynamic power consumed by a non-overclocked system (P_{no}), operating at voltage V_{no} and that of an overclocked one (P_{ov}) operating at voltage V_{ov}. Here, α and C are switching activity factor and circuit capacitance respectively.

$$P_{no} = \alpha.C.V_{no}^2/t_{no} \tag{5.1}$$

$$P_{ov} = \alpha.C.V_{ov}^2/t_{ov} = \alpha.C.V_{ov}^2.q/t_{no} \tag{5.2}$$

The above model is quite simplistic and does not account for the memory and timing error tradeoffs. Moreover, power is a naive metric for analysis when it comes to handheld devices. Rather, what is important is the impact of power. Power has two major impacts viz., on-chip temperature and battery lifetime. Considering this we choose (1) power-delay-product or the

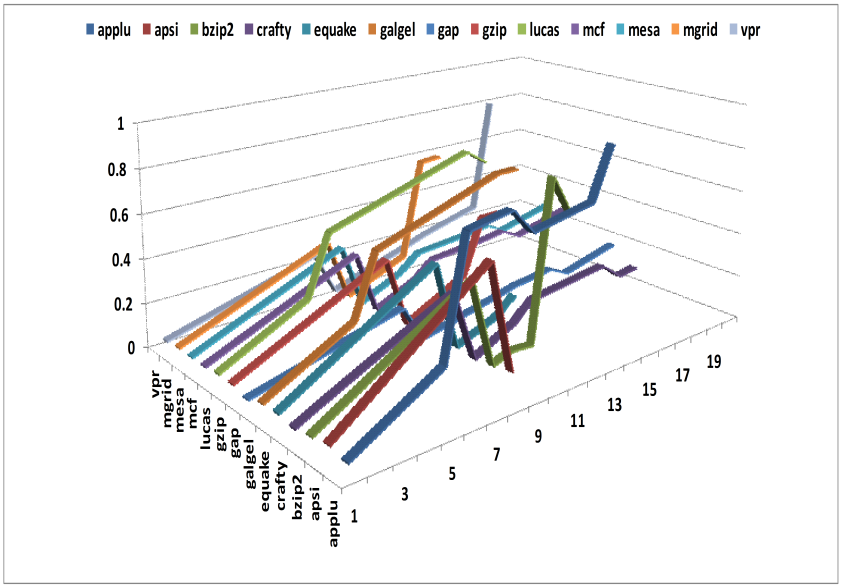


Figure 5.3 Thread activity trace for SPEC 2000 workloads running at $3.4GHz$

energy and (2) maximum and average temperatures for our study.

5.1.3 Energy Dissipation

The energy for the overclocked system is given by E_{ov}, as shown in Eqn 5.3. Again, by proper substitution we find E_{no}, the energy for a non-overclocked system.

$$E_{ov} = P_{ov}.n.(1 + \alpha.q.C_m + S_e.k).t_{ov} \tag{5.3}$$

Upon simplification, we get the following upper bound for energy savings.

$$P_{ov} < (\frac{q \times (1 + \alpha.C_m)}{1 + \alpha.q.C_m + S_e.k}).P_{no} \tag{5.4}$$

5.2 History Based Profile Prediction

The feedback loop in DVARFS is a simple cycle that regulates voltage up or down by a step for every sampling interval. One of the drawbacks in such a scheme is that the system

takes time to converge on a power budgets. This results in loss of efficiency. The second limitation is that the processor tends to get stuck at the lowest voltage level, and continue to work at a higher clock rate (with highest allowed error occurrences). This may not turn out to be the optimal selection of a $V - f$ pair. Choosing an optimal energy-aware power level is a hard problem. A reasonable objective is to adjust the $V - f$ pairs according to the workloads. In other words, it is wasteful to run the processor at higher frequency levels while executing memory bound workloads.

We implement a simple prediction scheme for estimating the workload activity during the next sampling interval. We adopt a technique similar to those in the literature. The basic idea is to track the processor pipeline activity during the sampling interval. The window size, to select the number of sampling intervals to be tracked down is decided depending upon which one helps to predict the activity accurately.

5.3 Evaluation

Table 5.2 shows comparative study results of power budgeting between DVFS ($dvfs$) and DVARFS ($dvarfs$). We used a random mix of SPEC integer and floating point benchmarks to create different number workloads for each run. Specifically, we tried three set of workload numbers, 32, 64 and 128. As mentioned earlier we used history based profiling for different window sizes (1,2,5 and 8). The window sizes refer to the number of sampling intervals used to predict activity during the next interval. Although we tried the experiment for different window sizes, we did not find a specific trend in any of the metric.

We set the power budget at 25 Watts. It is clearly observed that DVARFS provides a finer grained power supply compared to DVFS. This is due to the fact that DVARFS allows dynamic pairing of voltage and frequency levels. In spite of keeping up with the power constraints, DVARFS is able to run the processor at a higher frequency than DVFS due to reliable and aggressive frequency scaling. In general DVARFS runs at 2.71 GHz compared to DVFS running at 2.5 GHz. Energy and ED^2 trends are similar to the one observed in Chapter 3. DVARFS generally outperforms DVFS in both these metrics as well. This is obvious after looking at

the lower power dissipation and higher operating frequency from DVARFS. A very important thing to be noted here is that the number of timing errors is controlled within a predefined budget. In this case we assume the timing error set point to be 5%.

Table 5.2 Comparative study results of power budgeting between DVFS and DVARFS

WINDOW=1	WORKLOADS=32		WORKLOADS=64		WORKLOADS=128	
	$dvfs$	$dvarfs$	$dvfs$	$dvarfs$	$dvfs$	$dvarfs$
Avg Freq (GHz)	2.52	2.71	2.50	2.71	2.51	2.72
Power (W)	22.73	20.20	21.74	19.93	22.21	20.27
Avg Activity	0.63	0.50	0.54	0.50	0.55	0.51
Energy (J)($\times 10^{-3}$)	0.54	0.52	1.12	0.98	2.38	2.02
ED^2 (Js^2)	1.03e-10	0.003e-10	1.07e-10	0.02e-10	1.34e-10	0.23e-10

WINDOW=2	WORKLOADS=32		WORKLOADS=64		WORKLOADS=128	
	$dvfs$	$dvarfs$	$dvfs$	$dvarfs$	$dvfs$	$dvarfs$
Avg Freq (GHz)	2.51	2.70	2.50	2.71	2.50	2.72
Power (W)	22.21	20.04	21.66	20.14	21.81	20.26
Avg Activity	0.34	0.59	0.44	0.50	0.43	0.79
Energy (J)($\times 10^{-3}$)	0.62	0.57	1.23	1.14	2.47	2.097
ED^2 (Js^2)	1.35e-10	0.23e-10	1.39e-10	0.27e-10	1.71e-10	0.49e-10

WINDOW=5	WORKLOADS=32		WORKLOADS=64		WORKLOADS=128	
	$dvfs$	$dvarfs$	$dvfs$	$dvarfs$	$dvfs$	$dvarfs$
Avg Freq (GHz)	2.51	2.74	2.50	2.73	2.50	2.73
Power (W)	22.14	20.80	21.91	20.37	21.86	20.48
Avg Activity	0.24	0.44	0.23	0.46	0.24	0.30
Energy (J)($\times 10^{-3}$)	0.54	0.52	1.17	1.02	2.44	2.12
ED^2 (Js^2)	1.71e-10	0.50e-10	1.74e-10	0.52e-10	2.05e-10	0.75e-10

WINDOW=8	WORKLOADS=32		WORKLOADS=64		WORKLOADS=128	
	$dvfs$	$dvarfs$	$dvfs$	$dvarfs$	$dvfs$	$dvarfs$
Avg Freq (GHz)	2.50	2.74	2.50	2.69	2.51	2.73
Power (W)	21.91	20.69	22.06	19.74	22.20	20.43
Avg Activity	0.15	0.47	0.18	0.17	0.15	0.24
Energy (J)($\times 10^{-3}$)	0.63	0.48	1.27	1.09	2.50	2.15
ED^2 (Js^2)	2.05e-10	0.75e-10	2.10e-10	0.79e-10	2.42e-10	1.03e-10

5.4 Summary

In this chapter, we tried to bring out the importance of computation bounded threads on processor performance. In the Chapter 3, we showcased temperature constrained aggressive

microprocessor systems. Power, as a metric has earned its name to be a first class constraint in today's microprocessors. In this chapter, we focused on microprocessors running on constrained power budget. The goal of our investigation is to ensure if DVARFS can work under power constrained environment. We recognized that for the same power budget, there are more than one voltage-frequency pairs. It is up to the power management algorithm to choose the best one with power and performance in mind. In our next chapter, we address this issue from a wider perspective, in chip multiprocessors.

CHAPTER 6. UTILIZATION BASED TASK SCHEDULING IN CHIP MULTIPROCESSORS

As a result of remarkable evolution of process technology, the idea of placing more than one processing core on a chip is not farfetched anymore. With Chip Multiprocessors (CMPs) already available in market, the road maps predict hundreds of cores in the next decade [91, 55], essentially shrinking today's data centers to a single chip. This implies that the power and thermal management will be of the utmost importance in such systems [92]. Current power management techniques use DVFS for on-line power and thermal management. It has been shown that independent per-core DVFS combined with thread migration improves performance up to 2.6X over a per-core gating [47]. Several variants of DVFS extension to CMPs have been developed [93, 94, 95, 96, 52]. Nevertheless, as Chapter 3 emphasizes, the effectiveness of DVFS is hampered by slow voltage transitions. Products from the leading microprocessor vendors, such as Intel and AMD, have monitoring techniques that take necessary corrective actions to maintain power budget and on-chip temperature. Industry standards support a set of predefined power levels, and allow software applications to choose an appropriate level based on environmental conditions and workload. The software can precisely control the dynamic power of the system to achieve the desired power/energy set point. In spite of all these advancements, power management techniques based on DVFS suffers from circuit slow down and voltage transitions.

In this chapter, our main aim is to showcase the following:

1. *Utilization as a metric for Energy Efficiency :* Not all the power supplied is transformed into useful work! We present an analysis on inefficient power management using current workloads due to excess power supply. We bring out utilization as a factor for choosing

a power level, from a different perspective, rather than ad hoc division of total power budget based on core utilization.

2. *Utilization-aware Task Scheduling (UTS) :* We develop an energy-efficient power management solution for CMPs. In addition to the existing power constraints, we bring in the utilization metric constraint to map the set of threads to the cores and manage the excess power supplied. This model, along with aggressive, reliable framework will perform very differently than any of the existing power management techniques in improving the overall system efficiency provided timing errors are harnessed.

Although it may seem straightforward at first sight to extend DVARFS to a CMP, it is more involved. In CMP, the temperature not only depends on the current core's state, but also on neighborhood temperature. Thus, it requires a careful task scheduling aware of the core states. Moreover, the workloads that run on these cores are diverse. Each workload has different effect in raising the core temperature. Several works have been proposed characterizing tasks according to their workload intensity. However, very few works have been investigated, combining better than worst case approaches with task scheduling, to squeeze out extract maximum performance beyond conservative limits. We also take a step further to bring in dynamic lifetime reliability management by controlling voltage and frequency of the individual cores during run time.

6.1 Task Characteristics and Energy Efficiency

In order to develop an energy-efficient power management technique, it is important to understand the behavior of standard workloads. Different workloads have different patterns of computation, memory and input/output (IO). For illustration, consider Figure 6.1(a) and (b) that shows the application throughput for SPEC benchmark suite [97]. Each pair of points on a vertical level corresponds to one benchmark. The wide variability is shown in the figure across benchmarks and base versus peak. We observe from the figure that the benchmarks have variegated throughput. The throughput is expected to vary if the supply voltage is altered, depending upon the computational density (Recall Figure 5.1). Thus in addition to

the processor power requirements, we also need to consider thread power requirements, which cannot be ignored for energy-efficient processing.

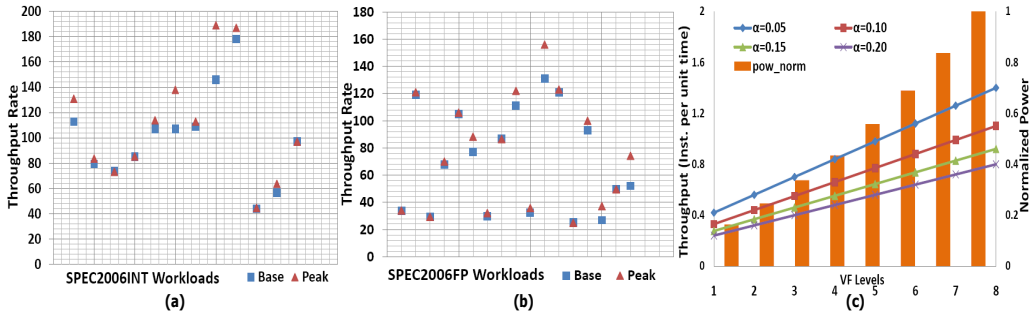


Figure 6.1 Throughput of SPEC benchmark suite on Intel Xeon processor (a) SPECINT (b) SPECFP (c) Throughput of different threads (with different α) and the dynamic power supplied at different power levels

It is a common scenario in the case of CMPs, where each core competes with the other to keep itself at the highest power level. If a core is set to operate at the maximum allowed power level based on meeting the target power budget, then it is likely that a core may be supplied more power than that is required to achieve the needed level of throughput. For instance, a highly memory (or IO) bound thread is not going to perform any better no matter how high the power level is. As current processors had already hit the power wall, it is outrageous to disuse energy that cannot be utilized. Therefore, it is necessary to consider the utilization factor awareness in addition to the existing power constraints when mapping threads to cores and deciding power levels for the cores.

6.1.1 Utilization Aware Power Management

Let us look back at Equation 3.3 to calculate the speed-up at different power levels. To understand the efficiency better, we plotted the throughput of threads with different values of μ and the dynamic power at different power levels. The results (lines) in Figure 6.1 (c) show the throughput values for various classes of workloads (Specifically, for $\mu = 5, 10, 15$ and 20%). For illustration purpose we chose $C_m = 10$ cycles. The vertical bars correspond to normalized power supplied at different power levels. The vertical axes are scaled accordingly

to match the throughput obtained and power supplied. The bars rising beyond the throughput curves illustrate the excess power supplied to the threads beyond the requirement. Although, the system meets the power budget and there is no loss in performance, keeping the cores at higher power set point leads to loss of efficiency. In other words, to increase the overall system efficiency without loss of performance, it is required that each core gets supplied the exact power it will use. In the forthcoming sections we propose our solution to achieve this goal.

6.2 Power Budgeting for Chip Multiprocessors

The power management techniques currently use DVFS for on-line power and thermal management. Traditionally, all the DVFS associated techniques focus on retaining the system at the maximum power level that meets the required power budget. As discussed above, this does not always guarantee higher CPU efficiency. Moreover, these predetermined power states are set at the worst-case level that leads to significant performance loss. To understand the issue of achieving the optimal performance, we briefly present how the state-of-the-art power management problem is formulated [98].

Suppose N threads are to be assigned to a N core system. The goal is to maximize the throughput (Tp) under the given power budget without breaching the voltage constraints. This is modeled as a linear optimization problem, as given below:

Maximize $Tp = \frac{1}{N}\sum_{i=0}^{N-1} tp_i$.

Subjected to:

1. $V_{low} \leq v_i \leq V_{high}$, $\forall i \in 0 \ldots N-1$,
2. $b_i v_i + c_i \leq P_{coremax}$ $\forall i \in 0 \ldots N-1$, and
3. $\sum_{i=0}^{N-1}(b_i v_i + c_i) \leq P_{target}$

Here, tp_i is the individual core throughput, which is modeled as being proportional to the core power level. That is, $tp_i = a_i v_i$ $\forall i \in 0 \ldots N-1$, where a_i is a workload dependent constant. Notice that a_i can be computed only after a task to core assignment has been completed. The existing approach is to use the predicted values based on iterative approach. V_{high} and V_{low} are the upper and lower bounds for individual core voltage levels, P_{target} is the total power

budget, $P_{coremax}$ is the maximum power that an individual core could handle, and b_i and c_i are core dependent parameters.

A multitude of research works similar to the above scheme have been proposed and successfully implemented [93, 94, 95, 96, 52]. Near future technology allows placing most of present day's off-chip components on core, enabling per-core power level adjustments [48]. With such flexibility, the future many-core systems enable ultra-fine grained power management using DVFS at nanosecond scale in contrast to the existing order of tens of microseconds. The existing power management algorithms do not fully make use of this potential. Current techniques simply try to split the total power budget (now, more accurately) across all the cores. They overlook the fact that the total power supplied is not always usefully expended. From the energy efficiency point of view this becomes a bottleneck.

In order to improve the overall system efficiency, it is necessary to redefine the objective of the power management scheme to accurately assign the required power to all the cores, subject to the power budget constraint. The required power by a core depends on various factors, such as workload, core locality and parameter variations.

6.3 Utilization-aware Task Scheduling for the Many-Core Systems

In this section, we propose Utilization-aware Task Scheduling (UTS), an energy-efficient power management solution for the CMPs. We will bring in the efficiency constraint to map the set of threads to the cores in addition to the existing set of power constraints. This model will perform very differently than any of the existing power management techniques. We will also consider the power needs of the thread based on the expected throughput. Our goal is to develop a thread power assignment algorithm, at every sampling interval, under the stated constraints such that the overall system efficiency is improved without bringing out additional overhead. In a many-core system, the temperature not only depends on the current core's state, but also on neighborhood temperature. Thus, it requires a careful task scheduling aware of the core states. Moreover, as the workloads are diverse, each thread has different effect in raising the core temperature. As we had shown earlier, DVARFS control loop is an effective

way to handle thermal emergencies.

We formulate UTS as a linear optimization maximizing problem akin to one explained in Section 6.2. We replace constraint 2 by $b_i v_i + c_i \leq P_{coremax}.a'_i$. Where, a'_i is the predicted workload dependent constant associated with throughput of core i. We use history based profiling techniques to predict the thread activity in the next execution window, similar to ones existing works implement. That is, for every core i, we obtain a_i during each interval. Based on all the a_is obtained over several intervals, a'_i is predicted for the next interval.

Depending on the computation, memory and IO densities of the profiles, the power level is assigned accordingly. While assigning the power level, we use the (V,f) pair assignment, discussed in Chapter 3.

6.4 Simulation Framework

As a part of evaluating our earlier works presented in [56] and [73], we developed a single core simulation framework using sim-outorder - a software functional simulator in SimpleScalar[59], for 64-bit Alpha EV6 processor. We extend this framework to build our CMP environment. The entire framework consists of four major modules, (a),(b),(c) and (d), as illustrated in Figure 6.3. Our experiment involves a combination of both online and offline simulations. A brief description of each of these modules is presented here.

6.4.1 Single Core Simulation

As stated above, we extend our single core evaluation framework to multiple cores. We use the software functional simulator, sim-outorder in SimpleScalar[59]. Table 4.1 provides the baseline configuration for the simulator. Our approach is based on exhaustive profiling of each of the workloads for all possible combinations of voltage-frequency (VF) pairs. This forms the first part of the offline simulations, illustrated by module (c) in Figure 6.3. In addition to the power profile dumped, we also collect relevant performance related profiles from the cycle accurate simulator, to facilitate the task scheduler. To incorporate the effect of voltage transition, we include a penalty as obtained from [99]. The penalty is included at the beginning

of execution during every voltage transition. We also include the leakage power dissipated by the pipeline during the transition period.

For modeling power, we use Wattch[61]. Wattch is an accurate, architecture level power tool that is embedded within sim-outorder of the SimpleScalar simulator. Wattch calculates the energy accumulated over the cycles. We modified the tool to track the instantaneous power for each functional block. As leakage power is becoming a dominant contributor to total power in the nanometer scale designs, we modified Wattch to include this. We establish our experiments with current state of the art by designing our simulations for the $45nm$ and sub-$45nm$ technologies.

6.4.2 Multiple Core Extension

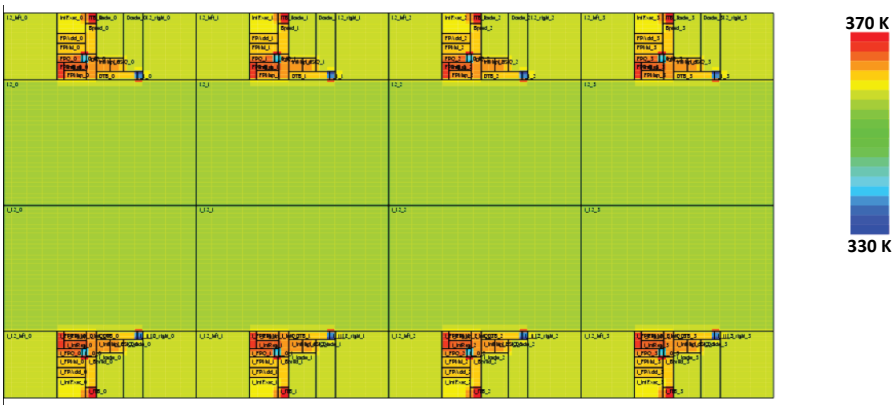


Figure 6.2 Steady state temperature profile for an oct-core processor. The floorplan is Generated by Mirroring and Replicating the Single Core floorplan

6.4.3 Incorporating Timing Errors

In order to bring in the aspects of timing speculation, we model a reliably overclocked processor using timing simulations. We obtain error profiles by running application binary on a hardware model. We record the number of timing errors that occur at a given clock frequency

in the hardware model of a superscalar processor. In that process, we analyze the error rate for the different pipeline stages of a superscalar, dynamically scheduled integer pipeline similar in complexity to the Alpha 21264 [100] that executes a subset of the Alpha instruction set. We use the Illinois Verilog Model (IVM) [101] - a Verilog RTL implementation of the Alpha microprocessor and synthesized individual pipeline stages using the 45nm OSU standard cell library [102]. This becomes the second part of the offline simulations (Figure 6.3 (c)). The errors corresponding to each circuit slowdown is used as the timing error profile for each VF pair.

6.4.4 Power and Timing Error Profile

The power and timing error profiles generated during the offline simulations are stored in a global directory, as key-value pairs, for individual workloads. Part (b) of the figure depicts this. For any given workload, each key-value entry reflects the corresponding processor state during that instant of execution. Thus the entire execution of the workload is preserved in the form of directory listing. These profiles are used by the different instances of cores during actual run.

6.4.5 Incorporating Thermal Model

We use HotSpot (part (d) of the figure), an efficient architecture level thermal modeling tool to calculate temperature[60]. HotSpot requires the CMP floorplan as one of the inputs. It also needs the instantaneous power dissipated for every cycle. Depending on the current core's VF pair assignment, this is provided from the directory corresponding to the executing workload. The total power accumulated during the previous sampling time and the maximum temperature reached by each of the cores is reported to the power monitor, part (a).

6.4.6 Task Scheduler and Power/Thermal Management

Part (a) of the simulation framework illustrates this module. Basically, this is the wrapper module for the remaining part of the framework. The task scheduler includes the task queue,

where each task is assigned individually to the respective cores. Power monitor assigns per core (V, f) levels according to the power/temperature reported during the previous cycle by HotSpot. An added advantage of using task scheduler as a wrapper module is that, we can dynamically issue, preempt or migrates tasks on the fly more conveniently in the former case. Further, the advantage of the profile based simulation compared to explicit multiple instances of single core simulation is several folds. Apart from the better wall-clock time for simulation, the former allows better scalability for multiple cores. The main reason for speed is because of the simpler synchronization procedure across cores. In the case of multiple execution copies (parallel simulations), the cores must be synchronized every cycle to match the corresponding temperature entries with HotSpot. During typical execution, this takes much longer time compared to synchronizing file reads.

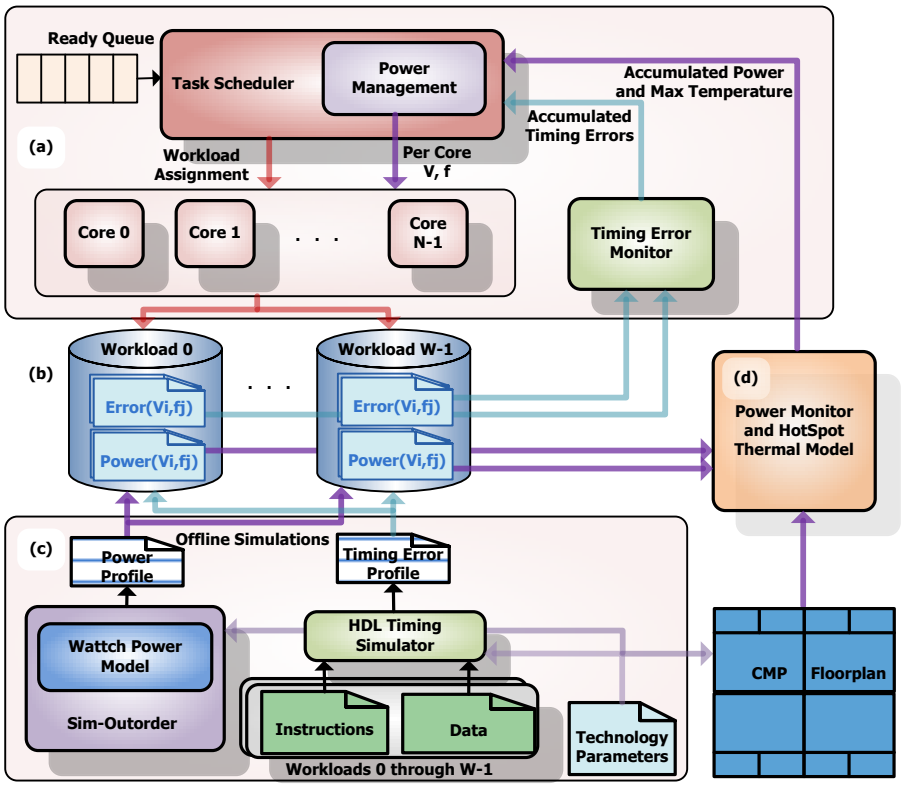


Figure 6.3 Simulation environment for managing power and temperature in aggressive and reliable chip multiprocessor

6.5 Evaluating UTS

We use the simulation framework mentioned above to evaluate UTS. We perform a comparative study against traditional linear optimization scheme. We use Intel Speedstep voltage-frequency pair settings for this mode (*trad*). The linear optimization for power budgeting in *trad* is similar to any of the existing schemes. For this study, we use the method described in [98]. In our next mode, we go a step further by incorporating dynamic (V,f) pairing as in the case of DVARFS ($dvarfs$). Intuitively, we expect a finer grained power management as there are multiple options of choosing a (V,f) level for the same power budget. We compare these two modes against UTS (*uts*). In *uts*, we use multiple frequency levels for any given voltage level similar to $dvarfs$. For our experiment, we use four voltage levels similar to single core scheme, and eight frequency levels per voltage level. In both these cases the system is prone to timing errors due to aggressive frequency scaling beyond worst case limits. We account for the penalties due to these errors as we did in the case of single core evaluation. In this case however, we limit the timing errors under 10% per core.

For our evaluation, we did a number of simulation runs for different settings. We perform our simulations for a number of core settings, viz., 8, 12, 16 and 20 cores. For each of these settings we chose three task queue sizes, viz., 32, 64 and 128. Each simulation setting is represented by (xC:yQ), where x and y are the number of cores and queue length, respectively. All the results presented are normalized relative to *trad* mode.

6.5.1 Activity and Power Dissipation

Figure 6.5 shows a window of activity trace for *trad* and *uts* during the run of 8 core CMP. The x-axis units represent the sampling intervals. As it is clearly seen, the execution window comprises of diverse workload executions in either cases. This is quite evident from the frequent fluctuations of the activity factor from one level to another, typically from 40% to 90%. Lower activity implies higher percentage of memory/IO instructions in the workload (or a memory/IO bound workload), while higher activity implies higher computational density in the workload. From this graph we understand that there is a good mix of workloads from

each type. The second reason to study this graph is to verify that the workload selection has created a level playing field for the different modes. As illustrated in the figure, this seems evident.

The chart illustrated in Figure 6.6 reports the total power supplied to the 8 core CMP at every sampling interval during the execution window corresponding to the activity trace chart. We chose 150W power budget for the CMP and 25W for individual cores. We also provided a 10% tolerance to the supplied power to make sure the linear optimization always converges, and also to allow a margin for possible error during activity prediction for next interval. We start the simulation runs assuming 100% activity for the first prediction interval. This is the reason why both *trad* and *uts* start at the highest power levels. The working of *uts* is clearly reflected from this figure, where we see a lot of fluctuations in the total power supplied. Looking deeper into it, we understand that these fluctuations directly correspond to the activity of the current workload. As mentioned earlier, we use a simple prediction algorithm for estimating the activity of the next interval for each mode. Whenever, the predicted activity is low, the power supplied is reduced correspondingly in *uts*. This is the reason why the total power supply never increases above the budget. In fact, *uts* tends to keep the power budget within the lower end of the 10% margin. However, this is not the case for *trad*, where not only there are very few options of choosing power levels, but also the fact that it tries to divide the maximum power supplied among the cores keeps the overall power constant.

Figure 6.7 shows the average power dissipation across all 8 cores for the three modes, *trad*, $dvarfs$ and *uts*. For the reasons explained above, CMP implementing *trad* tend to dissipate more power. The main point to be noted here is that, even though *trad* seem to consume more power, in reality it is still under the power budget (including tolerance). The significance of aggressive frequency scaling (multiple frequency levels at each voltage level) is evident from the power consumption of $dvarfs$. It should be noted that there is no change in the optimization algorithm or formulation from *trad* to $dvarfs$. The only difference is in the number of available frequency levels per voltage. As it can be observed, $dvarfs$ consumes 15-25% lesser power compared to *trad*.

In *uts*, the power supply to each core is adapted depending on the workload's computational density. It is observed that for some cores, over 45% of power saving is possible using *uts*. It is worthwhile to stress the point that this 45% power goes unused (excess power supplied), which is typically loss of efficiency in the case of *trad*.

6.5.2 Average Power, Performance and ED^2

Figures 6.8, 6.9 and 6.10 illustrates power, performance and ED^2 for different simulation configurations, respectively. The results shown are the average values across all cores. Power dissipation for *uts* in all the cases are better than the other two modes. *uts* saves from 25 to 45% of power relative to traditional power assignment scheme. Maximum power is saved for the 8 core CMP with queue size 32.

In most cases $dvarfs$ saves power, while in few cases *trad* consumes less power. Maximum power is consumed for the 16 core configuration with queue length 32. A general observation is that $dvarfs$ performs better for larger queue lengths. A reasonable explanation for this is that more than half of the selected workloads (6 integer and 7 floating point) are computation bounded. Hence, a larger queue provides much more opportunity for saving power while improving performance through overclocking than a smaller queue. However, this trend is no longer seen in the case of *uts* because, in this scheme aggressive overclocking is limited by the core activity.

Comparing the performance of *trad* with $dvarfs$ and *uts*, we see that there is a significant performance improvement (10%) of the latter modes relative to *trad*. However, there is not much difference in performance between the latter two modes. Maximum performance of around 12% is achieved for 16 core CMP with queue length 64.

ED^2 measures the useful work done in a power constrained high performance system. It is clearly illustrated how $dvarfs$ and *uts* use minimal ED^2 compared to existing methods. Over 50% savings in ED^2 is achieved for the 12 core configuration. In single core simulations we showcased how effective DVARFS scheme works in terms of ED^2. However, the utilization-aware task scheduling has gone a step further by outperforming DVARFS based scheme in

every configuration.

6.6 Summary

Power budgeting is one of the critical activities for efficient functioning of present day's chip multiprocessors. Traditional way of power allocation among cores leads to loss of efficiency due to excess power supplied for non-compute bound workloads. In this chapter, we brought forward the significance of efficiency in terms of core utilization and power supply. We proposed UTS, bringing the additional utilization constraint to the existing power management technique. From our evaluation we inferred that UTS improves performance by up to 12% due to aggressive power level switching. We also inferred that UTS saves over 50% in ED^2 compared to traditional power management techniques.

Based on our research we recommend that future multi-core systems should deploy

1. local fault detection and recovery circuitry to support aggressive, but reliable timing speculation for efficient, on-line power & thermal control, and performance enhancement.

2. a power management scheme that not only assigns power to individual cores based on its utilization, but also curtails the excess power that will be supplied to the cores.

3. utilization aware task scheduling along with aggressive timing speculation in order to squeeze out maximum performance from the system without loss of efficiency and breaching power & thermal constraints.

Results from our evaluation look promising for possible realization of the architecture in hardware prototype. We strongly believe that such a circuit will provide a unified solution for both fault-tolerant and power-aware systems.

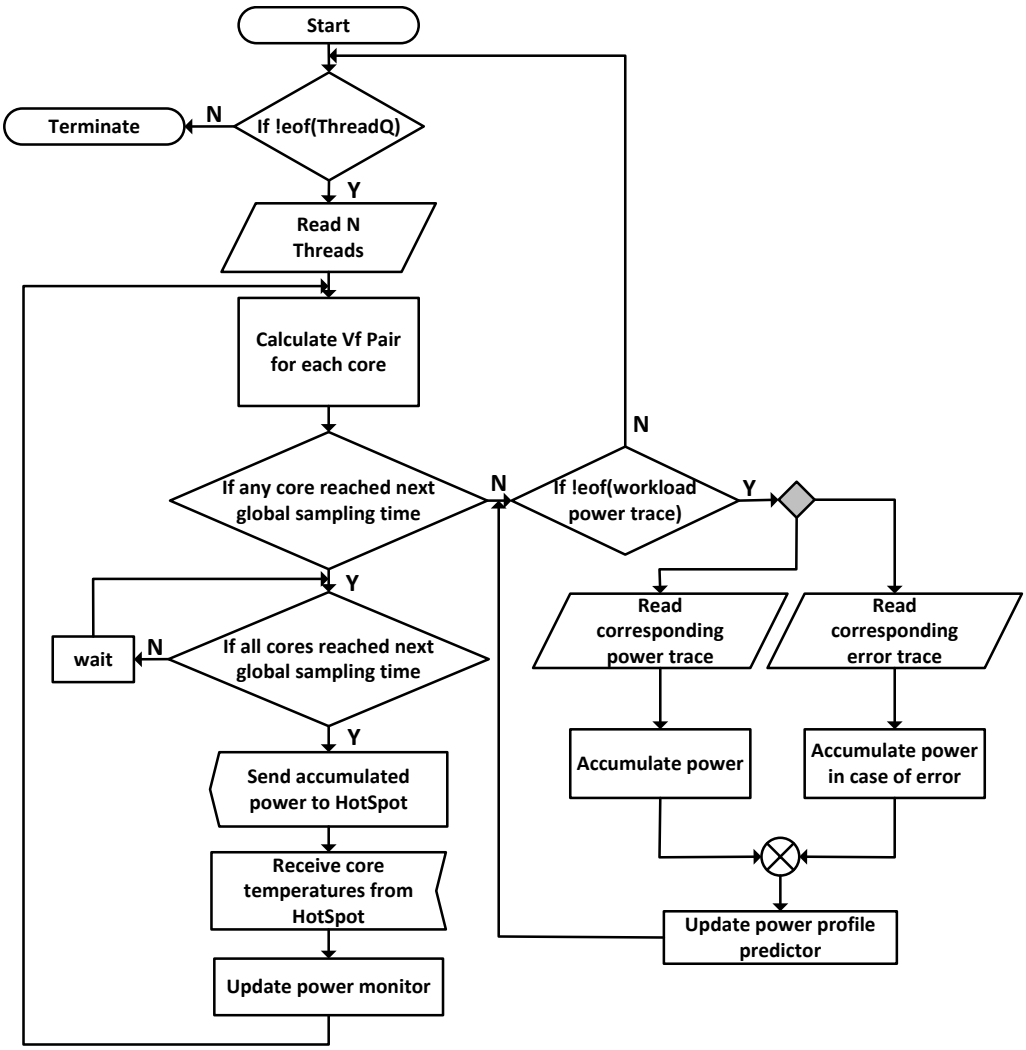


Figure 6.4 Flowchart for the chip multiprocessor simulation framework

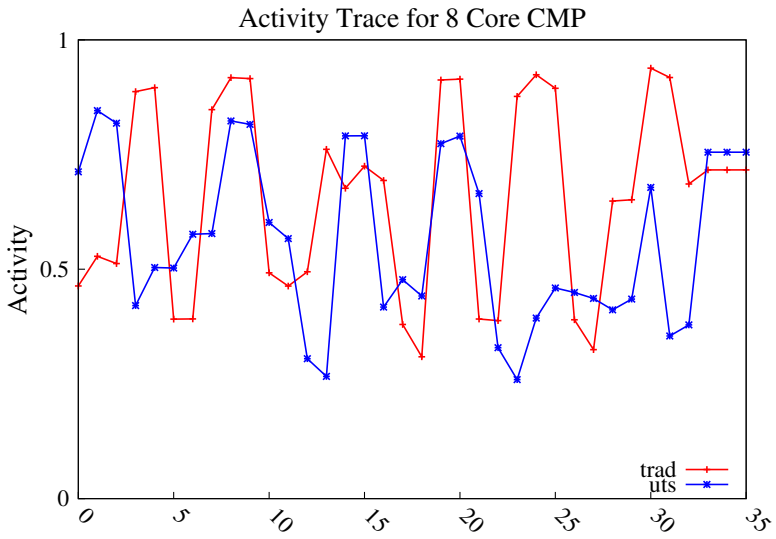


Figure 6.5 Activity trace across sampling intervals for SPEC workloads for 8 Core CMP

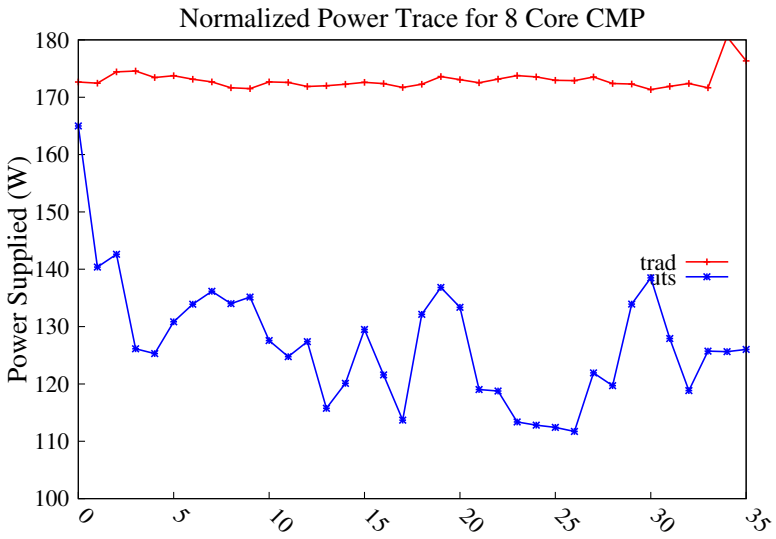


Figure 6.6 Power trace across sampling intervals for SPEC workloads for 8 Core CMP

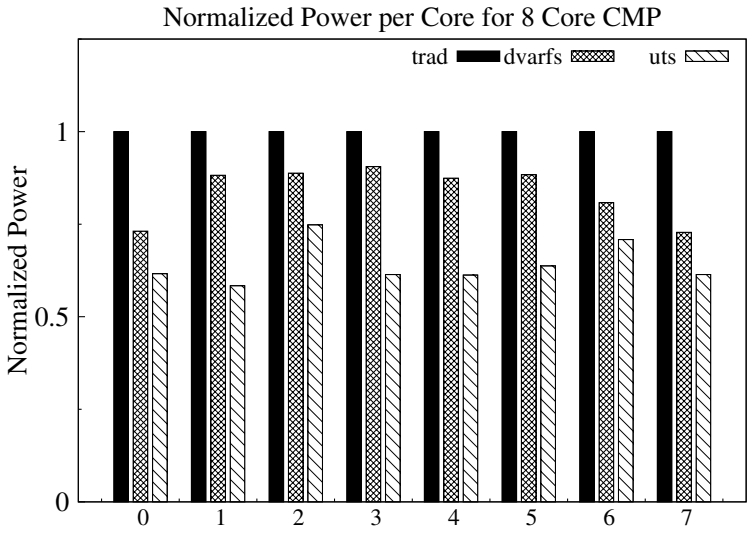


Figure 6.7 Normalized per-core power dissipation for SPEC Workloads for 8 Core CMP, with a 150W power budget and 10% tolerance

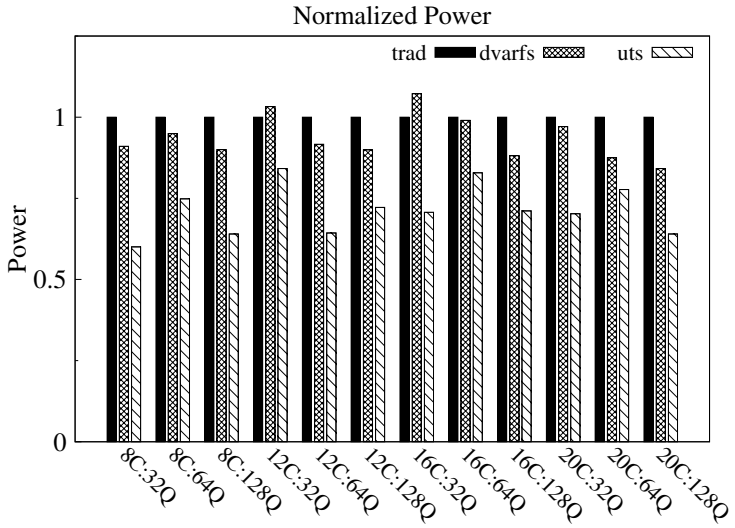


Figure 6.8 Normalized power chart comparing UTS against traditional and DVARFS scheduling schemes for different number of cores and workloads

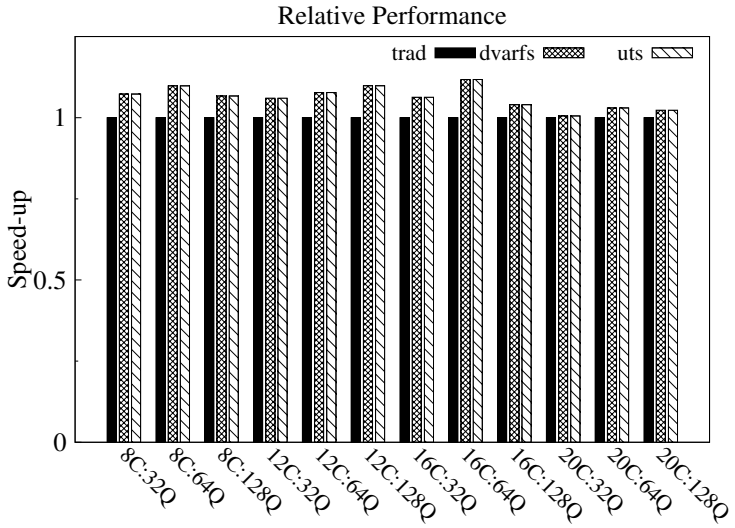


Figure 6.9 Performance chart comparing UTS against traditional and DVARFS scheduling schemes for different number of cores and workloads

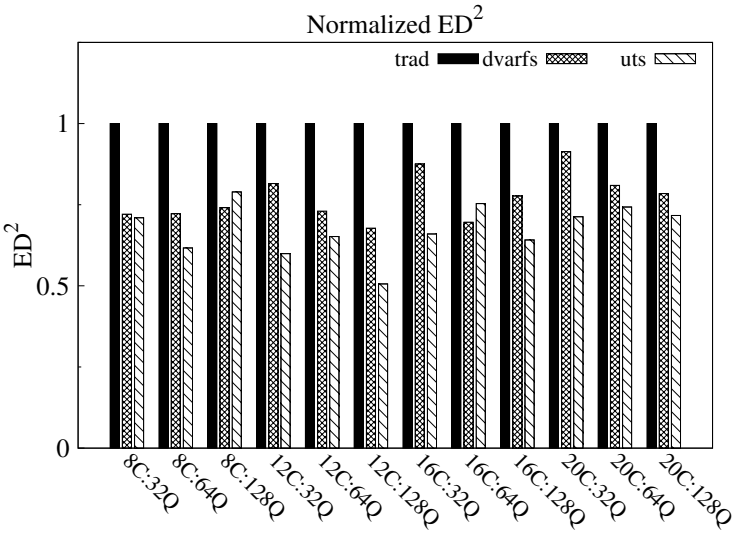


Figure 6.10 ED^2 chart comparing UTS against traditional and DVARFS scheduling schemes for different number of cores and workloads

CHAPTER 7. CONCLUSION AND FUTURE WORKS

Microprocessors are designed to function reliably for the worst-case settings, allowing possible performance improvement by making common cases faster, and creating opportunity to improve processor performance to a greater extent through overclocking. When the system is forced to operate beyond this conservative limit, they also adversely impact on-chip temperatures, leading to hot spots. Overclocking enthusiasts invest heavily in expensive cooling solutions to protect the chip from overheating, and such overclocked systems typically have significantly lower lifetime. Additionally, reliable overclocking techniques necessitate additional circuitry, leading to an increase in power consumption. Higher clock speeds and power densities invariably lead to accretion of on-chip temperature over a period of time. As systems operate faster, on-chip temperatures quickly reach and exceed the safe limits. This poses a serious threat to the lifetime reliability of the systems in the present and near future. The very intention of this dissertation is to overcome the challenges brought forth by recent technological advancements in the field of computer architecture. In particular, to develop software and hardware solutions to overcome the conservative design approaches due to process, voltage, and temperature variations in digital circuits.

In this thesis, we presented an overview of specific limits suffered by reliably overclocked systems. We showcased impact of power on chip temperatures and analyzed its effect on lifetime reliability. By adopting to a typical reliable overclocking framework, we studied the thermal behavior of Alpha processor. We made the case for the need of a powerful thermal management scheme in reliably overclocked circuits. We proposed an efficient technique for performance enhancement and thermal management called the DVARFS scheme, exploring a new direction to manage on-chip thermal conditions to achieve maximal performance benefits.

The DVARFS mechanism facilitates to reliably overclock the processor under thermal bounds at target lifetime with a programmable error rate. We established an extensive simulation framework environment, integrating various tools to perform our simulation studies. Using this framework we have shown that the DVARFS scheme performance in par with existing DVFS scheme despite exceeding the worst-case operating frequency. We achieved vast improvements in performance compared to traditional designs assuming target system lifetime. Our simulation results reveal that controlled reliable overclocking is indeed a beneficial way to enhance performance taking thermal constraints into consideration.

We have shown how short path delay in digital circuits is a major bottleneck for achieving higher performance in timing speculation architectures. After theoretical analysis, we demonstrated how much performance improvement is achievable by increasing the contamination delay of the circuit without affecting the critical path delays. We proposed a solution to increase the short path delays so that they are no longer a constriction for timing speculation. We introduced the Min-Arc algorithm that efficiently places buffers in along the edges of the short-paths. We have shown that the Min-Arc is able to increase the contamination delay of all the circuits without affecting propagation delay. Our algorithm is designed in such a way that even in the case where increasing short path delay is not possible without affecting critical paths, with a slight margin on propagation delay it achieves the intended goal. We observed moderate area increase in the circuits implementing the Min-arc algorithm. In short, it is not an overstatement to say that Min-arc algorithm successfully increases the contamination delay of logic circuits with moderate area penalty.

It has become common for the present day systems to have multicore processors. Allocation of power to each core is a tricky problem. With the way existing power schedulers work, there is still a wide margin for improving system efficiency. In this thesis, we showcased how prioritizing utilization in power management algorithm improves energy efficiency significantly. From our evaluation we inferred that UTS improves performance significantly due to aggressive power level switching. We also inferred that UTS cuts down ED^2 consumption by over half compared to traditional power management techniques. Results from our evaluation look promising for

possible realization of the architecture in hardware prototype. We strongly believe that such a circuit will provide a unified solution for both fault-tolerant and power-aware systems.

As a part of our evaluation process, we built an extensive simulation environment by integrating SimpleScalar - a C based simulator for Alpha EV6 processor, with Wattch power model and integrated thermal model, HotSpot. Timing details are brought from the hardware experiments that are run on a 45nm gate level implementation of the superscalar processor.

To summarize, the goal of this thesis is to develop a system having the potential to provide dynamic knobs to adjust power consumption, performance, energy and temperature. We believe this is a very significant paradigm that will revolutionize the design of multi-core architectures and peta-scale computing system design. This research experience has given many valuable insights into the functionalities of micro-architectures and chip multiprocessors.

This dissertation is an exploration to dynamically managing voltage and frequency beyond the worst-case design specifications. The results we have obtained are very promising, opening up different directions for the near future.

We are continuing this work by implementing our scheme on a hardware platform such as FPGA and tracking temperature on-line through thermal sensors. We are approaching industry to plan a test of our model with an ASIC model. The results we obtained at this juncture are very promising, setting up many different directions for the near future. Hardware realization, like FPGAs or ASICs, will make the case stronger for timing speculation architectures in commercial circuits.

A. EXECUTION TRACES

This appendix illustrates the execution traces of the SPEC INT and FP workloads that we simulated for the comparative study of DVARFS with other modes discussed in Chapter 3. We executed six integer and seven floating point workloads for the analysis. Figures A.1 and A.2 show the voltage trace, Figures A.3 and A.4 show the frequency trace and A.5 and A.6 show the error traces. Temperature and MTTF are illustrated in Figures A.7, A.8 and A.9, A.10, respectively.

Following this, we present the power and energy related metrics from Figures A.11 through A.18. A detailed, zoomed in illustration for each of these parameters and metrics during an execution window is presented in Figures A.19, A.20 and A.21. With all the traces, we were able to understand the working of different techniques for each of the executing threads. We were also able to assert the simulation functionality.

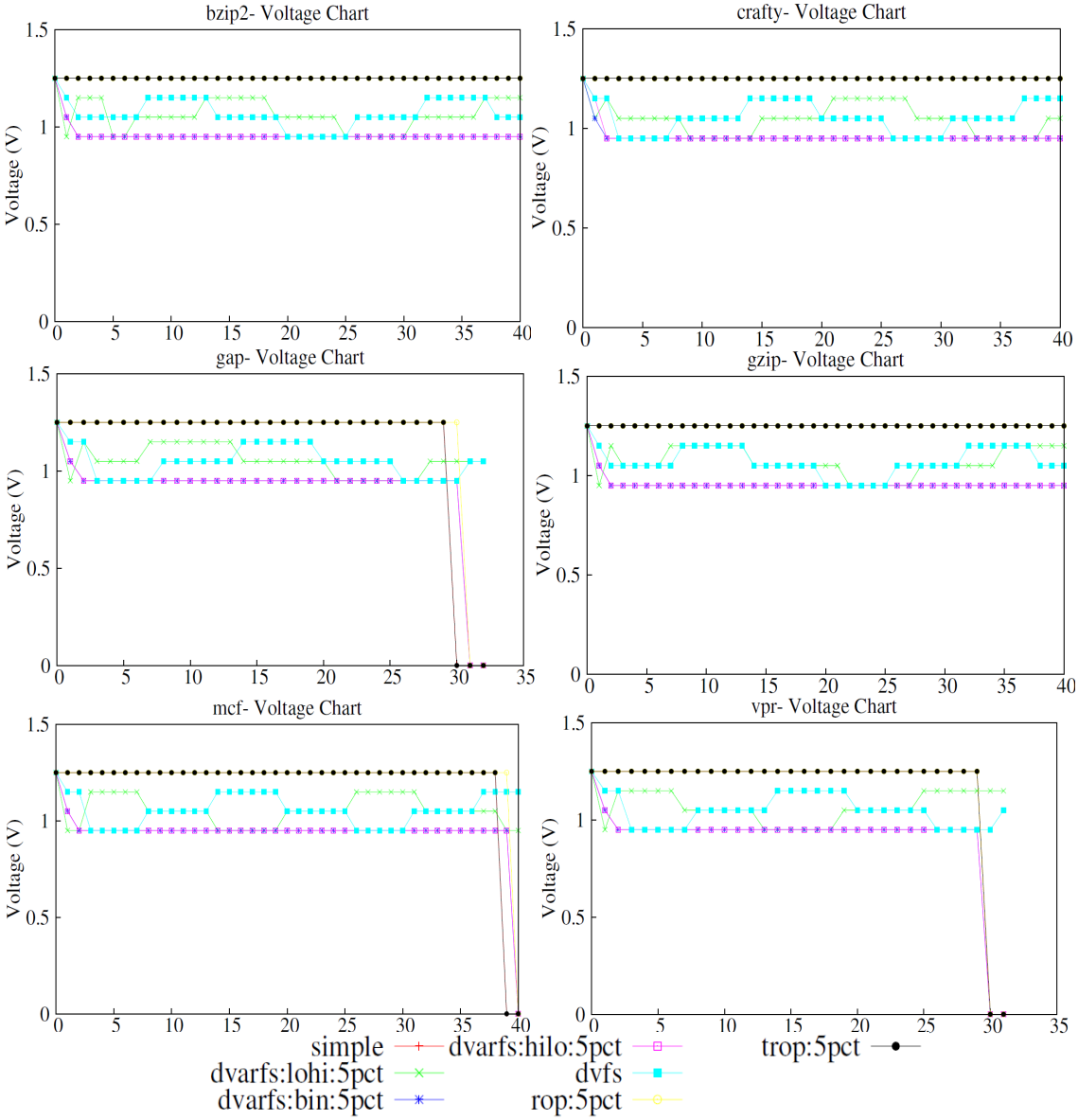


Figure A.1 Voltage trace for SPEC INT workloads

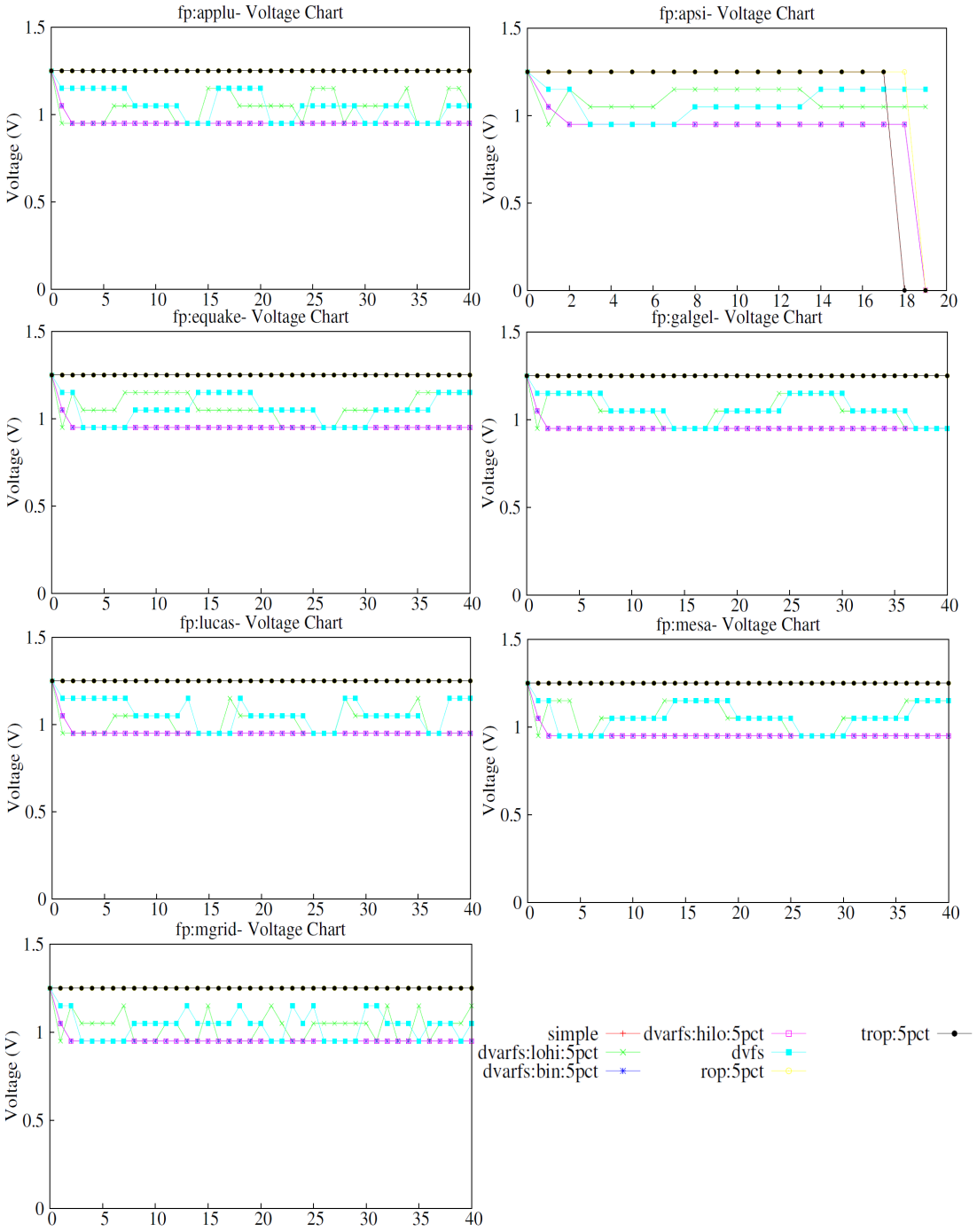


Figure A.2 Voltage trace for SPEC FP workloads

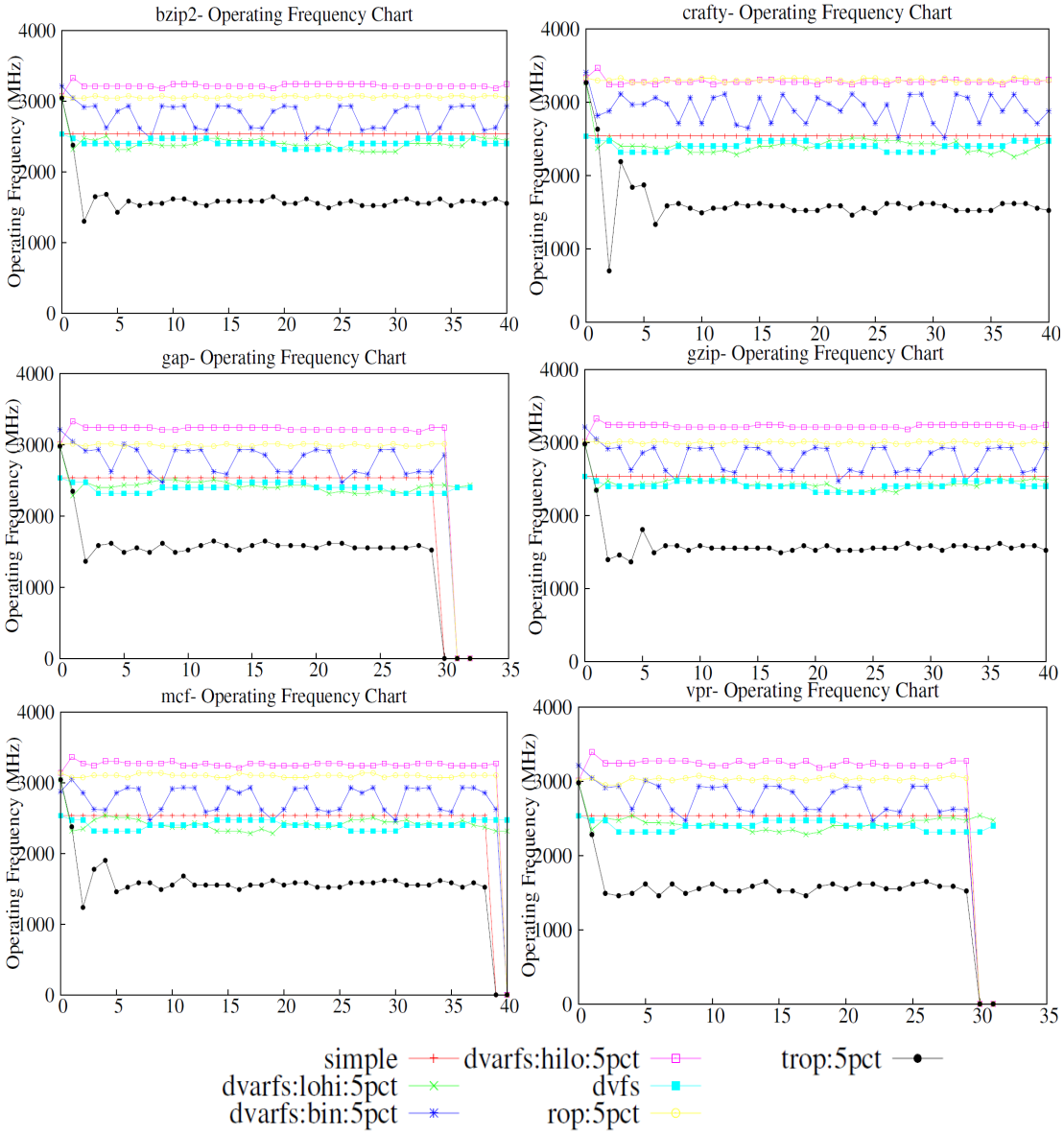


Figure A.3 Frequency trace for SPEC INT workloads

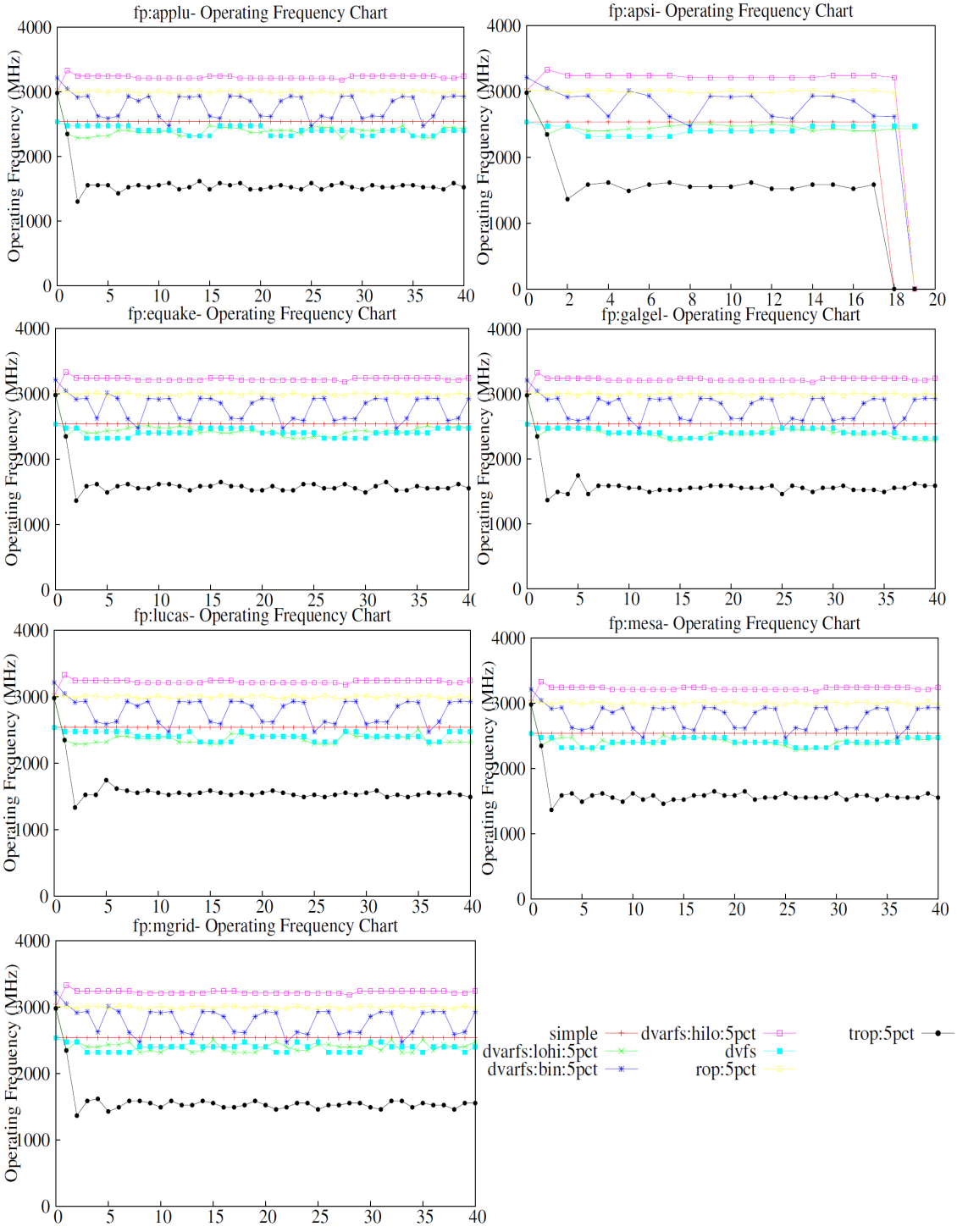


Figure A.4 Frequency trace for SPEC FP workloads

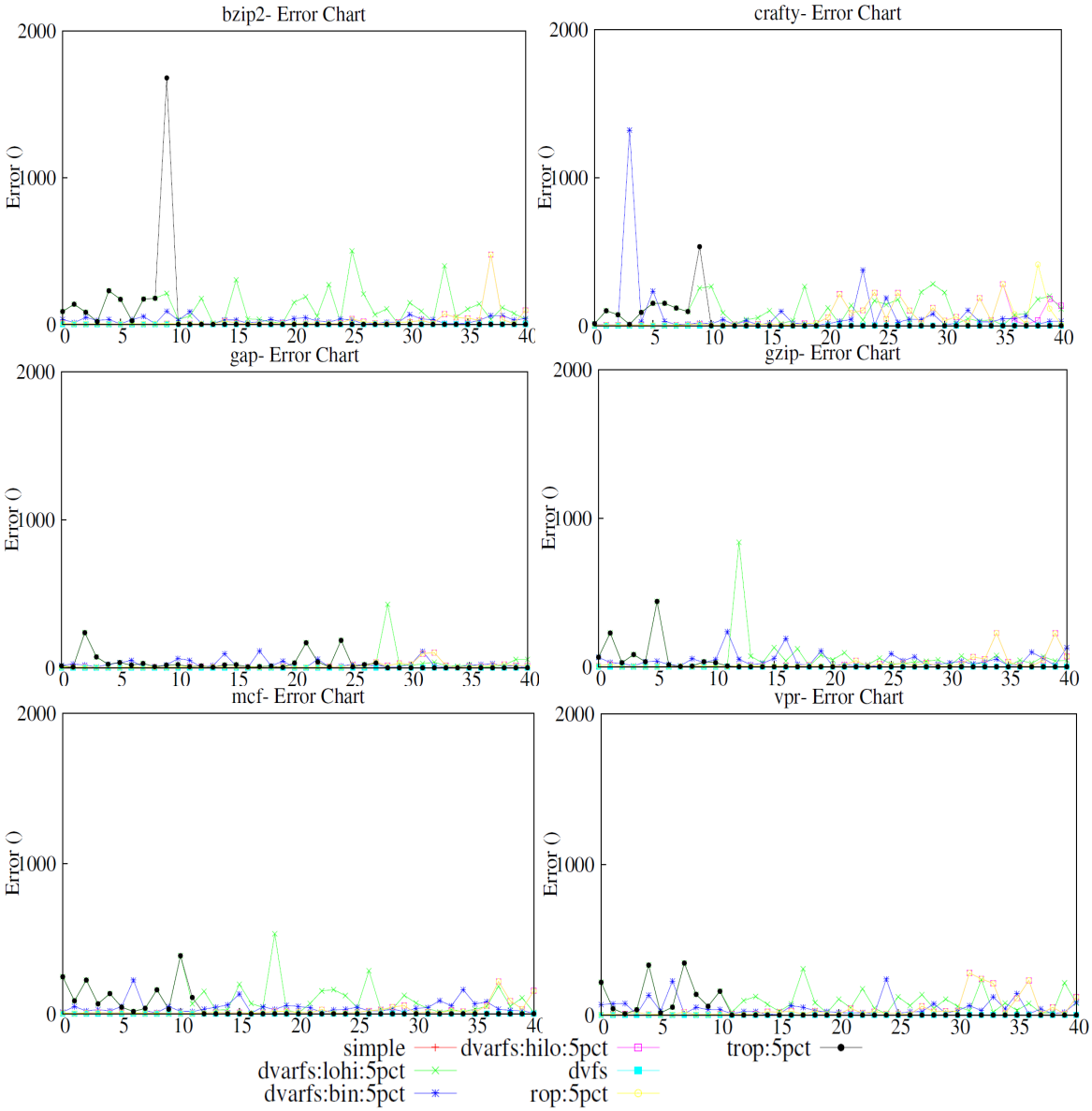


Figure A.5 Error trace for SPEC INT workloads

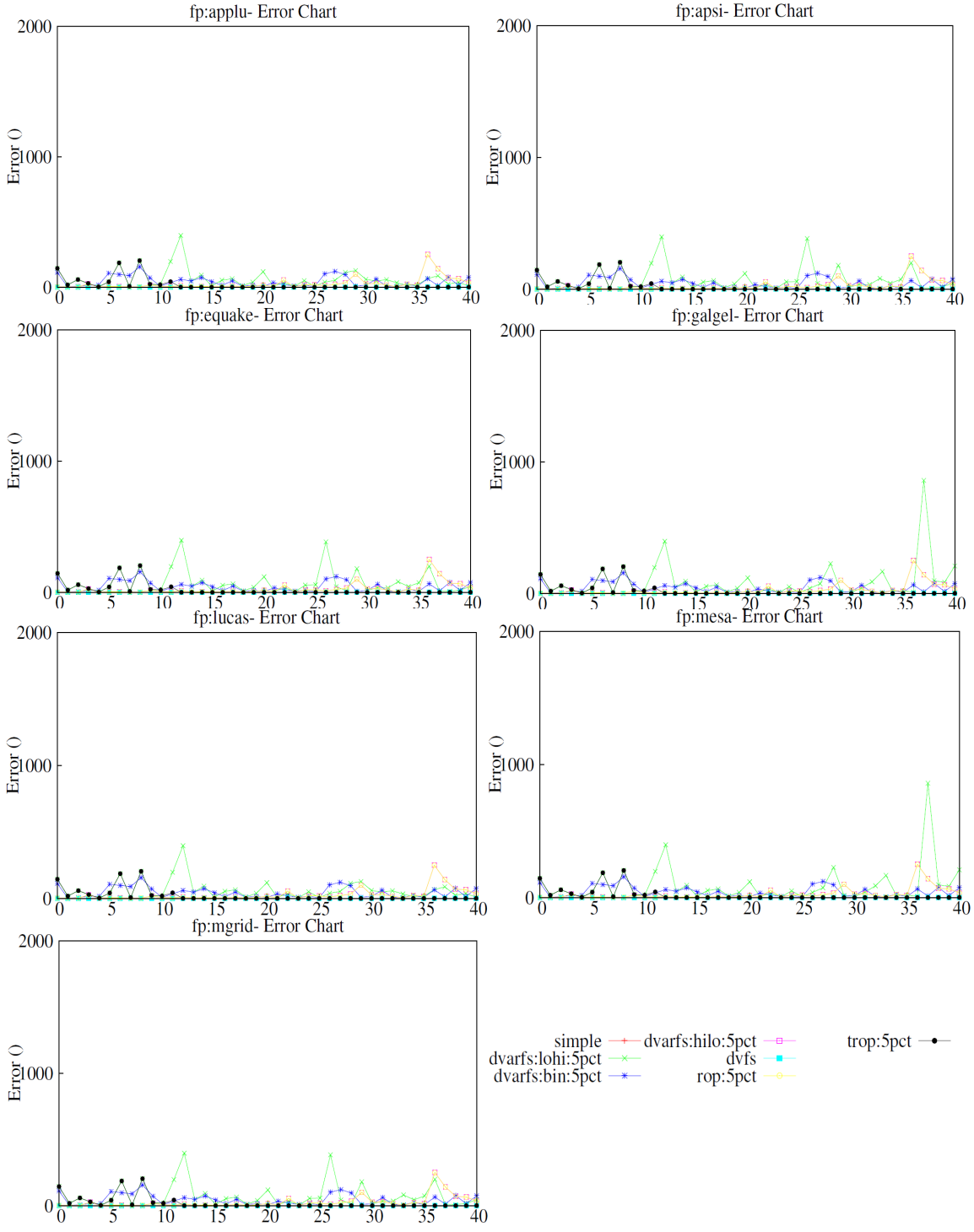


Figure A.6 Error trace for SPEC FP workloads

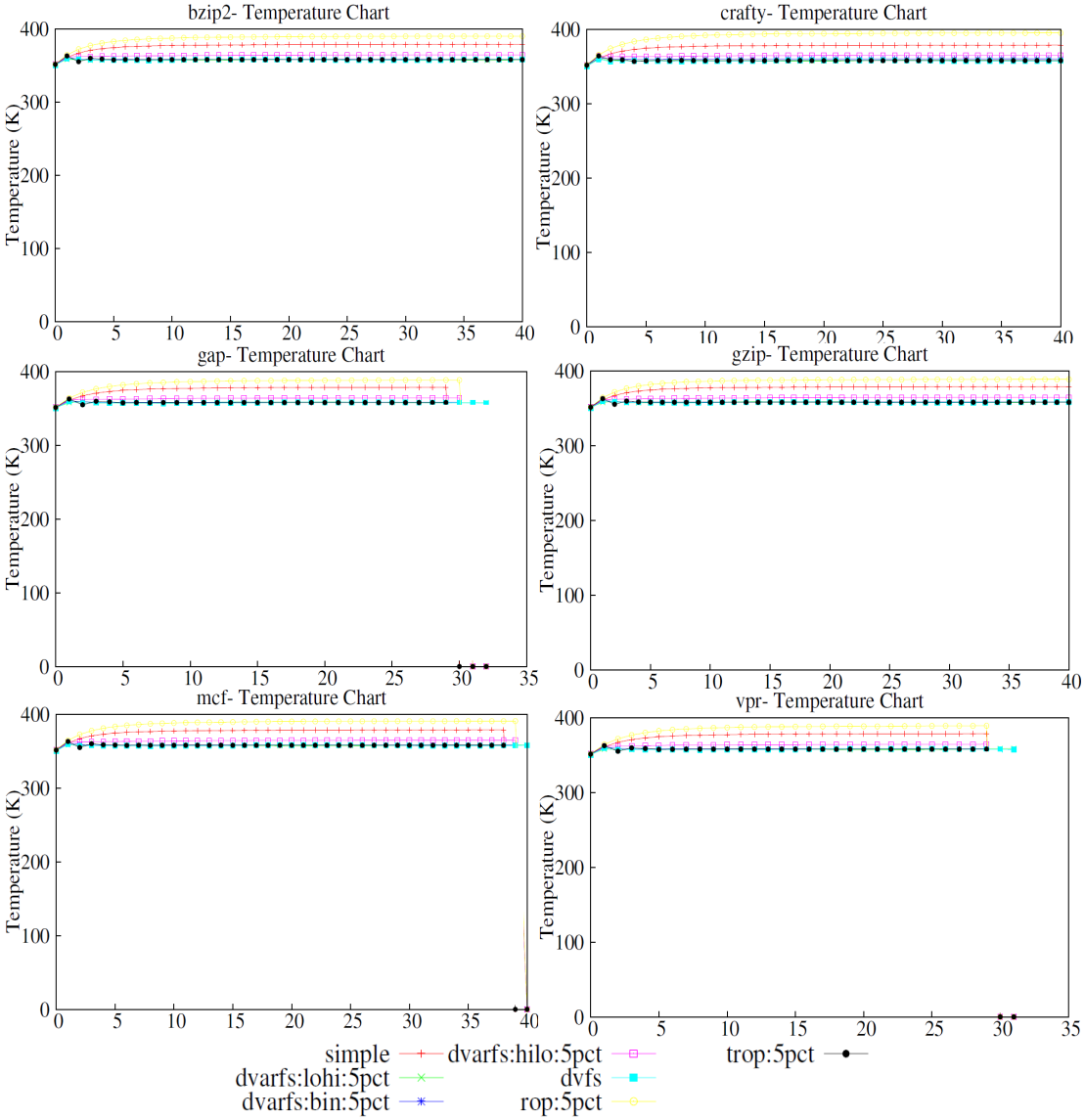


Figure A.7 Temperature trace for SPEC INT workloads

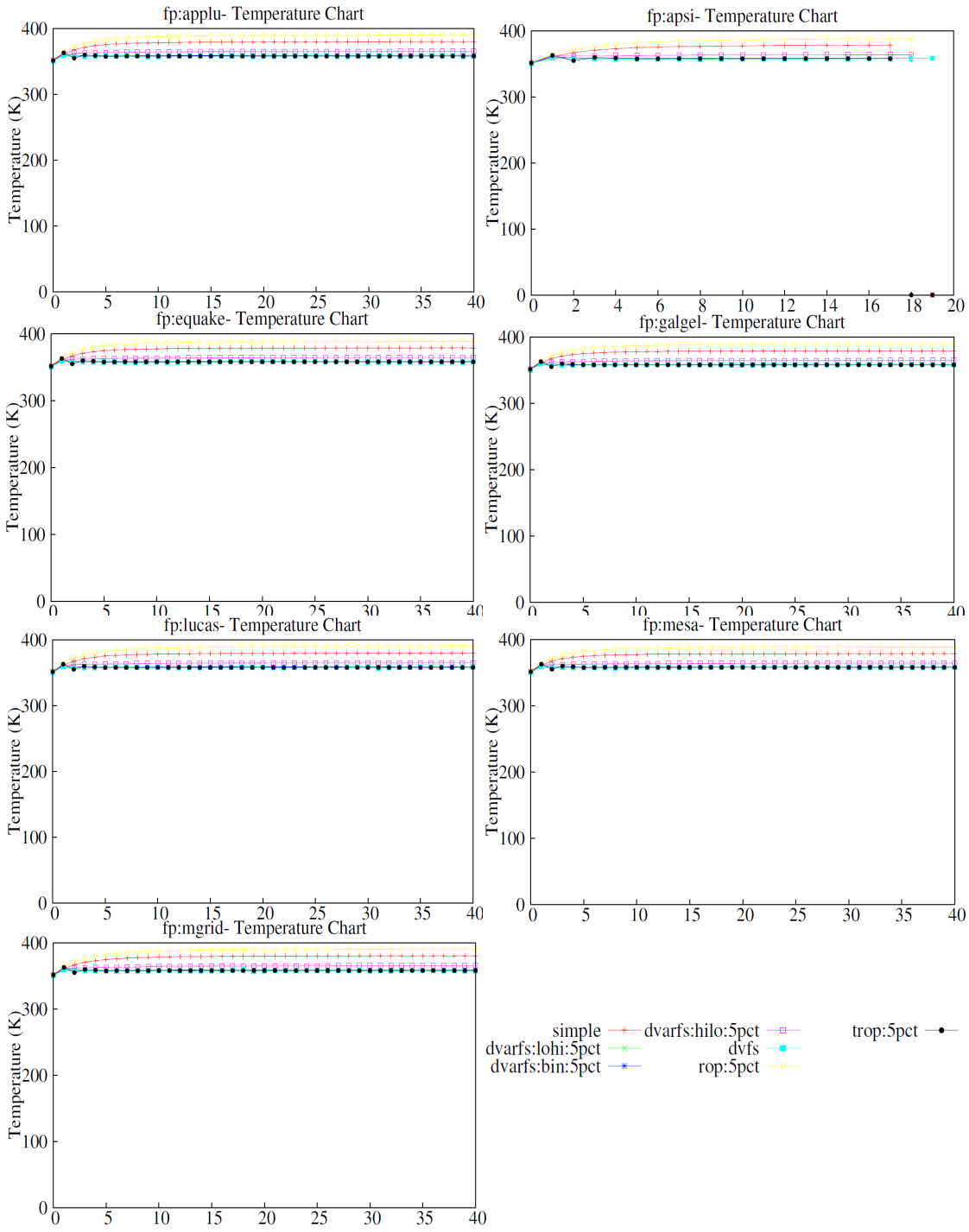


Figure A.8 Temperature trace for SPEC FP workloads

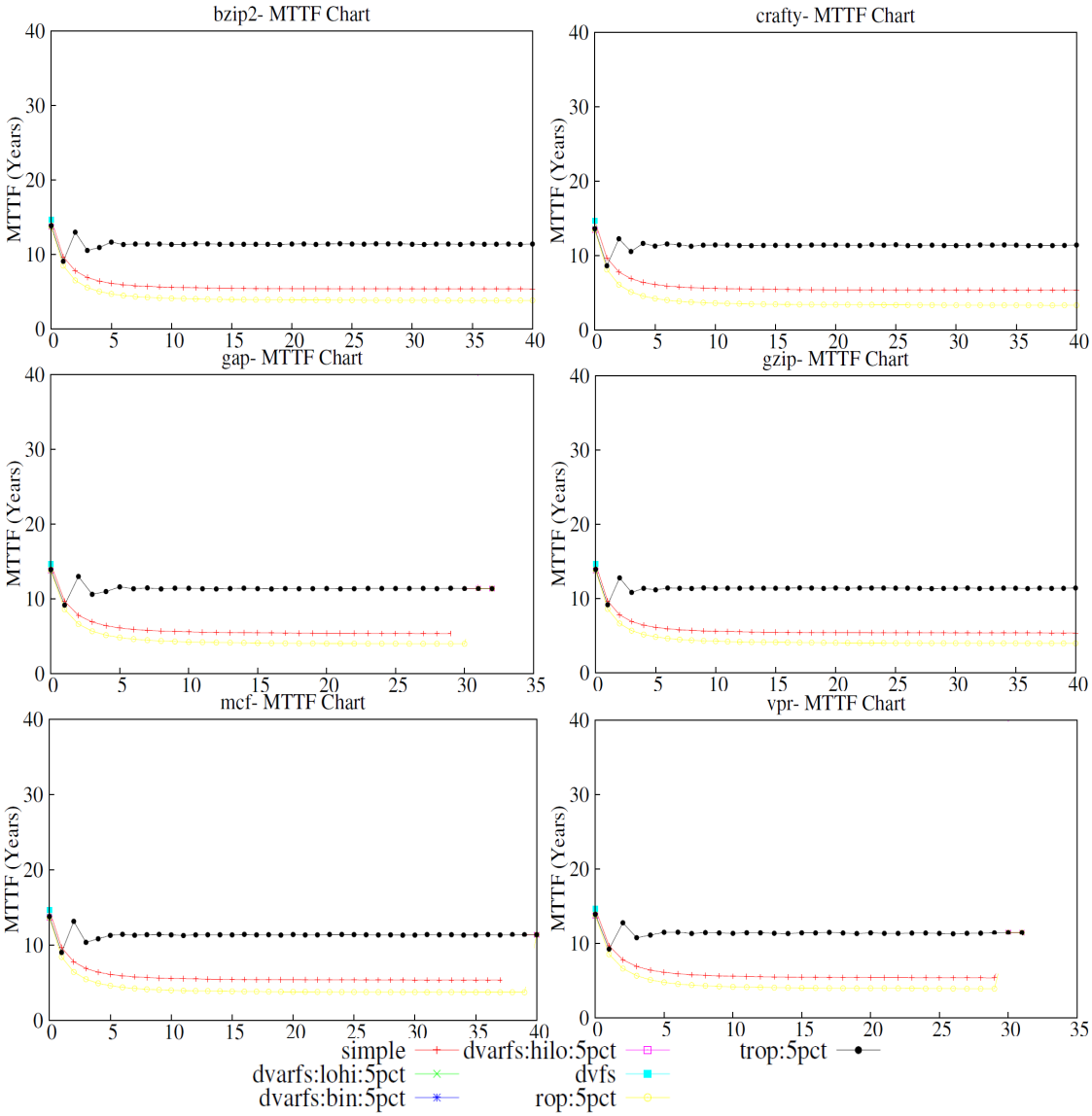


Figure A.9 MTTF trace for SPEC INT workloads

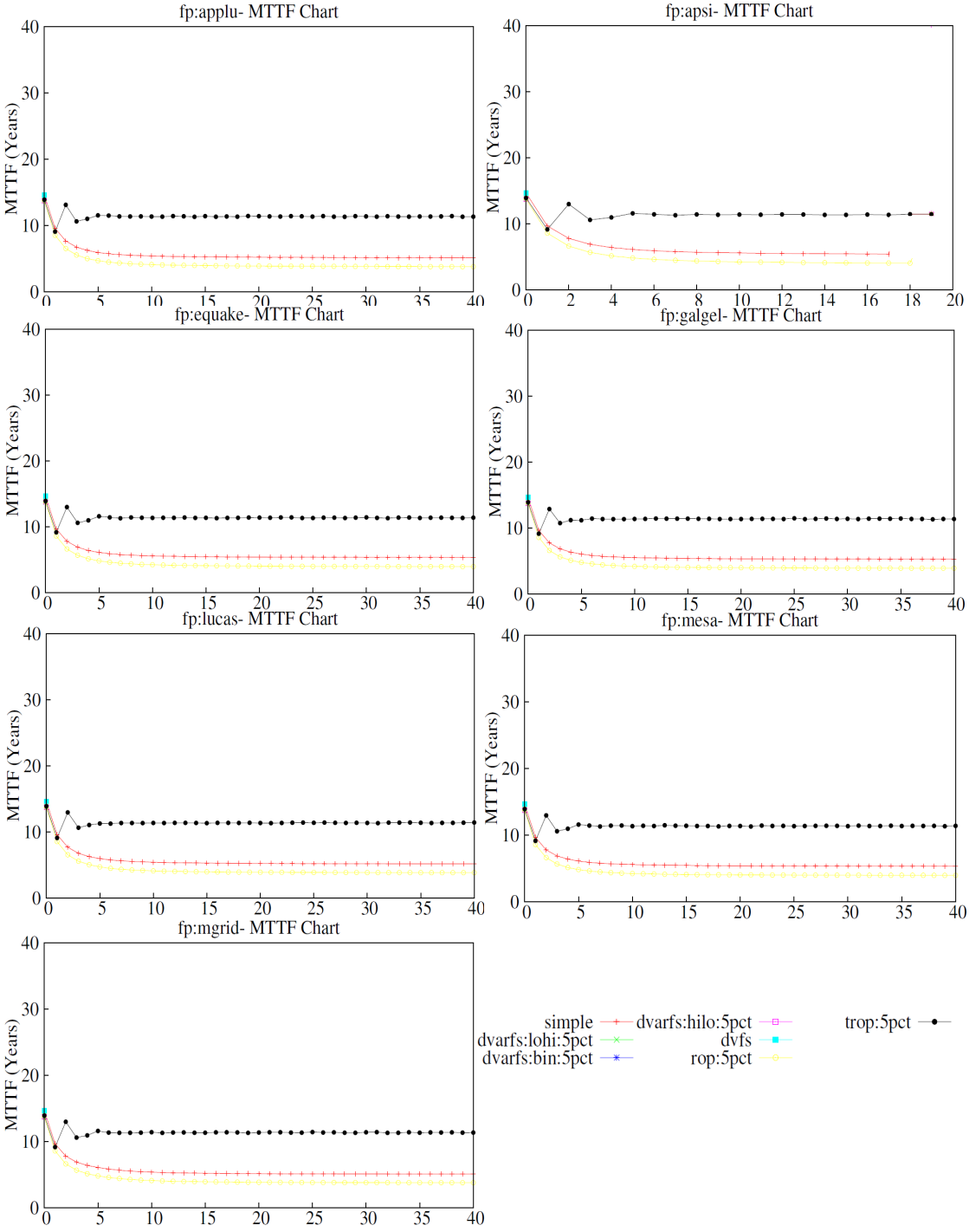


Figure A.10 MTTF trace for SPEC FP workloads

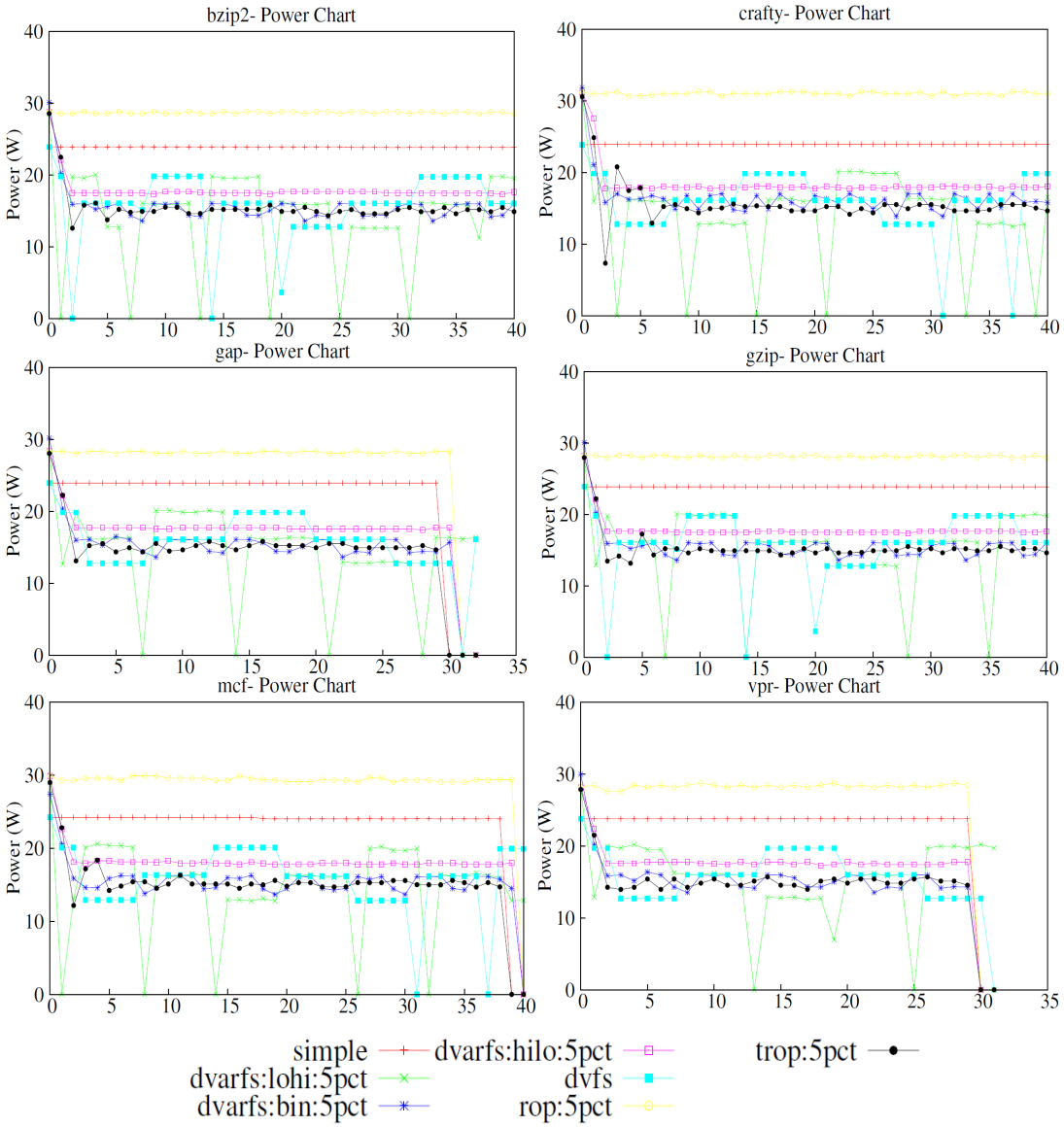


Figure A.11 Power trace for SPEC INT workloads

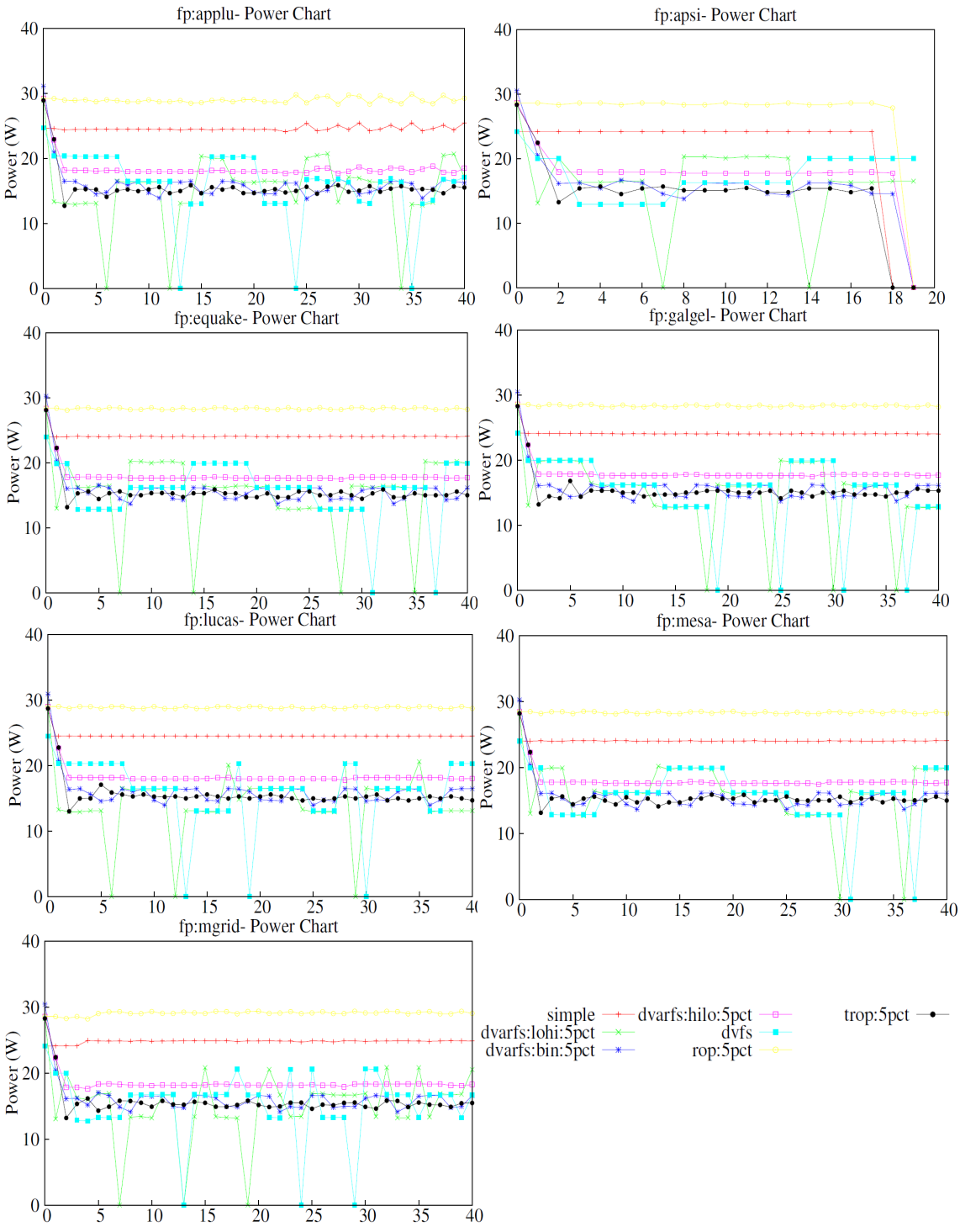


Figure A.12 Power trace for SPEC FP workloads

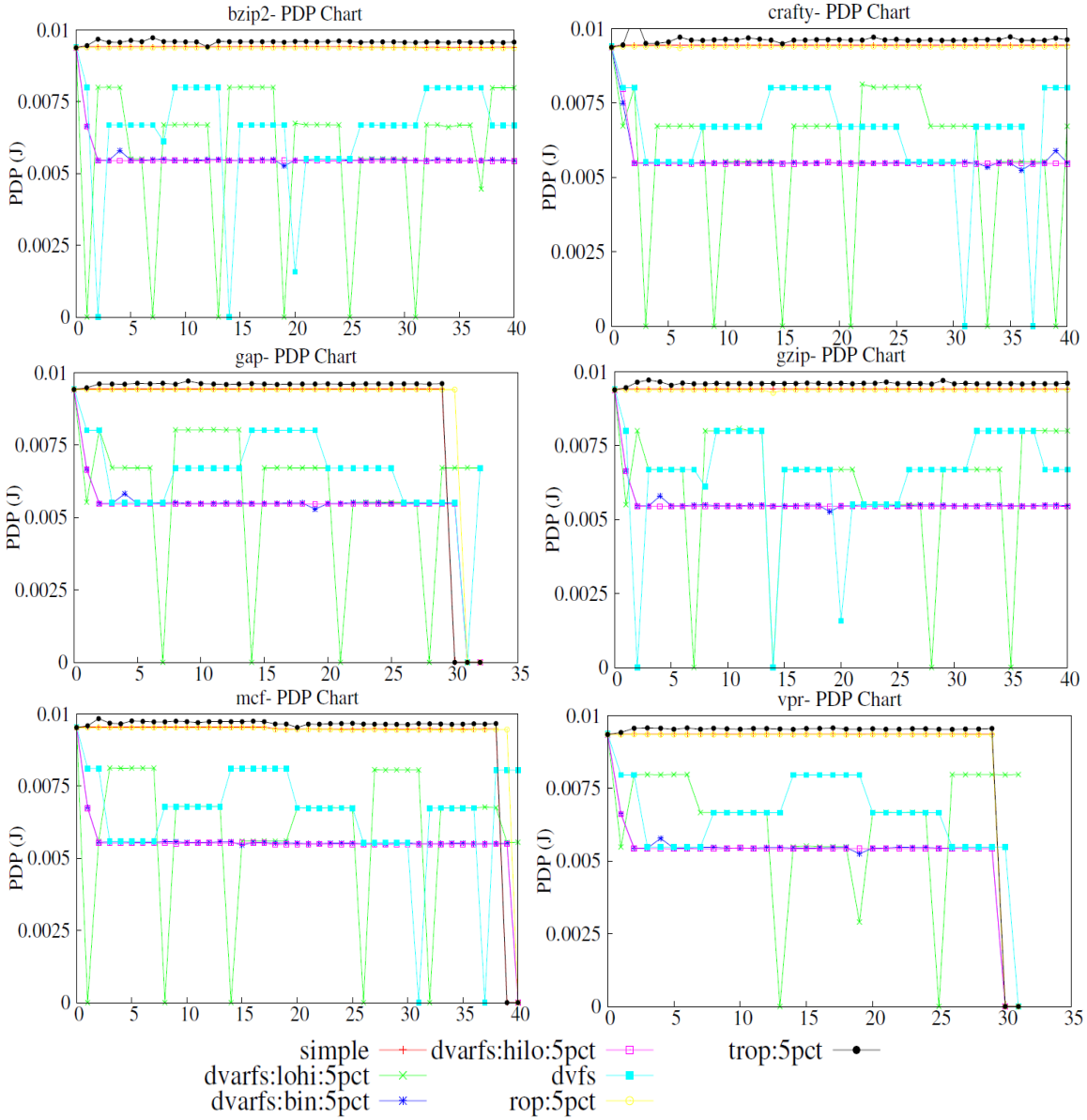


Figure A.13 PDP trace for SPEC INT workloads

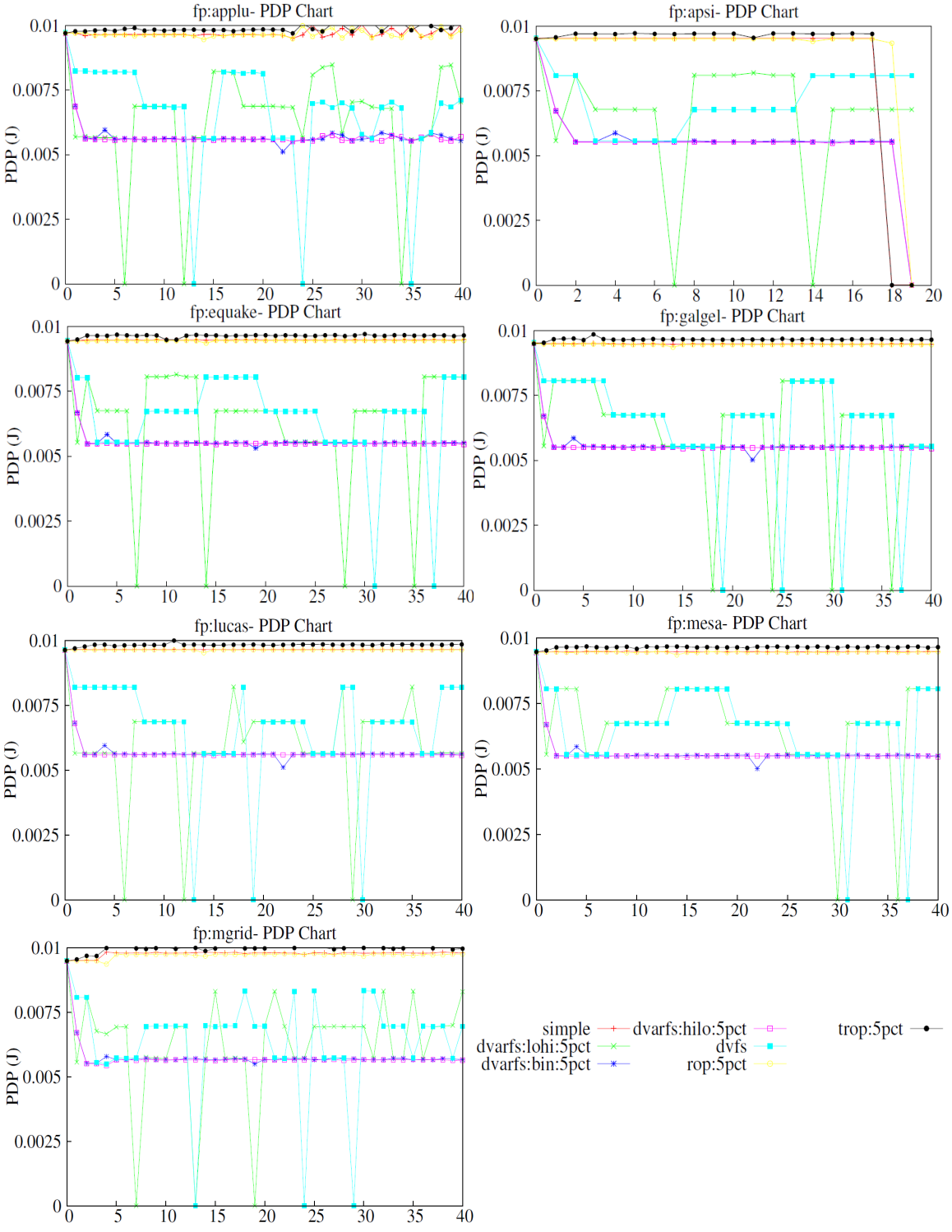


Figure A.14 PDP trace for SPEC FP workloads

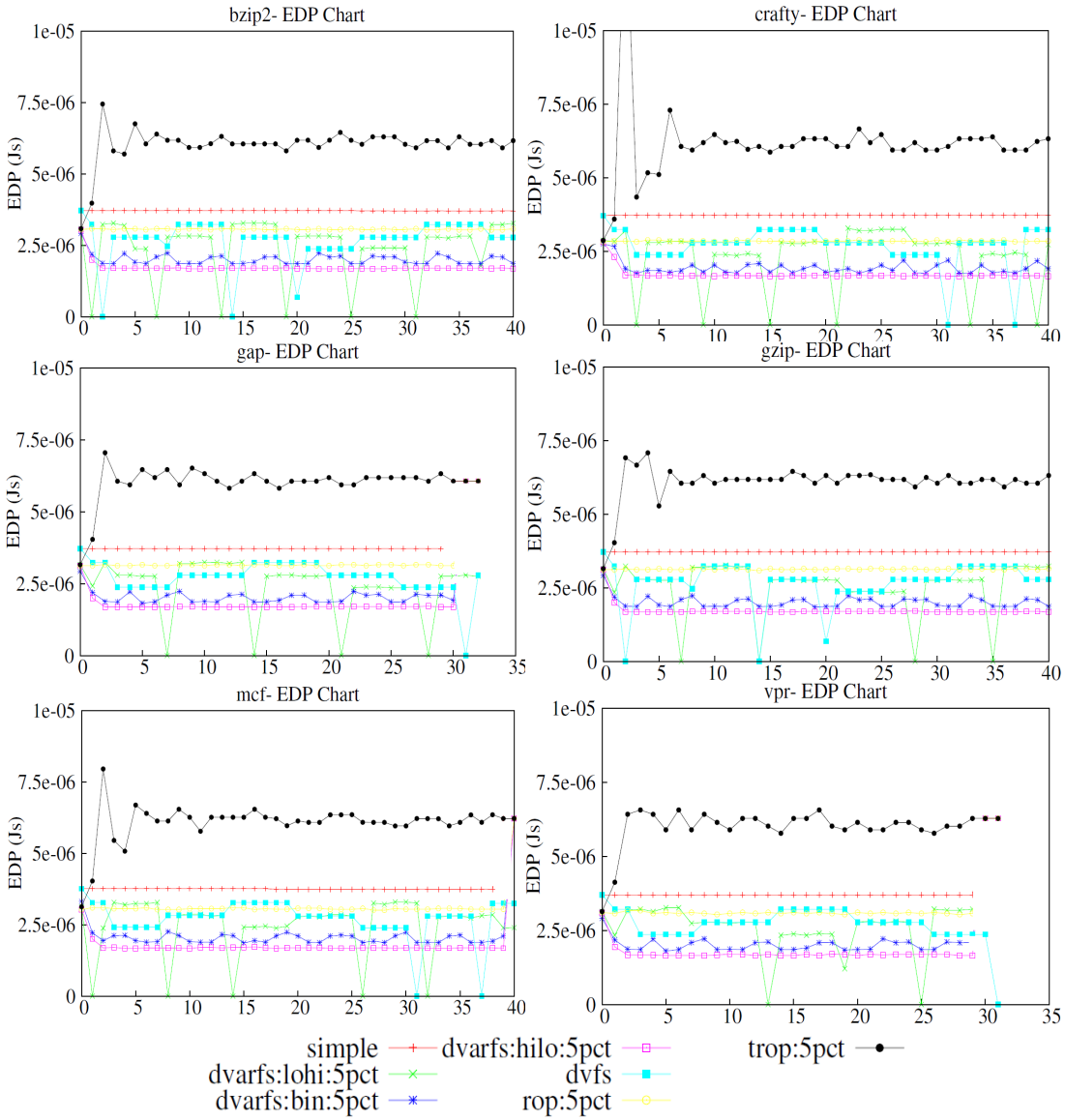


Figure A.15 EDP trace for SPEC INT workloads

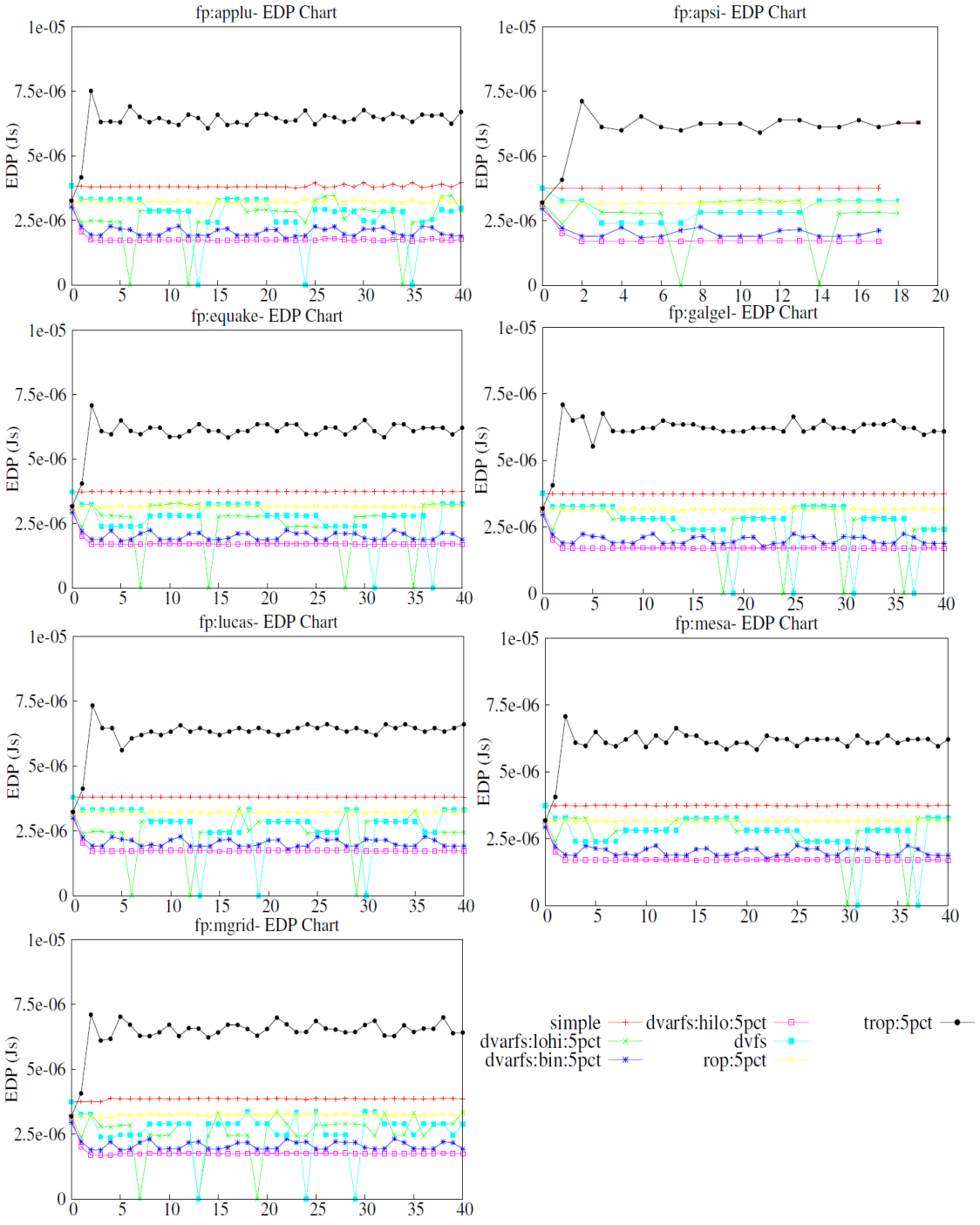


Figure A.16 EDP trace for SPEC FP workloads

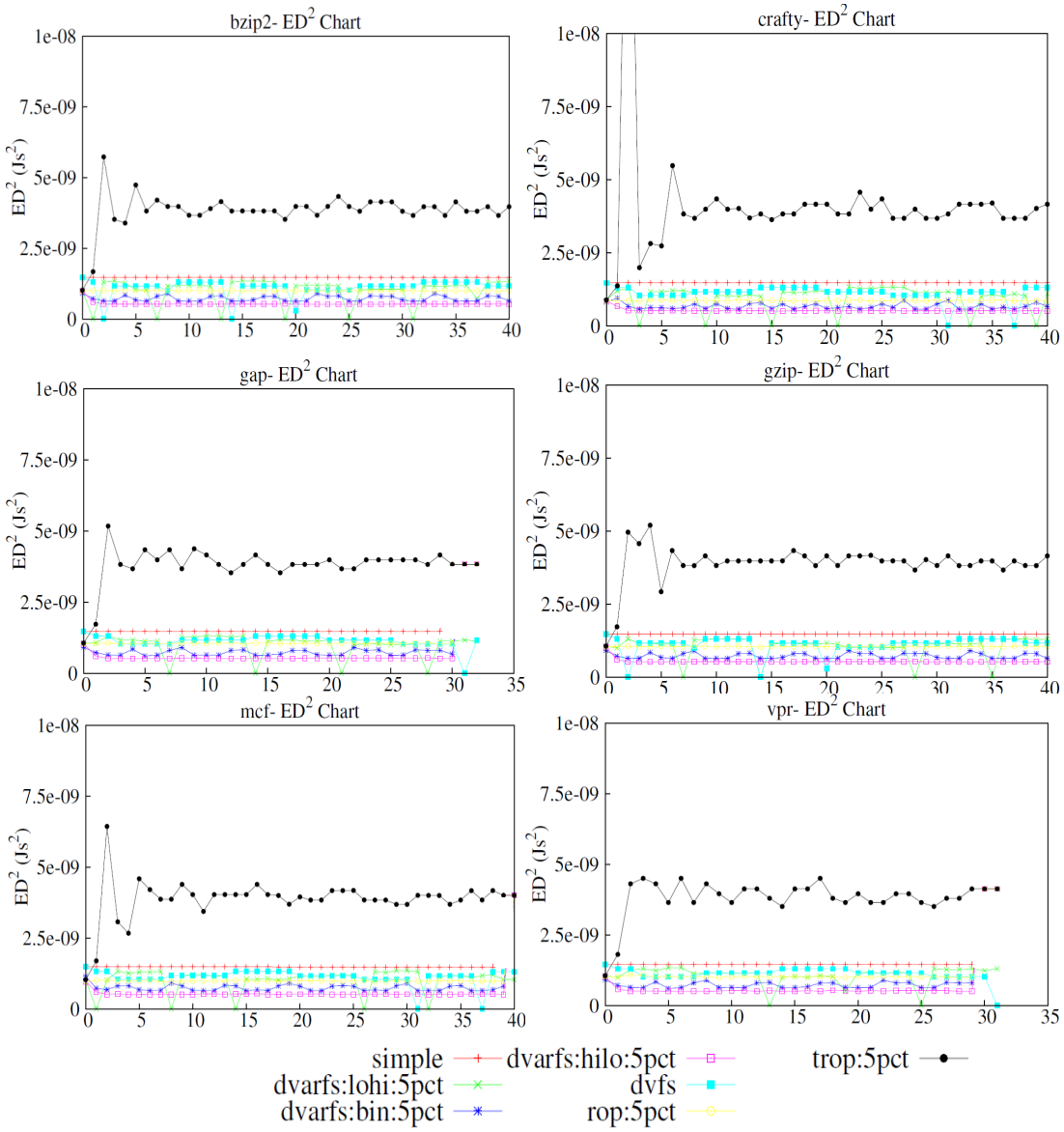


Figure A.17 ED^2 trace for SPEC INT workloads

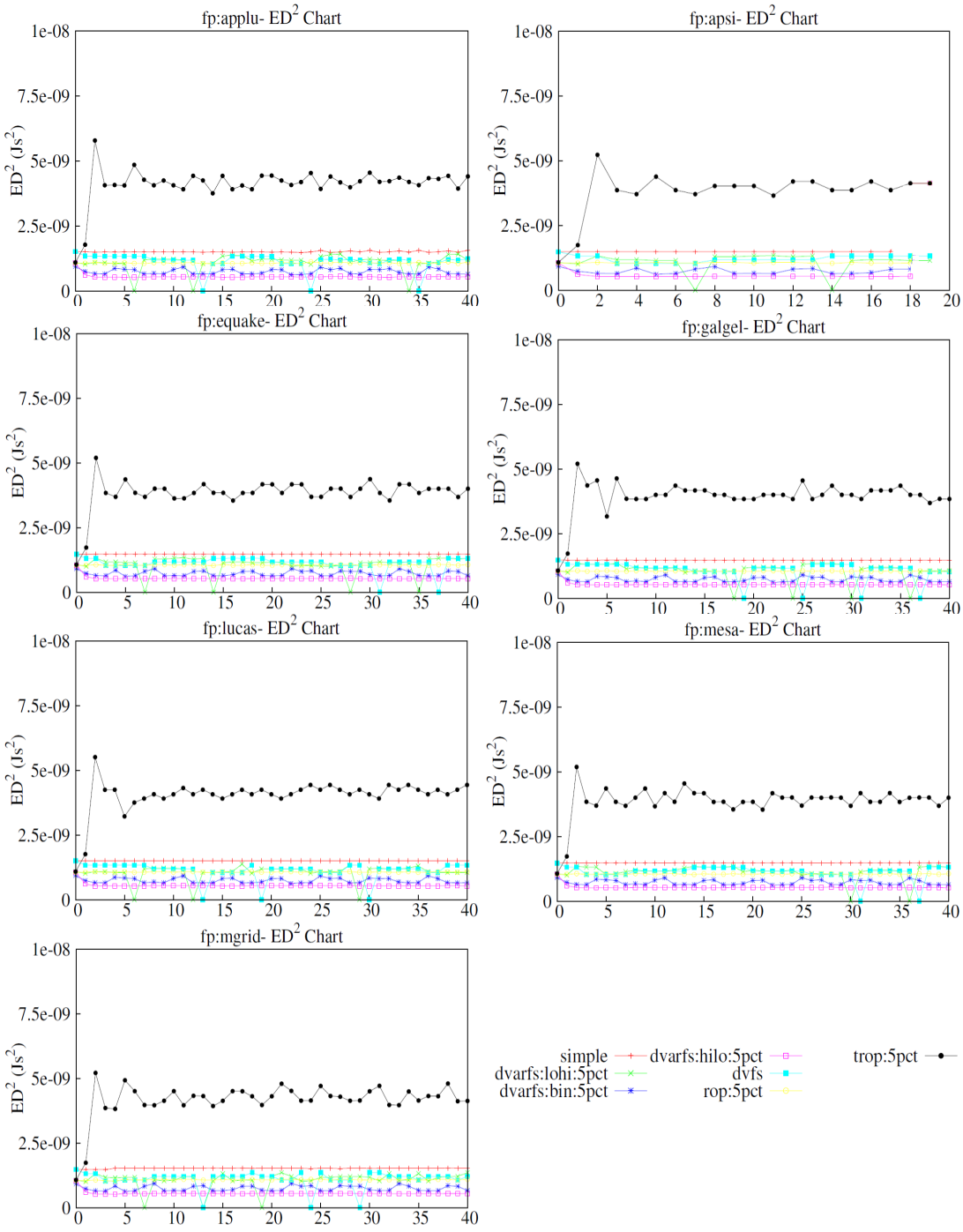


Figure A.18 ED^2 trace for SPEC FP workloads

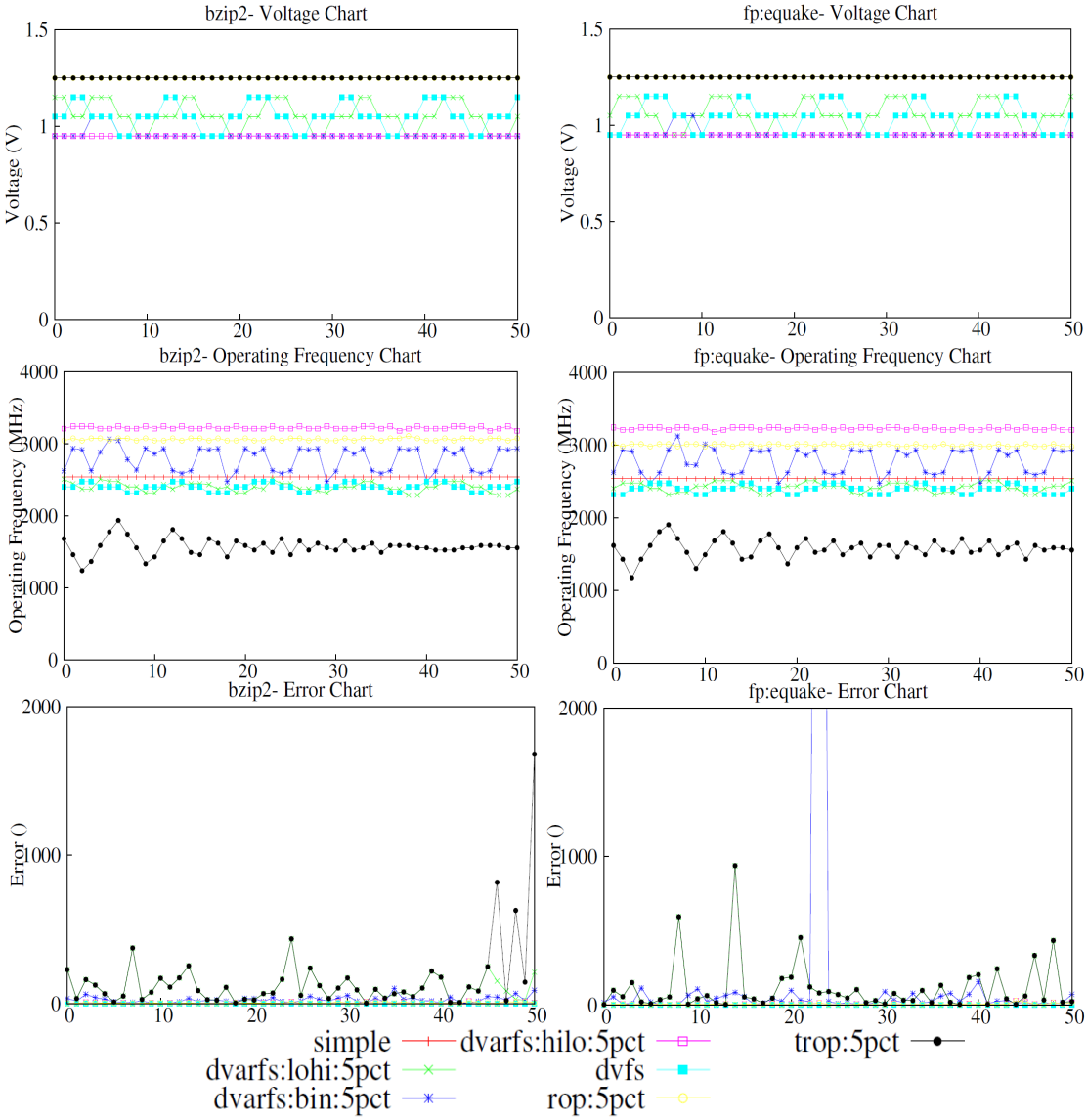


Figure A.19 Zoomed window of voltage, frequency and error traces

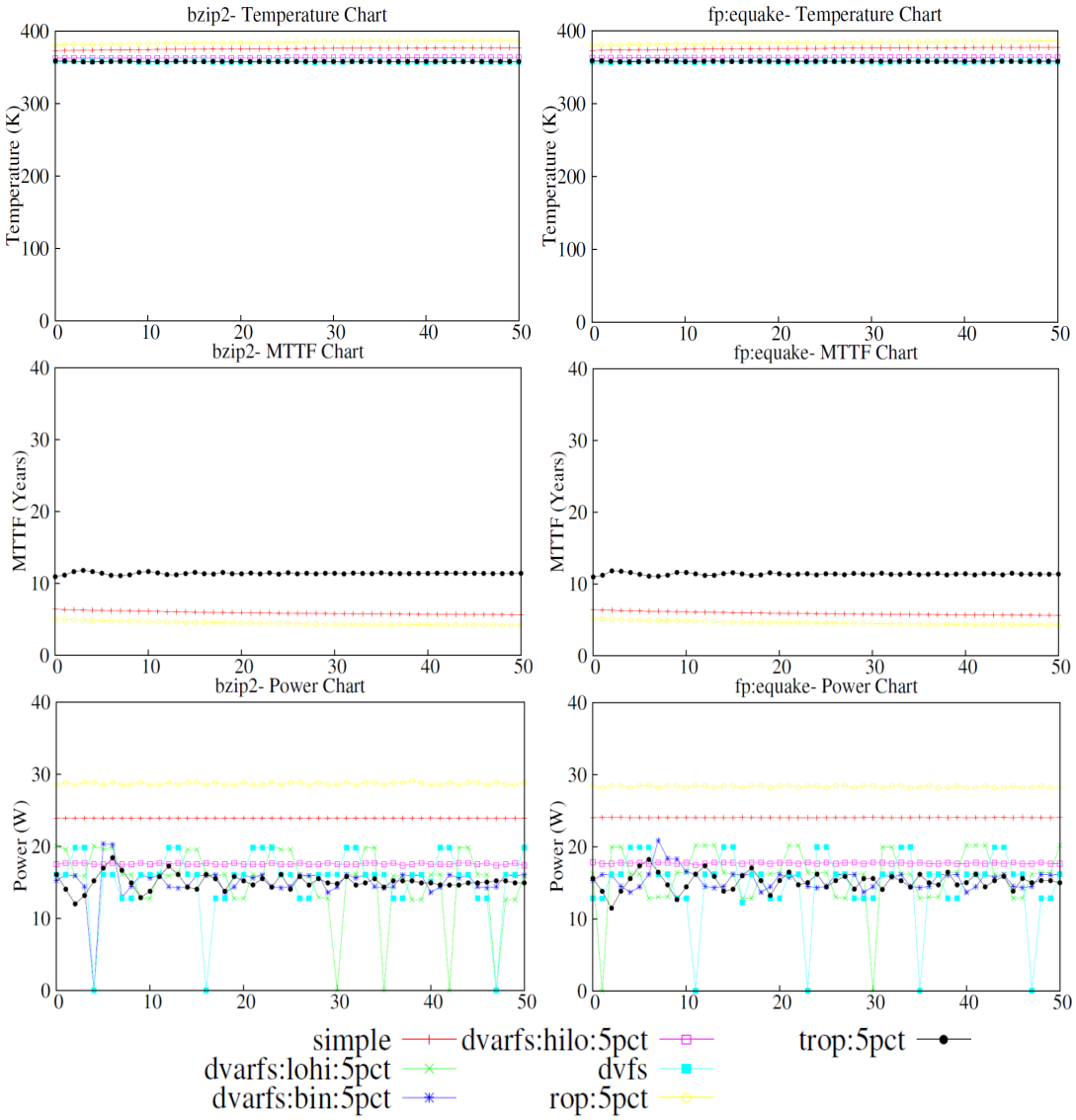


Figure A.20 Zoomed window of temperature, MTTF and power traces

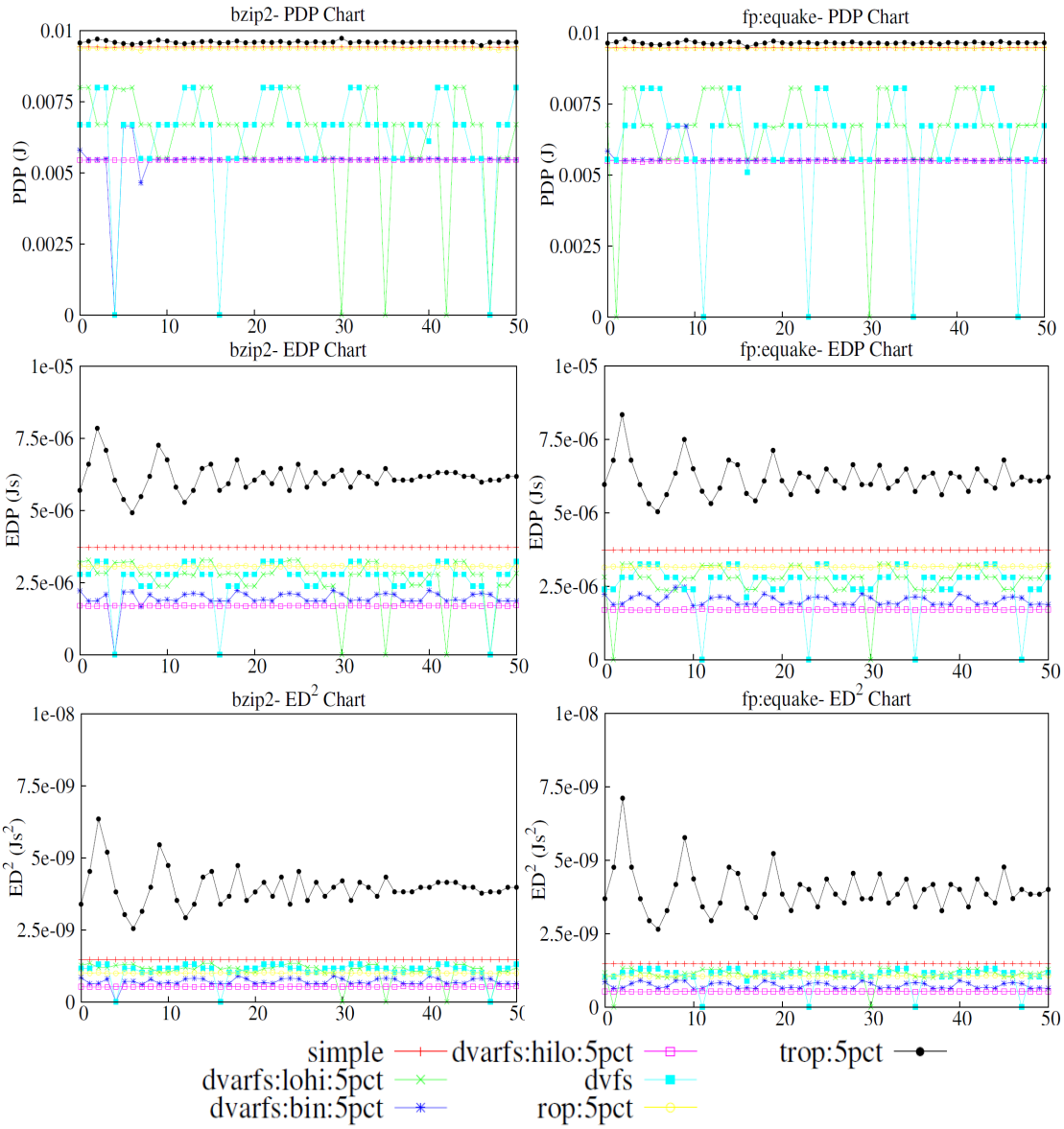


Figure A.21 Zoomed window of PDP, EDP and ED^2 traces

Bibliography

[1] G Wolrich, E McLellan, L Harada, J Montanaro, and R Yodlowski. *A High Performance Floating Point Coprocessor*. IEEE Journal of Solid State Circuits, 19(5), October 1984.

[2] *http://www.crn.com/hardware/212101254*.

[3] Viswanathan Subramanian, Mikel Bezdek, Naga D Avirneni, and Arun Somani. *Superscalar Processor Performance Enhancement Through Reliable Dynamic Clock Frequency Tuning*. International Conference on Dependable Systems and Networks, June 2007, pp. 196-205.

[4] Greskamp B. and Torrellas J. *Paceline: Improving Single-Thread Performance in Nanoscale CMPs through Core Overclocking*. 16th International Conference on Parallel Architecture and Compilation Techniques (PACT): 213-224, Sept. 2007.

[5] A.K. Uht. *Uniprocessor performance enhancement through adaptive clock frequency control*. IEEE Transactions on Computers, 54(2):132-140, February 2005.

[6] Dan Ernst et al. *Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation*. MICRO-2003, 2003.

[7] *http://www.intel.com/products/processor/corei7/index.htm*.

[8] International Technology Roadmap for Semiconductors. *Executive Summary, Date Accesed: April 10 2009*. [Online] http://www.itrs.net, 2007.

[9] A.K.Uht. Achieving typical delays in synchronous systems via timing error toleration. In *Technical report 032000-0100*, Dept. of Electrical and Computer Eng., Univ. of Rhode Island, Kingston, 2000.

[10] T. Sato and I. Arita. Constructive timing violation for improving energy efficiency. In *Compilers and operating systems for low power*, Kluwer Academic Publishers.

[11] X-Vera, O.Unsal, and A Gonzalez. X-pipe: An adaptive resilient microarchitecture for parameter variations. In Workshop on Architectural Support for GigascaleIntegration, 2006.

[12] Todd Austin, Valeria Bertacco, David Blaauw, and Trevor Mudge. Opportunities and challenges for better than worst-case design. In *ASP-DAC '05: Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, pages 2–7, New York, NY, USA, 2005. ACM.

[13] G. Memik, M.H. Chowdhury, A. Mallik, and Y.I. Ismail. Engineering over-clocking: reliability-performance trade-offs for high-performance register files. In *Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on*, pages 770–779, June-1 July 2005.

[14] K Sundaramoorthy, Z. Purser, and E. Rotenberg. Slipstreasm processors: Improving both performance and fault tolerance. pages 257–268, In ASPLOS, 2000.

[15] B. Greskamp and J Torrellas. Paceline: Improving single-thread performance in nanoscale cmps through core overclocking. pages 213–224, In PACT, 2007.

[16] F. Mesa J. Renau. Effective optimistic checker tandem core design through architectural pruning. In International Symposium on Microarchitecture, 2007.

[17] Todd M. Austin. Diva: a reliable substrate for deep submicron microarchitecture design. In *MICRO 32: Proceedings of the 32nd annual ACM/IEEE international symposium on Microarchitecture*, pages 196–207, Washington, DC, USA, 1999. IEEE Computer Society.

[18] S. Kim and A. K. Somani. Ssd: An affordable fault tolerant architecture for superscalar processors. In *Pacific Rim Dependable Computing Conference*, pages 27–34, December 2001.

[19] Abhishek Tiwari, Smruti R. Sarangi, and Josep Torrellas. Recycle: Pipeline adaptation to tolerate process variation. 34th International Symposium on Computer Architecture (ISCA), 2007.

[20] S. Sarangi, B. Greskamp, A. Tiwari, and J. Torrellas. Eval: Utilizing processors with variation-induced timing errors. pages 423–434, In International Symposium on Microarchitecture, 2008.

[21] S.R. Sarangi, B. Greskamp, and J. Torrellas. A model for timing errors in processors with parameter variation. In *Quality Electronic Design, 2007. ISQED '07. 8th International Symposium on*, pages 647–654, March 2007.

[22] B. Greskamp, Lu Wan, U.R. Karpuzcu, J.J. Cook, J. Torrellas, Deming Chen, and C Zilles. Blueshift: Designing processors for timing speculation from the ground up. pages 213–224, International Symposium on High Performance Computer Architecture, 2009.

[23] Seongmoo Heo, Kenneth Barr, and Krste Asanović. Reducing power density through activity migration. In *ISLPED '03: Proceedings of the 2003 international symposium on Low power electronics and design*, pages 217–222, New York, NY, USA, 2003. ACM.

[24] E. Rohou and M. Smith. Dynamically managing processor temperature and power. In *2nd Workshop on FeedbackDirected Optimization*, Nov 1999.

[25] T. Mudge. *Power: a first-class architectural design constraint.* IEEE Computer, 34(4):5258, April 2001.

[26] T. Burd T. Pering and R. Brodersen. *The simulation and evaluation of dynamic voltage scaling algorithms.* In International symposium on Low power electronics and design, pages 7681, 1998.

[27] P. Reed et al. *250 MHz 5W RISC microprocessor with onchip L2 cache controller.* Digest of Technical Papers - IEEE International Solid-State Circuits Conference, 40:412, 1997.

[28] H. Sanchez et al. *Thermal management system for high performance powerpc microprocessors.* Digest of Papers - COMPCON - IEEE Computer Society International Conference, 1997, page 325.

[29] David Brooks and Margaret Martonosi. *Dynamic Thermal Management for High-Performance Microprocessors.* Proceedings of the 7th International Symposium on High-Performance Computer Architecture, Monterrey, Mexico, January 2001.

[30] Karthik Sankaranarayanan et al. *A Case for Thermal-Aware Floorplanning at the Microarchitectural Level.* Journal of Instruction-Level Parallelism, 2005.

[31] G Paci et al. *Exploring "temperature-aware" design in low-power MPSoCs.* In the Proceedings of Design, Automation and Test in Europe, March 2006.

[32] S. Das, D. Roberts, S. Lee, S. Pant, D. Blaauw, T. Austin, K. Flaunter, and T. Mudge. *A Self-tuning DVS Processor using Delay-Error Detection and Correction.* IEEE Journal of Solid-State Circuits, 41(4):792-804, April 2006.

[33] T. D. Burd, T. A. Pering, A. J. Stratakos, and R. W. Brodersen. A dynamic voltage scaled microprocessor system. *IEEE Journal of Solid-State Circuits*, 35(11):1571–1580, November 2000.

[34] V. Gutnik and A. Chandrakasan. An efficient controller for variable supply-voltage low power processing. In *Symposium on VLSI Circuits*, pages 158–159, June 1996.

[35] S. Dhar, D. Maksimović, and B. Kranzen. Closed-loop adaptive voltage scaling controller for standard-cell asics. In *International symposium on Low power electronics and design*, pages 103–107, 2002.

[36] Jayanth Srinivasan, Sarita V. Adve, Pradip Bose, and Jude A. Rivers. *The Case for Lifetime Reliability-Aware Microprocessors.* The 31st International Symposium on Computer Architecture (ISCA-04), June 2004.

[37] Jayanth Srinivasan, Sarita V. Adve, Pradip Bose, and Jude A. Rivers. *Lifetime Reliability: Toward an Architectural Solution.* IEEE Micro, vol. 25, no. 3, pp. 70-80, May/Jun 2005.

[38] *Failure Mechanisms and Models for Semiconductor Devices.* Joint Electron Devoce Eng. Council Pub., JEP122-A, 2002.

[39] E.Y. Wu et al. *Interplay of Voltage and Temperature Acceleration of Oxide Breakdown for Ultra-Thin Gate Dioxides.* Solid-State Electronics J, pp.1787-1798, Nov 2002.

[40] S. Zafar et al. *A Model for Negative Bias Temperature Instability in Oxide and High K PFETs.* Symposia VLSI Technology and Circuits, pp.45-50, 2004.

[41] G. Hinton, D. Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker, and P. Roussel. The microarchitecture of the pentium 4 processor. In *Intel Technology Journal.*

[42] A. Hartstein and T. R. Puzak. The optimum pipeline depth for a microprocessor. In *In International Symposium on Computer Architecture*, pages 7–13, May 2002.

[43] Joel M. Tendler, Steve Dodson, Steve Fields, Hung Le, and Balaram Sinharoy. Power4 system microarchitecture. In *IBM Journal of Research and Development, Vol. 46, No. 1.*

[44] Ron Kalla, Balaram Sinharoy, and Joel M. Tendler. Ibm power5 chip: A dual-core multithreaded processor. *IEEE Micro*, 24(2):40–47, 2004.

[45] C. McNairy and R. Bhatia. Montecito: a dual-core, dual-thread itanium processor. *Micro, IEEE*, 25(2):10–20, March-April 2005.

[46] P. Kongetira, K. Aingaran, and K. Olukotun. Niagara: a 32-way multithreaded sparc processor. *Micro, IEEE*, 25(2):21–29, March-April 2005.

[47] James Donald and Margaret Martonosi. Techniques for multicore thermal management: Classification and new exploration. In *ISCA '06: Proceedings of the 33rd annual interna-*

tional symposium on Computer Architecture, pages 78–88, Washington, DC, USA, 2006. IEEE Computer Society.

[48] Wonyoung Kim, M.S. Gupta, Gu-Yeon Wei, and D. Brooks. System level analysis of fast, per-core dvfs using on-chip switching regulators. In *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, pages 123–134, Feb. 2008.

[49] Daniel C. Vanderster, Amirali Baniasadi, and Nikitas J. Dimopoulos. Exploiting task temperature profiling in temperature-aware task scheduling for computational clusters. In *Advances in Computer Systems Architecture, Volume 4697/2007*.

[50] Kyriakos Stavrou and Pedro Trancoso. Thermal-aware scheduling for future chip multiprocessors. *EURASIP J. Embedded Syst.*, 2007(1):40–40, 2007.

[51] A.K. Coskun, T.S. Rosing, and K. Whisnant. Temperature aware task scheduling in mpsocs. In *Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE '07*, pages 1–6, April 2007.

[52] Yefu Wang, Kai Ma, and Xiaorui Wang. Temperature-constrained power control for chip multiprocessors with online model estimation. *SIGARCH Comput. Archit. News*, 37(3):314–324, 2009.

[53] Abhishek Tiwari and Josep Torrellas. Facelift: Hiding and slowing down aging in multicores. In *MICRO '08: Proceedings of the 2008 41st IEEE/ACM International Symposium on Microarchitecture*, pages 129–140, Washington, DC, USA, 2008. IEEE Computer Society.

[54] Krishna K. Rangan, Gu-Yeon Wei, and David Brooks. Thread motion: fine-grained power management for multi-core systems. *SIGARCH Comput. Archit. News*, 37(3):302–313, 2009.

[55] International Technology Roadmap for Semiconductors. *Assembly and Packaging*. http://www.itrs.net/Links/2007ITRS/, 2007 Edition.

[56] Viswanathan Subramanian, Prem Kumar Ramesh, and Arun K. Somani. Managing the impact of on-chip temperature on the lifetime reliability of reliably overclocked systems. In *Second International Conference on Dependability-DEPEND'09*, June 2009.

[57] PowerPC 750GX DPPS facilities. *http://www.ibm.com/developerworks/power/library/pa-nl31-tip/index.html?S_TACT=105AGX16&S_CMP=EDU.*

[58] J. Huang et al. *A Robust Physical and Predictive Model for Deep-Submicrometer MOS Circuit Simulation.* Proceedings of the IEEE Custom Integrated Circuits Conference, pp.14.2.1-4, May 1993.

[59] Doug Burger and Todd M Austin. *The SimpleScalar Tool Set, Version 2.0.* University of Wisconsin-Madison Computer Sciences Department Technical Report #1342, June 1997.

[60] W. Huang, K. Sankaranarayanan, R. J. Ribando, M. R. Stan, and K. Skadron. *An Improved Block-Based Thermal Model in HotSpot 4.0 with Granularity Considerations.* In Proceedings of the Workshop on Duplicating, Deconstructing, and Debunking, in conjunction with the 34th International Symposium on Computer Architecture (ISCA), June 2007.

[61] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *International Symposium on Computer Architecture (ISCA)*, pages 83–94, 2000.

[62] James Donald and Margaret Martonosi. *An Efficient, Practical Parallelization Methodology for Multicore Architecture Simulation.* in Computer Architecture Letters, vol. 5, August 2006.

[63] Karthikeyan Sankaralingam, Ramadass Nagarajan, Stephen W. Keckler, and Doug Burger. *SimpleScalar Simulation of the PowerPC Instruction Set Architecture.* in UT Austin Technical Report: CS-TR-00-04, 2001.

[64] J.E. Stine et al. FreePDK: An Open-Source Variation-Aware Design Kit. In *Proceedings of the 2007 IEEE International Conference on Microelectronic Systems Education*, pages 173–174, 2007.

[65] M. Bezdek. Utilizing timing error detection and recovery to dynamically improve superscalar processor performance. Master's thesis, Iowa State University, 2006.

[66] Viswanathan Subramanian. Timing speculation and adaptive reliable overclocking techniques for aggressive computer systems. Technical report, A dissertation submitted to the Department of Electrical and Computer Engineering, Iowa State University, Ames, Iowa, 2009.

[67] Alain J. Martin, Mika Nyström, and Paul I. Pénzes. *ET2: a metric for time and energy efficiency of computation*, pages 293–315. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[68] T. M. Austin. Diva: a reliable substrate for deep submicron microarchitecture design. In *International Symposium on Microarchitecture*, pages 196–207, 1999.

[69] B. Greskamp, L. Wan, U.R. Karpuzcu, J.J. Cook, J. Torrellas, D. Chen, and C. Zilles. Blueshift: Designing processors for timing speculation from the ground up. In *IEEE 15th International Symposium on High Performance Computer Architecture, 2009. HPCA 2009*, pages 213–224, 2009.

[70] Viswanathan Subramanian and Arun K. Somani. Conjoined Pipeline: A Fault-Tolerant High Performance Microarchitecture. In *Pacific Rim International Symposium on Dependable Computing, , Taipei, Taiwan, Dec*, 2008.

[71] Naga Durga Avirneni, Viswanathan Subramanian, and Arun K. Somani. Low Overhead Soft Error Mitigation Techniques for High-Performance and Aggressive Systems. In *IEEE/IFIP Dependable Systems and Networks*, 2009.

[72] P.K. Ramesh, V. Subramanian, and Arun K Somani. System Level Analysis for Achieving Thermal Balance and Lifetime Reliability in Reliably Overclocked Systems. *International*

Journal on Advances in Systems and Measurements, issn 1942-261x, vol. 2, no. 4:258:268, "2009".

[73] P.K. Ramesh, V. Subramanian, and Arun K Somani. Thermal Management in Reliably Overclocked Systems. *IEEE Workshop on Silicon Errors in Logic - System Effects, Stanford University*, March 24-25, "2009".

[74] N. V. Shenoy, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Minimum padding to satisfy short path constraints. In *IEEE/ACM International conference on Computer-aided design*, pages 156–161, 1993.

[75] Kuang-Chien Chen and Saburo Muroga. Timing optimization for multi-level combinational networks. In *DAC '90: Proceedings of the 27th ACM/IEEE Design Automation Conference*, pages 339–344, New York, NY, USA, 1990. ACM.

[76] D.H.C. Du, S.H.C. Yen, and S. Ghanta. On the general false path problem in timing analysis. In *Design Automation, 1989. 26th Conference on*, pages 555 – 560, 1989.

[77] Lei Cheng, Deming Chen, Martin D. F. Wong, Mike Hutton, and Jason Govig. Timing constraint-driven technology mapping for fpgas considering false paths and multi-clock domains. In *Proceedings of the 2007 IEEE/ACM international conference on Computer-aided design*, ICCAD '07, pages 370–375, Piscataway, NJ, USA, 2007. IEEE Press.

[78] Shihheng Tsai and Chung-Yang (Ric) Huang. A false-path aware formal static timing analyzer considering simultaneous input transitions. In *Proceedings of the 46th Annual Design Automation Conference*, DAC '09, pages 25–30, New York, NY, USA, 2009. ACM.

[79] Olivier Coudert. An efficient algorithm to verify generalized false paths. In *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pages 188 –193, 2010.

[80] H.-C. Chen, D.H.-C. Du, and L.-R. Liu. Critical path selection for performance optimization. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 12(2):185 –195, February 1993.

[81] H.J. Touati, H. Savoj, and R.K. Brayton. Delay optimization of combinational logic circuits by clustering and partial collapsing. In *Computer-Aided Design, ICCAD-91. Digest of Technical Papers., 1991 IEEE International Conference on*, pages 188 –191, November 1991.

[82] Kurt Antreich Bernhard Rohfleisch, Bernd Wurth. Logic clause analysis for delay optimization. In *Design Automation, 1995. DAC '95. 32nd Conference on*, 1995.

[83] L.A. Entrena, J.A. Espejo, E. Olias, and J. Uceda. Timing optimization by an improved redundancy addition and removal technique. In *Design Automation Conference, with EURO-VHDL '96 and Exhibition, Proceedings EURO-DAC '96, European*, pages 342 –347, September 1996.

[84] Chen-Liang Fang and Wen-Ben Jone. Timing optimization by gate resizing and critical path identification. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 14(2):201 –217, February 1995.

[85] Aiguo Lu, H. Eisenmann, G. Stenz, and F.M. Johannes. Combining technology mapping with post-placement resynthesis for performance optimization. In *Computer Design: VLSI in Computers and Processors, 1998. ICCD '98. Proceedings. International Conference on*, pages 616 –621, October 1998.

[86] H.-F. Jyu and S. Malik. Statistical timing optimization of combinational logic circuits. In *Computer Design: VLSI in Computers and Processors, 1993. ICCD '93. Proceedings., 1993 IEEE International Conference on*, pages 77 –80, October 1993.

[87] Peichen Pan. Performance-driven integration of retiming and resynthesis. In *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, DAC '99, pages 243–246, New York, NY, USA, 1999.

[88] R. Fung, V. Betz, and W. Chow. Simultaneous short-path and long-path timing optimization for fpgas. In *Proceedings of the 2004 IEEE/ACM International conference on*

Computer-aided design, ICCAD '04, pages 838–845, Washington, DC, USA, 2004. IEEE Computer Society.

[89] Mark C. Hansen, Hakan Yalcin, and John P. Hayes. Unveiling the iscas-85 benchmarks: A case study in reverse engineering. *IEEE Design and Test of Computers*, 16:72–80, 1999.

[90] Jason F. Cantin and Mark D. Hill. Cache performance for selected spec cpu2000 benchmarks. *SIGARCH Comput. Archit. News*, 29:13–18, September 2001.

[91] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, and Katherine A. Yelick. The landscape of parallel computing research: A view from berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec 2006.

[92] K Skadron, Pradip Bose, Kanad Ghose, Resit Sendag, Joshua J. Yi, and Derek Chiou. Low-power design and temperature management. *IEEE Micro*, 27(6):46–57, 2007.

[93] A.K. Coskun, R. Strong, D.M. Tullsen, and T.S. Rosing. Evaluating the impact of job scheduling and power management on processor lifetime for chip multiprocessors. In *Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*, pages 169–180. ACM New York, NY, USA, 2009.

[94] S. Hebert and D. Marculescu. Variation-aware dynamic voltage/frequency scaling. In *High-Performance Computer Architecture*, 2009.

[95] Canturk Isci, Alper Buyuktosunoglu, Chen-Yong Cher, Pradip Bose, and Margaret Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 347–358, Washington, DC, USA, 2006. IEEE Computer Society.

[96] K. Meng, R. Joseph, R.P. Dick, and L. Shang. Multi-optimization power management for chip multiprocessors. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 177–186. ACM New York, NY, USA, 2008.

[97] SPEC. http://www.spec.org/cpu2006/results.

[98] Radu Teodorescu and Josep Torrellas. Variation-aware application scheduling and power management for chip multiprocessors. In *ISCA '08: Proceedings of the 35th International Symposium on Computer Architecture*, pages 363–374, Washington, DC, USA, 2008. IEEE Computer Society.

[99] *Intel Centrino Mobile Technology.* Intel Technology Journal, Volume 07, Issue 02, ISSN 1535-864X, May 2003.

[100] N. J. Wang, J. Quek, T. M. Rafacz, and S. J. Patel. Characterizing the effects of transient faults on a high-performance processor pipeline. In *IEEE/IFIP International conference on Dependable Systems and Networks*, pages 61–70, 2004.

[101] Illinois Advanced Computing Systems Group. *Illinois Verilog Model.* http://www.crhc.uiuc.edu/ACS/tools/ivm/about.html.

[102] J.E. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W.R. Davis, P.D. Franzon, M. Bucher, S. Basavarajaiah, J. Oh, et al. FreePDK: An Open-Source Variation-Aware Design Kit. In *Proceedings of the 2007 IEEE International Conference on Microelectronic Systems Education*, pages 173–174, 2007.