NORTHWESTERN UNIVERSITY

Microarchitectural Approaches for Optimizing Power and Profitability in Multi-core Processors

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Electrical Engineering and Computer Science

By

Abhishek Das

EVANSTON, ILLINOIS

October 2010

ABSTRACT

Microarchitectural Approaches for Optimizing Power and Profitability in Multi-core Processors

Abhishek Das

Chip multiprocessors (CMPs) or multicore processors continue to have increasing core-counts with advances in process technology. This more-core-per-die trend introduces several concerns which need to be addressed by architects. This thesis addresses two such important concerns: (i) profitability in the manufacturing process, and (ii) power-efficient computing. Firstly, manufacturing variations/defects is becoming more prominent with technology scaling and has a direct impact on chip power and performance. Because of process variations, chips from a single batch are rated by different frequencies and sold at different prices. An efficient binning distribution thus decides the profitability of the chip manufacturer. Similarly, variation-aware supply voltage assignment to cores opens up the possibility of extra power savings. The first part of this thesis proposes several architectural schemes that result in efficient speed binning, hence impacting the profitability of chip manufacturers. The

power problem is dealt by customizing voltage-islands in a multicore chip in such a way that both leakage and dynamic power dissipation is reduced.

The second part of the thesis addresses the multicore power problems with respect to on-chip interconnection network. The latter facilitates the active sharing of data on chip and already consume a large fraction of the chip's power, and the rapidly increasing core counts in future technologies further aggravate the problem. This thesis proposes Power-Aware Directory Placement (PAD), a novel distributed cache architecture that co-locates directories together with highly active sharers of the corresponding data, thereby eliminating a large fraction of the on-chip interconnect traversals. Results demonstrate that such a scheme can significantly reduce on-chip interconnect power with negligible hardware overhead and small performance improvement.

Acknowledgements

First and foremost I would like to thank my principal advisor Professor Alok Choudhary, not only for motivating and directing my research, but providing me with ample opportunities and flexibility to expand my role as a graduate student. Secondly, I would like to acknowledge the valuable advice and guidance of my co-advisor Professor Gokhan Memik, who owes much credit in shaping up this thesis. I would also like to thank Professor Nikos Hardavellas and Professor Joseph Zambreno (Iowa State University) for their collaborations during the last few years, and for serving on my Final Examination committee. Lastly, I would be negligent if I didn't mention the support of my father Professor Ramtanu Das and my mother Krishna Das, who have provided me with encouragement and support during every step of this journey.

This thesis work was supported in part by NSF grants CCF-0916746, CCF-0747201, CNS-0830927, CNS-0551639, IIS-0536994, HECURA CCF-0621443, OCI 0956311, OCI 0724599, and SDCI OCI-0724599.

Table of Contents

Table of Figures

Table of Tables

Chapter 1

Introduction

As the number of transistors on a chip doubles every eighteen months according to Moore's Law, power and reliability of has become the most important concern for the microprocessor designers. The reason is two-fold; first, the shrinking process technology (32nm and beyond) has caused what is called process variations or fluctuations in manufacturing process parameters. Because of such process variations, a chip can have high latency or excessive leakage power dissipation, which can lead to yield loss. Secondly, because of frequency scaling with every technology generation, the power (both leakage and dynamic) dissipation increases exponentially and gives rise to the phenomenon called 'power wall'. Technology scaling produces no more benefits at this point. To handle this problem chip manufacturers have started fabricating multiple processing cores on the same silicon die, and hence these are known as chip multi-processors or CMPs. Furthermore to ensure effective communication between multiple communicating cores an on-chip interconnection network is used. However, recent work shows that such an interconnection fabric can dissipate up to 23% of the

chip power [39]. Thus, optimizing on-chip interconnection power can contribute to the lowering of the overall processor power budget.

In this thesis we cover both these problem namely that of reliability and power. Particularly, we focus on process variation-tolerant and power-aware multi-core architectures. The following paragraphs discuss the above mentioned problems in detail and propose microarchitectural and system level solutions which try to solve each.

Reliability and Process Variations: As transistors are reduced in size, it becomes harder to control variations in device parameters such as channel length, gate width, oxide thickness, and device threshold voltage. These fluctuations in the process parameter distributions or simply process variations cause increased variability in circuit performance and are likely to be more dominant in sub-90 nm domain. Even in a relatively mature technology like 130 nm, these variations are known to result in as much as 30% decrease in maximum frequency and 500% increase in leakage power [18]. For newer technologies, these variations can be even higher: 20-fold increases in leakage have been reported for 90nm [9]. Process variations consist of With-in-Die (WID) and Inter-Die or Die-to-Die (D2D) parameter variations. As a direct impact of this, a chip may under-perform or turn out to be leakier than a certain threshold and hence may be eventually dropped resulting in effective yield loss. A common practice to remedy the effects of process variations is speed-binning (Figure 3.1**Error! eference source not found.**). Speed-binning is usually performed by testing each

manufactured chip separately over a range of frequency levels until it fails. As a result of the inherent process variations, the different processors fall into separate speed bins, where they are rated and marketed differently. This process helps the chip manufacturer create a complete product line from a single design.

In contrast to speed-binning, architectural changes made for performance enhancements are generally analyzed by considering its effects on high-level metrics such as instructions-per-cycle (IPC) and/or cycle time. However, because of the effects of process variations, different chips can have different post-fabrication frequencies irrespective of the changes made. Hence IPC and clock cycle are not enough to judge the effects of an architectural modification on the performance of a whole batch of chips. As a result, we need to establish new metrics when the process variations are considered. Chip yield is one obvious metric, as the continuing downward scaling of transistor feature sizes has made fabrication considerably more difficult and expensive [46, 49, 52]. However, an approach that optimizes solely for yield, would not take into account the fact that CPUs concurrently manufactured using a single process are routinely sold at different speed ratings and prices. For example, from a manufacturer's perspective, having a 20% yield where the chips have 2.4 GHz frequency may be more desirable than having a 50% yield where the processors have 1.0 GHz maximum clock frequency. In this thesis, we show that changing the binning distribution can actually impact the revenue of a company and add to the profitability of the chip manufacturer.

We also propose a new metric called batch-performance (BP) that corresponds to the average performance of the batch of chips manufactured using a given architecture, therefore captures the distribution of chips as well as the chip yield. The first part of the thesis proposes several microarchitectural schemes that optimizes for both revenue and BP and draw a comparison between the two.

Impact on power dissipation in CMPs: CMPs are the latest trend in chip industry. With process technology advancements, multiple cores are laid down on the same die to exploit parallelism in applications and provide higher performance. Just like their single processor counterparts, CMPs are no exceptions with respect to D2D and WID variations. In fact, in CMPs, the problem of parameter variability is more acute because rampant WID variations may result in core-to-core (C2C) variations [28]. As a result, the performance of some cores drop beyond the expected level and a nominal frequency of operation is chosen to be equal to the frequency of the slowest core. Besides, D2D variations also cause chips to differ from each other. Under such circumstances, having a single Vdd level for all the manufactured chips is power-inefficient, since there are significant variations between chips manufactured in the same batch. Intuitively, power savings can be achieved by setting a customized Vdd for each chip, or a set of cores in a chip thus forming one or more voltage islands. This thesis report covers schemes that can result in dynamic power savings under process variations in CMPs.

Power due to on-chip interconnects in CMPs: With aggressive technology scaling the core-count within a single die increases, leading to increase the last-level on-chip cache (LLC). With increasing cache size together with increasing input data size the cache access latency increases exponentially. To remedy this latency problem multicore designers have proposed non-uniform cache architectures (NUCA) whereby the LLC is distributed and each core has a local cache slice which it can access with lower latency compared to a remote cache slice associated with another core. To maintain data coherence for such a private NUCA cache organization a chip-wide global directory is maintained to keep track of the coherence information. Hence any request by a tile to access a block of shared data is forwarded to this chip-wide global directory, which for most cases is located in a remote tile. Such a request messages to a remote tile will generate on-chip network traffic which in turn contributes to the overall active power dissipation.

In this work we analyze such data access patterns for shared and private data in such a tiled architecture. We observe a large fraction of the on-chip interconnect power is spent on the control messages sent to the directory for data coherence, which is a consequence of the static dependence of the directory location on the physical address of a block. Moreover, this work proposes *Power-Aware Directory Placement (PAD)* scheme that tries to reduce the network messages by co-locating a directory slice with the requesting core or one of the cores which participates in data sharing. Results from

trace and timing simulations show considerable scope for reducing the total number of network messages, which has an impact on the overall chip power. The implementation and evaluation of the proposed scheme are presented in the latter half of the thesis.

The rest of the Ph.D. thesis is organized as follows. Chapter 2 discusses the background and related work in the areas of power and reliability in processor architecture. The cache-resizing and cache-redundancy schemes for efficient speed-binning and batch-performance and revenue optimization are described in Chapter 1. Chapter 1 discusses voltage-island schemes for power reduction in multicore processors. Chapter 1 describes the Power-Ware Directory co-location scheme on-chip interconnection power reduction. Finally, Chapter 1 concludes the thesis and suggests research directions which can be pursued in the future.

Chapter 2

Background and Related Work

Power and variability in integrate circuits has been extensively studied. In this chapter, we present brief overviews of the prior works that are proposed to mitigate effects of parameter variations and reduce on-chip power dissipation, ranging from architectural to circuit-level techniques.

Cache Redundancy schemes for increasing reliability:

Several cache redundancy schemes have been proposed [10, 32, 48, 66, 68]. These techniques have been/could be used to reduce the critical delay of a cache. Victim caches [32] are extra level of caches used to hold blocks evicted from a CPU cache due to a conflict or capacity miss. A substitute cache storing the slower blocks of the cache is orthogonal to a victim cache, which stores blocks evicted from the cache. Sohi [68] shows that cache redundancy can be used to prevent yield loss, using a similar concept to our proposed SC. Shivakumar et al. [66] introduce a new yield metric and propose the utilization of redundant structures to increase it. Similarly, techniques like sparing DRAMS and Chipkill are used in Sun's UltraSPARC T1 processor [10]. Cache redundancy is also present in commercial processors like Itanium, which uses extra

banks to improve fault tolerance [48]. Finally, Romanescu et al. [60] propose prefetching data into fast buffers to address Process Variations in L1 caches as well as prioritizing critical instructions to utilize fast registers/functional units. Compared to these alternatives, our SC scheme introduces considerably lower cost in terms of area and latency.

Circuit level techniques:

Previous works show that several circuit-level techniques could be adopted to counter the negative effects of process variations [9, 15, 18, 54, 58, 68]. The inter- and intra-die process variations and their effects on circuit leakage is studied in detail by Rao et al. [57]. In another work, Rao et al. [56] analyze the impact of process variations on circuit leakage and propose methods to reduce them. Most of these techniques focus on analyzing the design statistically or by using static timing analysis, and then modifying the parts of the circuits that are most susceptible to variations. Liang et al. have proposed a variation-tolerant 3 transistor, 1 diode on-chip dynamic memory as a substitute for the traditional 6 transistor SRAM [41]. Many gate-sizing strategies have been used on the critical or near-critical regions of the circuit in order to reduce the effective latency [17, 79]. Although these techniques increase the overall yield, no analysis of the impact on binning has been done.

Performance binning has also been discussed as a means for increasing yield [9, 18, 58]. Datta et al. [18] propose a novel approach of changing the effective speed-binning by

gate sizing, and thus increasing the profit. Unlike our schemes, most of their analyses are based on statistical estimations of yield, and the optimizations are for high-level synthesis. Kim et al. [37] have studied the effects of cache size on leakage and analyzed the tradeoff on access time when multiple threshold voltages are used for L1 and L2 caches. In contrast, we perform the replication within the cache and also present a detailed implementation.

Variation-tolerant architectural schemes:

Ozdemir et al. [52] present microarchitectural schemes that improve the overall chip yield under process variations to as much as 97%. The authors have shown how powering down sections of the cache can increase the effective yield. Our work, on the other hand encompasses extra redundancy in L1 caches to facilitate efficient binning and profit maximization. Besides, our model includes the entire processor pipeline, as opposed to only L1 caches. Techniques for mitigating the effects of process variations by using variable latency register files and execution units have been proposed by Liang et al. [40]. In a recent work, Liang et al. [42] have studied the effectiveness of post fabrication techniques like voltage interpolations and variable latency in different pipeline loops. Besides, Agarwal et al. [2] propose a scheme that prevents yield loss due to failures in the SRAM cells of the cache. Their approach is mostly based on Built-In Self-Test (BIST) circuitry and the cache optimizations are concentrated towards yield maximization. Teodorescu et al. [73] use variation-aware instruction scheduling and

power management to reduce the overall energy delay product and increase throughput of chip multiprocessors. Tiwari et al. [75], on the other hand, propose ReCycle to balance the delay variations between different pipeline stages due to process variations by utilizing a skewed clock. This way, they achieve a processor clock frequency that is equivalent to the average frequency of all pipeline stages instead of the lowest one. They also propose a method to convert the slack in faster stages into power savings. Lee et al. [38] discusses a new metric involving yield, area, and performance for evaluating the tradeoff between yield and performance in caches. In comparison to the abovementioned works, our efforts are directed towards efficient binning and revenue optimization for set associative caches. In addition, most of the previous techniques listed above have performance implications, i.e., different chips in a frequency bin may exhibit varying performance levels (due to a variation in the IPC). However, our SC scheme provides the same performance (constant IPC) for all the chips in a frequency bin.

Other works on reliability:

There has been plethora of studies analyzing cache resizing for different goals such as minimizing power consumption or increasing performance. Selective Cache Ways by Albonesi [3] is one of the first works in cache resizing and optimizing energy dissipation of the cache. Flautner et al. [23] have proposed a drowsy cache architecture, which takes into account the state of a cache line and correspondingly changes its mode.

The concept of cache decay, on the other hand, exploits the usage information of each cache line in order to turn them off when they are not in use to save leakage power [34]. Yang et al. [80] have analyzed the effects of various cache resizing schemes on reducing the energy-delay of deep submicron processors. Powell et al. [53] have introduced the Gated-Vdd approach, by which the supply voltage is turned off in the unused SRAM cells to save leakage energy. Finally, Tadas and Chakrabarti have developed a scheme in which the adjacent micro-blocks of a cache are resized depending on the hit and miss rate [71].

Power-aware voltage/frequency scaling in multicore architectures:

Variable Voltage/Frequency Islands (VFI's) have been previously used by other researchers [16, 43-44]. Marculescu et al. [43] show that VFI-based latency-constrained system are more likely to meet timing constraints than Single Clock, Single Frequency (SSV) based systems. In another work, Marculescu et al. [44] have suggested a GALS like architecture with multiple voltage islands for energy awareness under parameter variations. Dhar et al. [19] have designed a controller-based adaptive supply voltage scaling (AVS) mechanism for standard cell ASICs. Niyogi et al. [51] have addressed the issue of using multiple VFIs for energy optimization in media and signal processing applications. These works, although important and showing the advantages of customized voltage islands, do not study the CMPs but concentrate on application-specific processors. In a recent work, Humenay et al. [29] have studied the effects of

core-to-core (C2C) variations on power dissipation and yield of chip multicore processors. The authors have investigated the effects of systematic variations on dense and distributed floorplans of a CMP, and used Adaptive Voltage Scaling (AVS) techniques to boost the performance of slow cores. Our work on the other hand emphasizes on the importance of multiple voltage islands in CMPs, to reduce power dissipation, and performs a detailed analysis of the advantages for various voltage island formations.

CMP cache and memory management schemes:

To mitigate the access latency of large on-chip caches, Kim *et al.* propose Non-Uniform Cache Architectures (NUCA) [35], showing that a network of cache banks can be used to reduce average access latency. NUCA caches and directory management in chip multiprocessors (CMPs) have been explored by several researchers in the past.

To improve cache performance, Dynamic NUCA [35] attracts cache blocks to the requesting cores, but requires complex lookup algorithms. CMP-NuRAPID [14] decouples the physical placement of a cache block from the cache's logical organization to allow the migration and replication of blocks in a NUCA design, but requires expensive hardware pointers for lookup, and coherence protocol modifications. Cooperative Caching [12] proposes the statistical allocation of cache blocks in the local cache to strike a balance between capacity and access speed, but requires centralized structures that do not scale, and the allocation bias is statically defined by the user for

each workload. ASR [8] allows the allocation bias to adapt to workload behavior, but requires complex hardware tables and protocol modifications. R-NUCA [26] avoids cache block migration in favor of intelligent block placement, but distributes shared data across the entire die. Huh advocates NUCA organizations with static block-mapping and a small sharing degree [27], but the mapping is based on the block's address and is oblivious to the data access pattern. Kandemir proposes migration algorithms for the placement of cache blocks [33] and Ricci proposes smart lookup mechanisms for migrating blocks in NUCA caches using Bloom filters [59]. PDAS [81] and SP-NUCA [45] propose coarse-grain approaches of splitting the cache into private and shared slices. However, none of these works optimize for power and energy. Nahalal [25] builds separate shared and private regions of the cache, but the block placement in the shared cache is statically determined by the block's address. Finally, Page-NUCA [13] dynamically migrates data pages to different nodes whenever the system deems it necessary, but requires hundreds of KB to MB of extra storage which scales linearly to the number of cores and cache banks, and complicated hardware mechanisms and protocols. Overall, all these schemes place data blocks and optimize for performance; PAD places directories and optimizes for power and energy. Thus, PAD is orthogonal to them and can be used synergistically. Moreover, PAD does not require complex protocols or hardware.

OS-driven cache placement has been studied in a number of contexts. Sherwood proposes to guide cache placement in software [65], suggesting the use of the TLB to map addresses to cache regions. Tam uses similar techniques to reduce destructive interference for multi-programmed workloads [72]. Cho advocates the use of the OS to control partitioning and cache placement in a shared NUCA cache [31]. PAD leverages these works to guide the placement of directories using OS mechanisms, but, unlike prior proposals, places directories orthogonally to the placement of data.

Several proposals suggest novel coherence mechanisms to increase the cache performance. Dico-CMP [61] extends the cache tags to keep sharer information. Zebchuk proposes a bloom-filter mechanism for maintaining tagless cache coherency [82]. These proposals are orthogonal to PAD, which co-locates the directory with a sharer, as opposed to changing the cache coherence protocol.

Zhang observes that different classes of accesses benefit from either a private or shared system organization [84] in multi-chip multiprocessors. Reactive NUMA [22] dynamically switches between private and shared cache organizations at page granularity. Marchetti proposes first-touch page placement to reduce the cost of cache fills in DSM systems [36]. Overall, these techniques optimize performance in DSM systems. In contrast, PAD introduces a mechanism that decouples a block's address from its directory location, allowing the directory to be placed anywhere on chip, without space or performance overhead, and without complicating lookup. PAD uses

this mechanism to minimize power and energy in CMPs. Finally, the first-touch directory placement policy is only used because it is simple and effective, and is orthogonal to the PAD mechanism.

Power-aware on-chip interconnection networks for CMPs:

Several studies optimize power for on-chip interconnects. Balfour optimizes router concentration for higher energy efficiency [6]. Wang proposes circuit-level techniques to improve the power efficiency of the link circuitry and the router microarchitecture [76]. Shang proposes dynamic voltage and frequency scaling of interconnection links [62], dynamic power management [36, 63], and thermal-aware routing [64] to lower the power and thermal envelope of on-chip interconnects. All these techniques are orthogonal to PAD, which focuses on reducing the overall hop count and number of network messages by changing the directory placement. PAD can work synergistically with prior proposals to lower power and enhance the energy efficiency even further.

Chapter 3

Mitigating the Effects of Process Variations: Profitability

We investigate the effects of cache resizing schemes on batch-performance and revenue, and then propose a novel scheme called Substitute Cache (SC) that aim at improving overall binning distribution with post-fabrication modifications. Next, we study the impact of two resizing schemes namely, One-Way Sizing (OWS) and Multi-Way Sizing (MWS). OWS disables selected cache word lines with critical or near-critical delay. By disabling the high-latency word lines after manufacturing, this approach improves the yield at the low end of the frequency distribution, and also increases the likelihood that any valid chip will be placed in a higher-frequency bin. MWS extends this idea to a word line as it spans multiple sets in the cache, working off the theory that a high-latency word line in a single cache set would also likely be a critical path in the other sets. In addition to these schemes, we also propose the SC scheme, in which the level 1 (L1) cache is augmented with a small substitute cache storing the most critical cache words. With the help of minimal control logic, the

processor can fetch data from SC instead of the main data array whenever a read/write access is made to these critical words. Hence access latency is minimized with no extra cache misses.

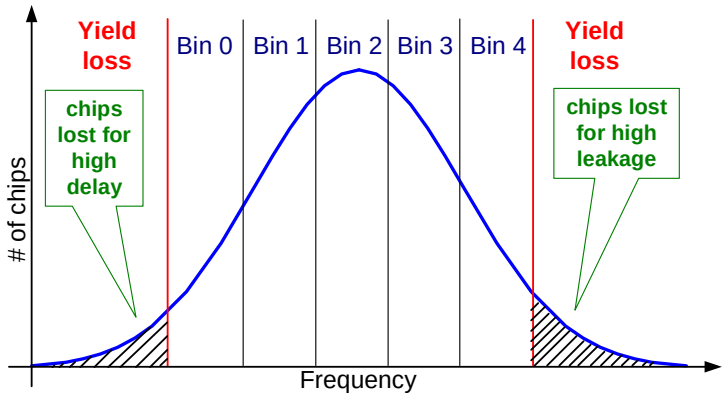

Figure 3.1. Speed binning in modern processors.

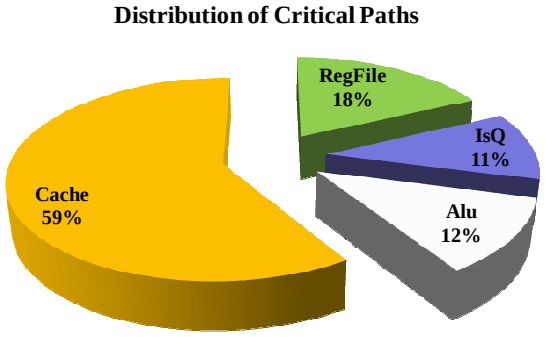

Figure 3.2. Distribution of processor critical paths to modeled architectural units.

The reason why we emphasize on caches is because L1 caches are likely to be the critical path under process variations. Figure 3.2 illustrates the latency distributions of various architectural units; for a set of 2000 simulated chips (the details of the

modeling framework are described in Chapter 3.3). The analysis reveals that 58.9% of the critical paths lie in the L1 cache. Therefore, in this work, we focus on the level 1 cache.

3.1 The Substitute Cache Scheme

In this section, we describe our proposed cache redundancy technique called Substitute Cache (SC), which masks the effects of process variations by including extra storage in the L1 cache. Particularly, in this scheme each way of the L1 cache is augmented with a fully associative data array, which stores the most critical lines as a result of process variations. Once the SC holds high-delay words, they are never accessed from the main array, allowing the L1 cache to run at higher frequencies. In addition to shifting more chips towards high-priced bins, this scheme also reduces yield losses due to delay violations.

The core idea behind the architecture of SC is to augment each cache way with extra storage that will be used if certain locations in the main cache exhibit long latencies. In such cases, the data will be read from the SC, and chips from the lower frequency bins can now be placed in higher frequency bins, because the high latency lines are not used. Moreover, some of the chips that would have failed due to high access latencies will be added to the overall yield.

The anatomy of the proposed cache architecture is shown in Figure 3.3. SC is highlighted within the dashed block. For the sake of clarity, we detail the use of SC on a single cache way; however, each cache way has a similar SC associated with it. SC is similar to a fully-associative cache structure. In our study, its size is either 4 or 8 entries. As opposed to the L1 cache, SC has smaller line sizes. Particularly, it consists of only 64-bit entries, because it stores words of the main data array. Instead of storing the whole cache line, only the critical word in the line is stored in the SC, because our study reveals that the words with maximum access latency are always the ones that are furthest from the decoder. As a result, by just storing these words, we obtain the same improvement in cache frequency while keeping the SC size small. However, if necessary, words in other locations can also be placed into the SC. An SC is divided into 2 components: an index table and a data array. Note that the SC uses the column multiplexers and output drivers of the main array. Whenever a cache word is placed in the data array of the SC, index bits of its address, which is equal to the sum of the row and column addresses (10 bits in our architecture) are placed in the index table of the SC. For example, if we decide to place the word with index value 0x044 to the SC, we will have an entry in the index table with value 0x044. Note that this word would have resided in the row with index 0x8 in the main array with the column address being equal to 0x4. In case of a data access, the index table is checked with the index bits of the address. A match implies that the data will be read from the SC instead of the main

array. Specifically, if the index of the address is found in the SC index table, the contents of the corresponding data array row are forwarded to the column multiplexers of the main array. The additional control logic shown in Figure 3.3 will then set the column multiplexers correctly. If the index of the address does not match any index table entries, the main array will be accessed. Note that, even if there is a match in the index table, the access can still miss in the cache if the corresponding tag does not match. However, the tag structure is not affected by the addition of the SC. If there is a miss due to tag mismatch, we will still output the data, which will be ignored because the tag will indicate the miss. Overall, the tag match/mismatch is independent of the SC design. We only care whether the corresponding parts of the address match with the values stored in the index table so that we can decide whether to supply the data from the main array or the SC.

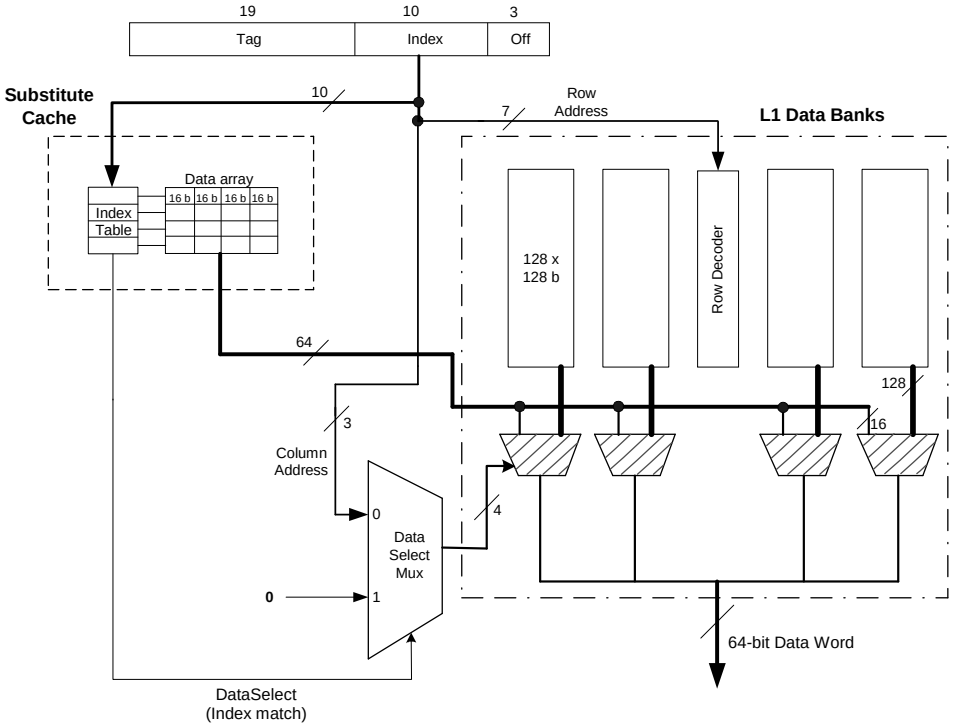

Figure 3.3. One cache way of a 32KB 4-way set associative L1 cache with Substitute Cache. Column muxes are shaded as they select data from 9 inputs as opposed to 8 inputs.

Now let us consider a typical read operation in the main array. The row address part of the index field selects the appropriate row in the data array through the row decoder. The appropriate word is then chosen by the column multiplexers with the help of the column address bits of the index. One of the key observations is the difference between the times taken by each of these steps. Particularly, the inputs to the column multiplexers are available at the same time the decoder is accessed. However, the signals provided to the decoders will traverse through the decoder logic, the word lines,

the memory cell, the bit lines, and the sense amplifiers before it will reach the column multiplexers. We utilize this imbalance to operate our SC structure. As soon as the address is available, we start accessing the SC index table. If a hit is recorded, we change the input to the column multiplexers to 0. In other words, we forward the output of the SC as the output of the cache. If, on the other hand, there is no match in the index table, we will set the column multiplexer to the original position indicated by the column address. If the time to check the index table in the SC is less than the delay of the data array (the sum of the delays of the decoder, word line, memory cell, bit line, and sense amplifier), then, this operation does not cause any delay overhead on the cache, because while the data array is accessed, we would have already determined the hit/miss in the SC index table. Hence, the addition of the SC structure does not cause any significant increase in the critical path latency of the cache.

Similar to a read operation, a write access (either a store operation or write operation during the replacement of a cache line) selects the appropriate index using the row and column addresses and updates the selected word in the cache way selected by the way-select logic. For L1 caches augmented with SC, the index of the data word to be written is searched within the SC index table. If there is a match, the new data word is loaded in the data array of the SC. We must also note that the addition of the SC does not impact the tag (and any related operation including snoop requests).

3.2 Cache Resizing Schemes

The main idea in these schemes is to analyze the design of the cache, determine the word lines that can cause a delay violation and then modify the architecture of the cache such that these word lines may be disabled. In the core of these ideas lies one common characteristic: if a path is found to be a critical path in the cache, it will be the critical path in a large number of chips. In general, when process variations are considered, it is hard to determine a single path that is the critical path. Therefore, each path is associated with a probability of being a critical path. If this probability is X%, the corresponding path is expected to be the critical path in X% of the manufactured chips. In our cache model, we have observed that these probabilities can be very high. Particularly, our analysis of the cache architecture and the process variation simulations reveal that one particular word line is the critical path in 67.3% of the 1000 caches we have studied. The reasons for this phenomenon are two-fold. First, cache architectures are regular; most paths exhibit the same characteristics. Second, because of spatial correlation in process variations, all the word lines are affected similarly. The consequence of this phenomenon is crucial: if we select a word line to be the critical path during the design process, it will be the critical path in many chips and hence disabling it may reduce the overall cache delay.

One-Way Sizing (OWS): As the name suggests, One-Way Sizing (OWS) refers to the cache resizing scheme when resizing is restricted to a single cache way. The main

idea in OWS is to disable word lines that are likely to generate cache delay violations or cause the chip to be placed in the lower bins. Take for example the 4-way set associative cache described in Chapter 1, Section 3.2. If due to the effects of process variations the incurred extra delay makes the cache very slow, then turning off the delay-intensive line will be helpful in decreasing the cache latency. As a result, the chip can be placed in a higher-frequency bin. Our OWS scheme is based on this concept. Particularly, we first analyze the cache architecture and determine the critical paths. Each critical or near-critical path corresponds to a word line. Then, we select n such paths and change their word line select bit logic to allow the designer to disable them (i.e., turn them off). The number of cache rows or word lines to be disabled depends on the cost and overhead the designer is willing to allow. For example, OWS-4 refers to disabling the 4 most critical word lines of the cache. Note that, to simplify the process of disabling, we do not allow each line to be turned on/off individually. On the contrary, all the selected lines are enabled/disabled together. To clarify the process, consider the process of developing the OWS-8 scheme. For OWS-8, we first analyze the delay of all word lines in a cache way. In our cache architecture, there are 128 such lines; hence, we order them according to their expected latency. Then, we choose the topmost 8 and change their word line select logic. This can be performed by adding an additional input to the AND gates that activate the local word line select signals. This additional input is used for the enable signal. The enable signals for all the 8 word lines are connected to

the same “resize enable” signal. After the manufacturing, using this enable signal, the designer can choose to disable all the selected 8-rows. If one of these word lines is the critical path, the total delay of the cache will be reduced. As a result, the chip may be placed in a higher bin.

Note that, in OWS, each cache way has a separate “resize enable” signal. As a result, the speed-binning process after the manufacturing needs to be changed to test the overall delay while each of these signals is asserted. Although it is possible to control each enable signal (and hence the cache way) individually, the number of possible combinations can be large. In addition, if several word lines corresponding to the same index are disabled, the associativity for those indexes may decrease, potentially resulting in a large number of cache misses. Therefore, we allow at most one set of disabled words lines. In other words, only selected word lines from one cache way can be disabled at a given time. To implement this, each “resize enable” signal will be asserted sequentially during the testing stage, and one signal will be allowed to remain high if this changes the outcome of the speed-binning.

Multi-Way Sizing (MWS): OWS aims to locate the likely critical paths in a cache and embed enable/disable signals for them, so that these word lines can be disabled. However, if a word line is the critical path in one of the cache ways, it is very likely to be the critical path in the remaining ways. As a result, although OWS can disable one of these paths, the remaining ones will still be enabled and cause a long

cache access delay. Another drawback of the OWS scheme is the increased complexity due to the “resize enable” signals in each way. The Multi-Way Sizing (MWS) technique aims at attacking these limitations. Particularly, MWS disables all the chosen critical word lines from all the cache ways instead of disabling the word lines in a single cache way as done in OWS. To explain the idea, consider that word line N is determined to be the most likely critical path. Then, similar to the OWS scheme, MWS will change the AND gates on word line N to allow it to be disabled. However, unlike OWS, MWS will allow the designer to disable all word line N’s from all the cache ways simultaneously. If the word line N is the critical path in all the cache ways, this will eliminate the longest path in each way and cause a significant reduction in the cache delay. Because of the spatial correlation of process variations, the probability that the same index remains the critical path in different cache ways is high; in these cases MWS improves upon OWS.

A second advantage of the MWS scheme is the reduction in the number of enable/disable signals. Since the decision of enabling/disabling is done for the whole cache, the cache will implement a single “resize enable” signal as opposed to one for each way in the case of OWS. This will reduce the complexity of the control circuitry.

For MWS, similar to OWS, the designer has to select the number of rows that will implement the enable/disable signals. If 4 word lines from each cache way are

connected to the “resize enable” signal, the scheme is called MWS-4. Note that, this corresponds to disabling 16 word lines simultaneously for a 4-way cache.

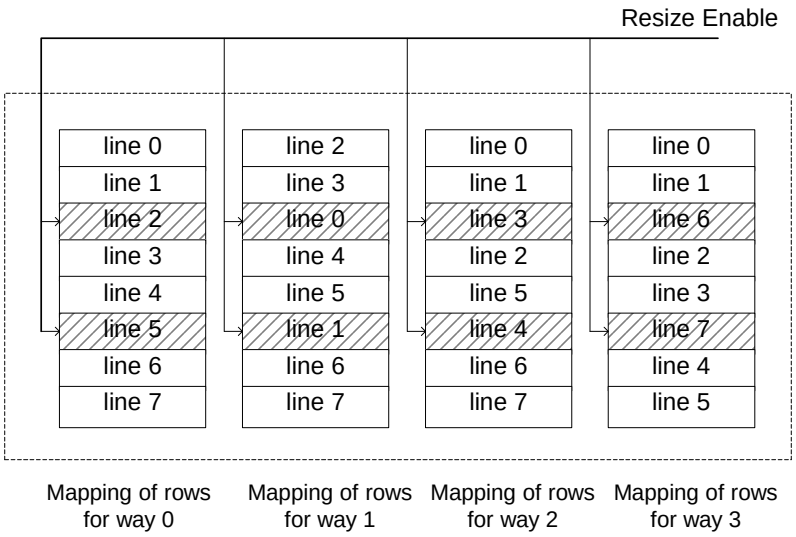

Figure 3.4. The mapping of indexes to word lines for MWS. Straddled blocks show the lines disabled by the “resize enable” signal.

A problem with the MWS technique is that when a cache line is disabled across all the ways, that index loses its address space. For the above-presented example, if we decide to disable all the word lines N, then any addresses with the corresponding index will miss in the cache. To tackle this issue, the orientations of the decoder lines are changed in such a manner that no identical indexes are disabled in two different ways. To be precise, we modify the mapping of indexes to word lines in each cache way such that each index can be disabled at most once. Figure 3.4 presents the change of the mapping for a 4-way, 32-entry cache (8-entries for each way). The initial word lines to

be disabled are found using the delay analysis. In our example, these are lines 2 and 5. Then, for the remaining cache ways, the rows to be disabled are found by considering the lowest row number that has not yet been placed into a disabled line. In our example, these are lines 0 and 1 for cache way 1, lines 3 and 4 for cache way 2, and lines 6 and 7 for cache way 3. The remaining rows are mapped to remaining index numbers in order. As a result of this reordering, when the cache is resized, the associativity for each index reduces by at most one. Particularly, for our example architecture, each index has exactly 3 enabled rows; hence, the cache miss rate will be identical to that of a 3-way associative cache with 24 total entries.

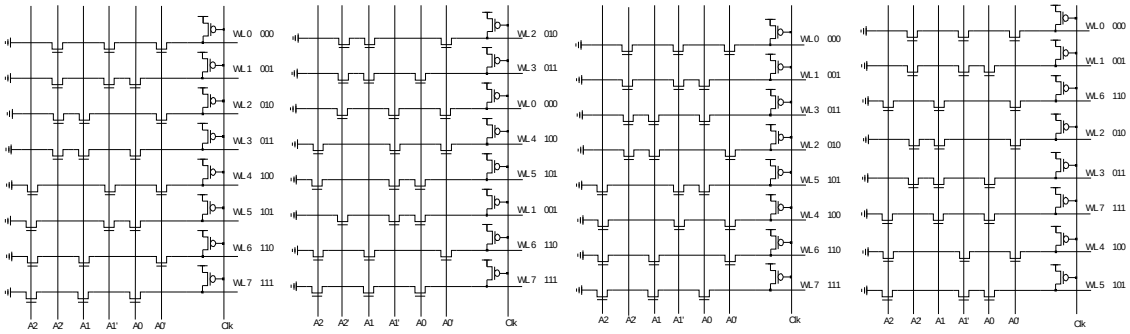

Figure 3.5. Post-decoder implementation for changing the index to word-line mapping.

This remapping of the indexes to word lines can be implemented by changing the post-decoders that are implemented in high-performance caches. Particularly, the decoders in modern caches work in two stages: a pre-decode and a post-decode stage. The pre-decode stage generates a number of signals and broadcasts them to each word line, where the post-decoders are waiting for certain combinations. The new mapping of

the indexes to word lines can be implemented by simply changing these combinations. The post-decoder implementations for the cache architecture shown in Figure 3.4 are depicted in Figure 3.5. The signals A0, A0', A1, A1', A2, and A2' are produced by the pre-decoder. The select logic (i.e., transistors on the word line select logic) for each word line corresponds to the post-decoder stage. As shown in the figure, by simply reordering the locations of these transistors, we achieve the desired reordering. Note that this change does not incur any penalty on the delay of the cache.

Complexity of Resizing: Both our MWS and OWS schemes have design overheads. Note that we implement the enable signals on the critical paths of the cache; hence any change, due to our schemes, increases the cache delay. The particular modification we make to the cache is to change the 2-input AND gate that enables the word line select signal to a 3-input AND gate. We found that the delay overhead for this extra circuitry is 0.75% on average. This increase in delay has an impact on the binning of the chips. However, we must note this overhead is not applicable to MWS when these lines are disabled. To be precise, when the selected word lines are disabled, they will never be used throughout the lifetime of the chip. Therefore, the delays of these lines are not considered during the critical path analysis, hence the overall delay is not affected for MWS. For OWS, on the other hand, this increase in delay may have an impact on the cache delay. Since OWS disables word lines in only one of the cache ways, the delay of

the word lines in the remaining cache ways may increase, which in turn will increase the critical path delay.

Effects of MWS and OWS on Cycles-per-Instruction (CPI): Since we are performing cache resizing, our schemes may increase cache miss rates, which will result in performance degradation. We analyze how our schemes change the cycles-per-instruction (CPI) for the SPEC2000 applications. SimpleScalar 3.0 [67] simulator is used to measure the effects of our proposed cache resizing techniques. The necessary modifications have been implemented on the base simulator to model selective cache replay, the buses between caches, and port contention on caches. Changes were also made to SimpleScalar to implement the cache resizing schemes, which disable certain indexes from corresponding cache ways. The base processor is a 4-way processor with an issue queue of 128 entries and a ROB of 256 entries. The simulated processor has disjoint level 1 data and instruction caches: level 1 data cache is a 32 KB, 4-way set associative cache with block size of 64-bytes and latency of 4 cycles, and the level 1 instruction cache is a 32 KB 4-way set associative cache with block size of 32-bytes and latency of 2 cycles. The level 2 cache is a unified 1024 KB, 8-way set associative are cache with 128 byte block size and 20 cycle latency. The memory access delay is set to 350 cycles. We have performed our simulations using 11 floating point and 12 integer benchmarks from the SPEC2000 benchmarking suite [69].

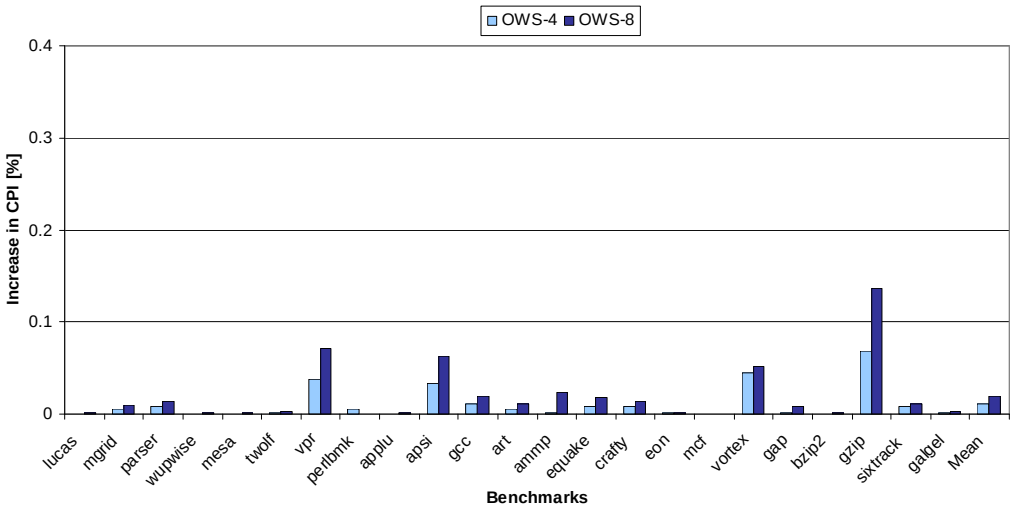

Figure 3.6. Performance results for OWS schemes for the SPEC2000 applications.

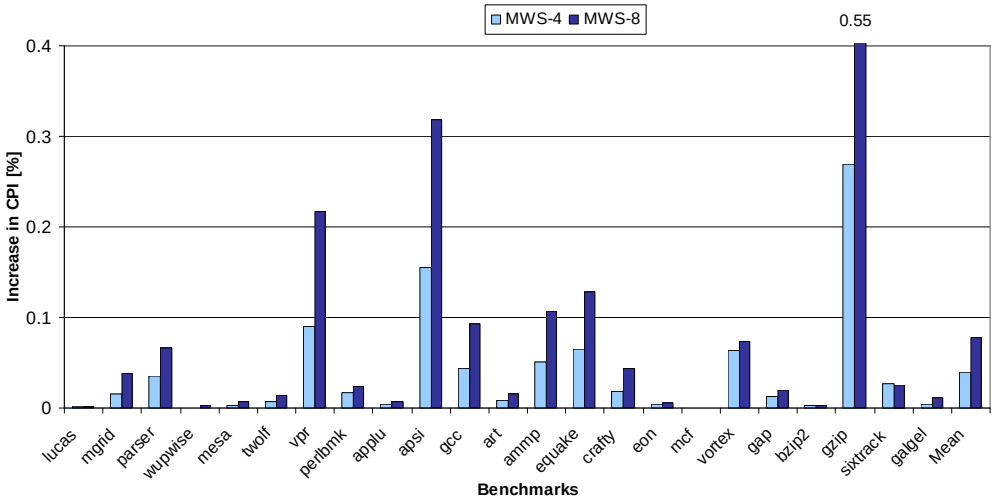

Figure 3.7. Performance results of MWS schemes for the SPEC2000 applications.

Figure 3.6 and Figure 3.7 present the increase in CPI for the MWS and OWS schemes, respectively. The average increase in the CPI is 0.08% for MWS-8 and 0.02% for OWS-8 schemes. Among the studied applications, only two exhibit an increase in

CPI exceeding 0.3%: gzip and apsi. For these applications, the increases in CPI for the MWS-8 scheme are 0.55% and 0.32%, respectively.

Note that the schemes disable sporadic indexes, and hence different indexes have varying associativity, creating heterogeneous cache architecture. Therefore, the increase in the CPI for these two applications is directly caused by their usage of the disabled indexes.

3.3 Modeling Process Variation and Speed Binning Methodology

This section presents a detailed description of our framework, which consists of the processor model, process variations model and models for generating a speed-binning distribution. Each one is described below in detail.

Processor Model: To model a processor core, we have taken into account the 7-stage pipeline in the Alpha-21264 (EV6) architecture. The main critical components of our processor are the Fetch Unit, the Rename Unit, the Issue Queue, the Integer Execution Unit, the Register File, and the L1 Data cache. All these components are modeled in SPICE using the 45nm BPTM technology models [11]. The fetch and rename units are modeled as a combination of 16 fan-out of four (FO4) gates. The issue queue is based on that of the Alpha EV6 and has 20 entries. The register file is an 80-entry structure with 4 read and 2 write ports. The integer execution unit is modeled

using the net list generated after synthesizing the corresponding component in the Sun OpenSPARC code [70]. Our L1 cache is a 32 KB 4-way set associative cache, the model of which is based on the architecture described by Amrutur and Horowitz [4].

Figure 3.8 highlights a cache way for our base cache model. Each of the 4 ways is divided into 4 banks. Each bank has 128x128 cells or storage bits. Thus, each bank has exactly 128 rows (i.e., lines) and can hold 2-KB of data. The bit line delays are reduced by partitioning the bit line into two. We must note that our SPICE models are based on highly optimized circuit descriptions (e.g., the cache model is based on CACTI 3.2). To account for the effects of submicron technologies on circuit behavior, we added coupling capacitances at three places in the cache: between the lines in the address bus from the driver, between parallel wires in the decoder, and between bit-line and bit-line bar. Furthermore, these lines as well as global and local word lines are replaced by distributed RC ladders representing the local interconnect wires inside the cache. Although the L2 cache is another SRAM structure within the processor core where process variations can have a significant impact according to the FMAX theory, we omitted this component in our study because it doesn't lie on the processor critical path (and is not a part of the processor pipeline) so other techniques like high-threshold transistors or NUCA caches can be utilized to mitigate the effects of process variations on them.

Simulating Process Variations: Process variations are statistical variations in circuit parameters like gate-oxide thickness, channel length, Random Doping Effects (RDE) etc., due to the shrinking process geometries [9, 50]. They mainly consist of die-to-die (D2D) and within-die (WID) variations. D2D variation refers to the variation in process parameters across dies and wafers, whereas WID variation is the variation in device features within a single die, causing non-uniform characteristics inside a chip. Independent of their type, process variations generally fall into two categories: spatially-correlated variations where devices close to each other have a higher probability of observing a similar variation level, and random variations causing random differences between devices within a die.

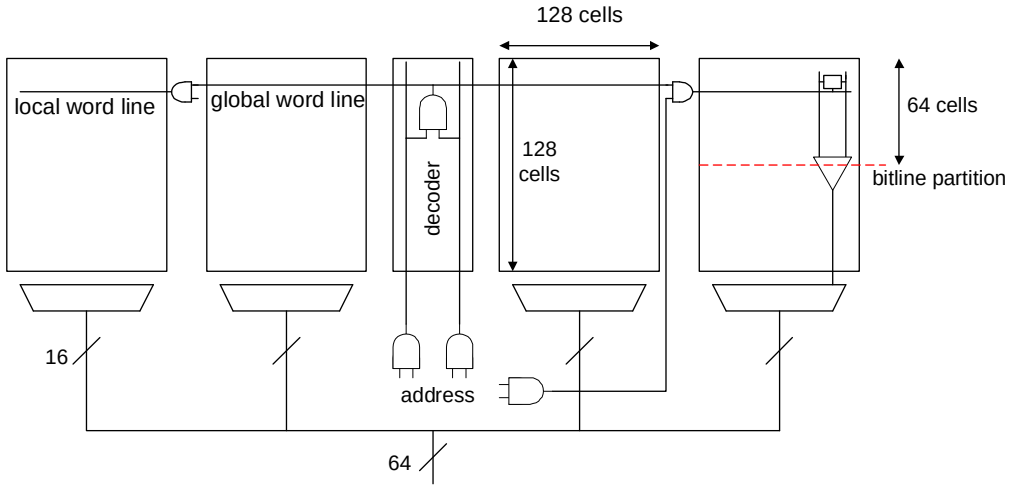

Figure 3.8. A single cache way for a 4-way set associative L1 cache.

Table 3.1. Nominal and 3σ variation values for each source of process variations modeled

	Gate Length (L_{gate})	Threshold Voltage (V_{th})	Metal Width (W)	Metal Thickness (T)	ILD Thickness (H)
Nominal Value	45 nm	220 mV	0.25µm	0.55µm	0.15µm
3σ - Variation [%]	±10	±18	±33	±33	±35

To measure the impact of process variations on the delay and leakage of our cache model, we considered 5 most important variation parameters. These are metal thickness (T), inter-layer dielectric thickness (ILD or H), line-width (W) on interconnects, gate length (Lgate) and threshold voltage (Vth) for the MOS devices. The statistical distributions of these parameters are based on limits given by Nassif [49] and their statistical distribution (mean and variation) are listed in Table 3.1.

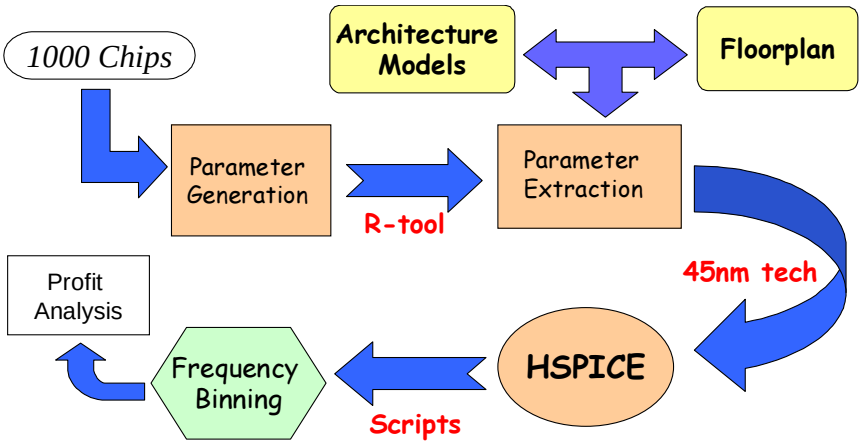

Figure 3.9. Monte Carlo SPICE simulation framework.

We model both systematic and random process variations for our processor model. To take into account the spatial correlation we use a range factor (φ) in the two dimensional layout of the chip. Thus, each process parameter can be expressed as a function of its mean (μ), variation (σ), and the range (φ) values. For the sake of simplicity we use the following inverse linear function to minimize computational time.

$$C_i = 1 - \frac{d_i}{\varphi} \qquad (1)$$

Equation 1 describes the spatial correlation function we used for our framework. If two points, x_i and y_i on a 2D plane are separated by a distance d_i, then the spatial correlation factor C_i between them can be thought of as an inverse linear function involving φ and d_i. Note that there is no correlation between two spatial points which are φ units or more apart.

With this background, we have generated a spatial map of various parameter values using the R statistical tool [74]. This spatial map indicates that φ is a measure of randomness; a higher φ will mean higher correlation and vice versa. To extract the parameter values corresponding to the different functional units, we use the floorplan of Alpha EV6 processor. In other words, the process variation values for the chip were generated first, followed by the extraction of the values that correspond to the particular positions of the studied components from this modeled chip. Note that all our components consist of other smaller components. For example, to model the cache, we pick different process variation values for the decoder, each cache line, pre-charge

logic, etc. In addition to the spatial variation, we also model random variations in the process parameters. To model them, we chose process parameters randomly from a uniform distribution. Since spatially correlated process variations are found to be the dominating factor [24], our framework assumes a higher percentage of spatially correlated variation compared to random variations. We set this ratio as 70% to 30% for correlated and random variation, respectively. Figure 3.9 shows the Monte Carlo simulation framework used in our parameter generation and extraction experiments.

Modeling Speed Binning: In order to effectively estimate the binning distribution and demonstrate the effect of process variations on it, we chose a set of 1000 chips for our analysis. Using the process parameters mentioned above, their delay and leakage current values are obtained from SPICE simulations for the cases when φ=0.3 and φ=0.5, which in turn are used to determine the binning and yield loss. The cut-off for delay has been set to be the sum of the mean (μ) and standard deviation (σ) of the delay of the simulated chips (i.e., $\mu + \sigma$), whereas the leakage cut-off has been set to be three times the mean leakage value. These cut-off limits are based on previous studies [56].

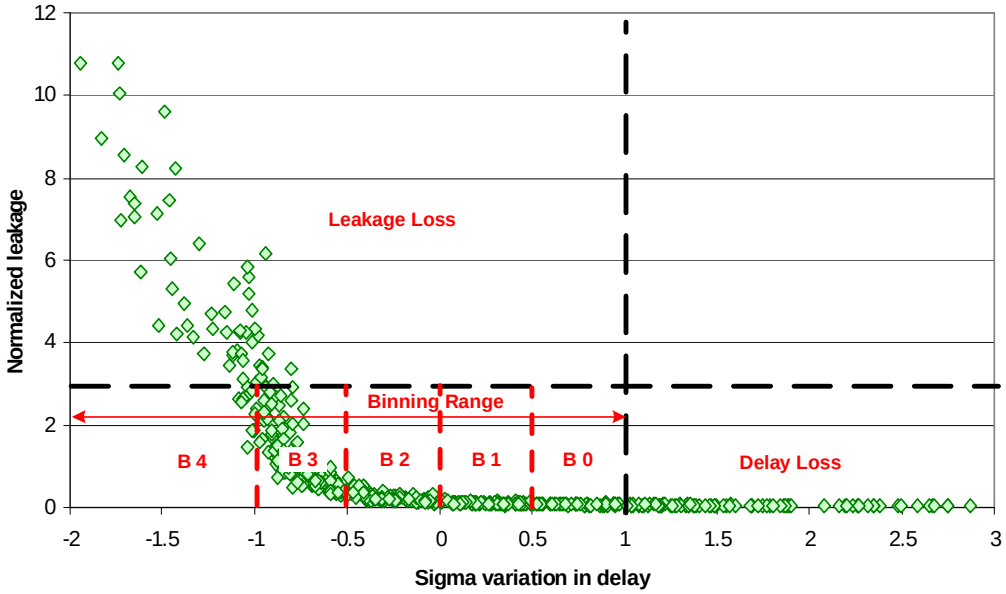

Figure 3.10. Normalized leakage and delay distribution scatter plot for simulated chips showing the binning for 5-bin strategy. B0 through B4 represent the bin numbers from lowest to highest frequency.

Most processor families are available in discrete frequency intervals. For example, the frequency for the Intel Pentium 4 processor family starts with 3.0 GHz and reaches 3.8 GHz with equal intervals of 0.2 GHz [30]. Moreover, most commercial processors are marketed with 5 or 6 different frequency ratings. Similarly, our binning methodology assumes equal binning intervals. This interval is chosen depending on the number of bins to be generated. Regardless of the number of bins, any chip that has a delay greater than the '$\mu + \sigma$' limit is referred to as a delay loss. Chips that satisfy this criterion are used for binning into discrete bins starting from the slowest to the fastest

bin. Within each bin, the chips that are lost due to excessive leakage (exceeding the limit of 3x mean leakage) are referred to as the leakage loss. Figure 3.10 shows the distribution of the normalized leakage power consumption versus the distribution of processor latencies for the base case (i.e., without any architectural optimizations) for the 1000 simulated chips for φ value of 0.5. It also shows the binning for a strategy that generates 5 distinct bins. In this case, the chips that lie within '$\mu + \sigma$' and '$\mu + 0.5\sigma$' delay values are put into Bin0 (denoted by B0 in Figure 3.10). These correspond to the slowest chips. Similarly, chips with latencies within '$\mu + 0.5\sigma$' and 'μ' are assigned to Bin1. The intervals for the remaining bins are set following the same '0.5σ' interval. Note that the highest bin consists of the chips with delay values less than '$\mu - \sigma$'. Using a similar methodology, we model a strategy that generates 6 bins. In this case, we reduce the binning interval to '0.4σ'. Hence, Bin0 consists of chips that fall between '$\mu + \sigma$' and '$\mu + 0.6\sigma$', Bin1 consists of chips that fall between '$\mu + 0.6\sigma$' and '$\mu + 0.2\sigma$', and likewise.

3.4 Experimental Results

In this section, we describe the analysis of our proposed schemes.

Binning Results: Since our binning schemes are divided into two categories, namely 5-bin and 6-bin strategies, we describe them separately. For both 5-bin and 6-bin strategies, the proposed MWS, OWS, and SC schemes are applied and the resulting

changes in the number of chips in each bin are found. To find how the chips are placed into different bins, we first analyze our architecture with the base cache and find the mean and standard deviation of the 1000 cache delays. Then, based on these values, the boundaries for each bin are set. We then apply the MWS, OWS, and SC to find the new delays for each chip and find the corresponding bin distribution.

Figure 3.11 and Figure 3.12 show the binning results for 5-bin strategy for MWS and OWS, respectively. The results for the 6-bin strategy were similar and hence are not presented in detail. To understand the figures, consider the leftmost bar for each bin. This bar corresponds to the number of chips in that bin for the base cache architecture. The bars next to it (i.e., the ones in the middle) represent the number of chips in that bin when MWS-4 or OWS-4 schemes are applied. The right bars represent the number of chips in the corresponding bin for the MWS-8 or OWS-8 schemes. In general, we see that our schemes can successfully increase the number of chips in the higher bins. For example, in the 5-bin strategy, the number of chips in the highest bin (Bin4) is increased by 8.2% using MWS-8. Figure 3.13 depicts the binning results for the SC scheme. In case of the SC, the chip yield is catapulted to a larger extent (14.4%). Like MWS and OWS, it also shows a sharp increase in the chip of the last bin for the 5-bin strategy.

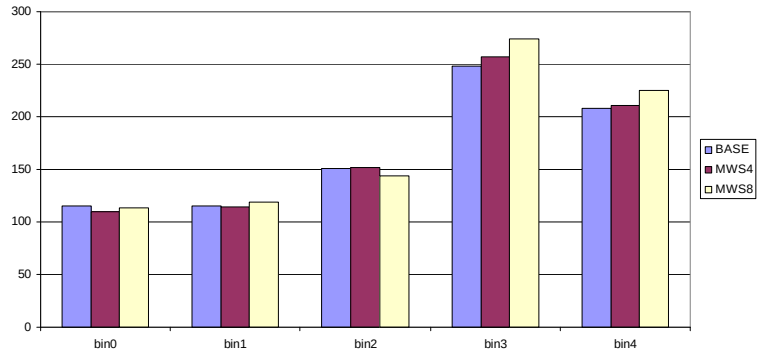

Figure 3.11. Binning with 5-bin strategy for MWS.

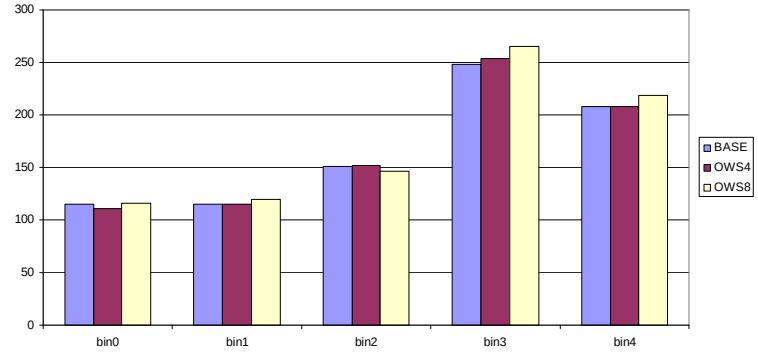

Figure 3.12. Binning with 5-bin strategy for OWS.

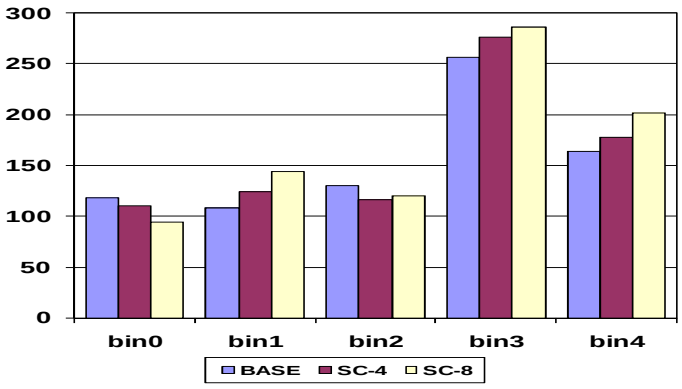

Figure 3.13. Binning with 5-bin strategy for SC.

It is misleading to draw any conclusion about high-frequency chip yield by simply considering the chips in the highest bin. The gain in the highest bins for all the 3 schemes are accompanied by a reduction in the number of chips in the lower bins. However, we must note that the total yield is increased using these schemes. Specifically, the total yield increases by 4.5%, 3.5% and 9.7% using the MWS-8, OWS-8, and SC-8 schemes, respectively (for φ=0.5). Although there are no additional chips lost due to leakage for the resizing schemes, the SC is associated with a power overhead. The SC-4 and SC-8 schemes cause an additional 9.1% and 11.7% loss of chips, respectively. In spite of that, the total yield increases for SC, because it converts a high number of delay loss chips into yield. Even though the total number of chips increases, the schemes tend to move a larger number of chips towards the higher bins. As a result, the chip counts in the lower bins tend to decrease.

One of the reasons for the significant change of yield gain from MWS-4 to MWS-8, OWS-4 to OWS-8, and SC-4 and SC-8 is the fixed cost of implementing the schemes. As described in Section 3.2, implementing the resizing scheme incurs a circuit delay of 0.75% over the base cache architecture. When the “resize enable” signal is off, this overhead in delay is added to the critical path, whereas when it is on it does not affect the critical delay in the MWS scheme. Therefore, MWS has a more profound impact on the speed-binning outcome. In case of OWS, the overhead may cause other

cache ways to become the critical path, limiting its overall impact. For SC, this overhead is even lower and hence it achieves better binning results than MWS.

Revenue and Batch Performance (BP) Estimation: This part describes the analysis of the total revenue and the implications on BP and chip revenue. It is important to note that, in all the following studies a simplistic market supply/demand model is assumed where all fabricated chips can be sold at predicted/predetermined price levels according to their clock frequencies. Since a real-life demand model would depend on various other factors, the resulting numbers given here should be considered as potential increase in revenue or profit. The binning data is used in revenue calculation. The chips that fall in the higher/faster bins after testing are sold with higher prices than those lying in the lower/slower bins. To have an estimate of this increased revenue, we use the model that provides the highest accuracy among the models studied. Our architectural configuration is fed into our price models to find the relative prices of the chips in each bin. These relative prices are found to be 1, 1.03, 1.13, 1.39, and 2.84, for the Bin0 through Bin4 for the 5-bin strategy and 1, 1.02, 1.09, 1.23, 1.63, and 4.00, for the Bin0 through Bin5 for the 6-bin strategy. Then, the number of chips in different bins for the base case is multiplied with their respective prices to calculate the revenue for the base case. Using the same methodology, the revenue for SC-4 and SC-8 schemes are calculated based on their new binning distributions. The relative change in revenue is then calculated with respect to the revenue of the base case.

Table 3.2 presents the increase in revenue obtained using different SC schemes. For φ=0.5, the SC-8 scheme increases the revenue by up to 12.60% and 13.14% for the 5-bin and 6-bin strategies, respectively. Note that, the SC scheme has power consumption overhead and hence causes some power-related yield losses. However, despite the increase in the power consumption, we are observing that the SC scheme tends to provide better revenues because it is able to generate an elevated number of chips in higher bins. We must note that the increase in revenue is smaller compared to the increase in the number of chips in the highest bin. Take for example the 6-bin case. For SC-8, a 15.0% increase in the number of chips in the highest (i.e., highest-priced) bin results in an increase of the total revenue by only 11.4%. The main reason behind this can be explained as follows. Due to the normal distribution nature of the binning curve, the yield in the next-highest bin is higher. This bin also has a high price gradient and hence it constitutes a large fraction of the overall revenue. We observe that the number of chips in this bin either reduces or stays roughly constant. As a result, the increase in total revenue is limited by a moderate percentage.

Table 3.2 presents the increase in revenue obtained using different SC schemes. For φ=0.5, the SC-8 scheme increases the revenue by up to 12.60% and 13.14% for the 5-bin and 6-bin strategies, respectively. Note that, the SC scheme has power consumption overhead and hence causes some power-related yield losses. However, despite the increase in the power consumption, we are observing that the SC scheme

tends to provide better revenues because it is able to generate an elevated number of chips in higher bins. We must note that the increase in revenue is smaller compared to the increase in the number of chips in the highest bin. Take for example the 6-bin case. For SC-8, a 15.0% increase in the number of chips in the highest (i.e., highest-priced) bin results in an increase of the total revenue by only 11.4%. The main reason behind this can be explained as follows. Due to the normal distribution nature of the binning curve, the yield in the next-highest bin is higher. This bin also has a high price gradient and hence it constitutes a large fraction of the overall revenue. We observe that the number of chips in this bin either reduces or stays roughly constant. As a result, the increase in total revenue is limited by a moderate percentage.

Table 3.2. Increase in revenue for various SC configurations

Range factor (φ)	Binning strategy	Increase in revenue with respect to the base architecture [%]	
		SC-4	**SC-8**
0.5	5-bin	5.03	12.60
	6-bin	3.90	11.41
0.3	5-bin	6.98	12.00
	6-bin	5.54	13.14

Using the same revenue results, we can also estimate profit. Let's assume that the cost per chip is identical, which equals to 80% of the selling price of the lowest frequency chip. This means, the cost of each chip is 0.8 in terms of our relative price.

Therefore, the total cost for 1000 chips (note that even the chips that do not meet delay or leakage constraints contributes to cost) is 800. We can then subtract this amount from the total revenues and find the profit. If we apply this methodology, we find that the SC-8 increases the profit in the 5-bin strategy by 46.6%. For a chip company, which invests billions of dollars in the manufacturing process, this extra revenue can prove to be a considerable margin. It should be noted we are neglecting the extra testing costs needed for the new cache design.

Comparison with Performance: To compare the effects of our architectural scheme on profit and performance of the whole batch of chips, we use another metric called batch performance (BP). Batch performance is calculated using the frequency of each speed-bin and the chip yield (number of chips) in that bin. Thus, batch performance corresponds to the overall performance of the chips obtained from a single batch of microprocessors. The BP metric is similar to utility metric defined by Romanescu et al. [60]. If there are k different frequency bins having frequency ratings f_1, f_2, …, f_k with each of them having yields n_1, n_2,…,n_k; the total batch-performance is given by:

$$BP = \sum_k (f_k \times n_k) \quad (2)$$

This BP formula can be extended in two ways. First, if an architectural scheme has an impact on the CPI, the change can be captured by incorporating it into the

equation. Specifically, if a scheme achieves an IPC of i_1, i_2, …, i_k for each bin, the new batch performance will be calculated by:

$$BP = \sum_k (f_k \times n_k \times i_k) \quad (3)$$

Finally, to find the average performance, this sum is divided to the number of manufactured chips. We have calculated the average BP for the base cache architecture and our proposed schemes based on Equation 3. Table 3.3 presents the increase in BP with different SC schemes.

Table 3.3. Increase in batch performance for various cache-architectures

Range factor (φ)	Binning Strategy	Increase in Batch Performance with respect to the base architecture [%]	
		SC-4	**SC-8**
0.5	5-bin	5.88	11.50
	6-bin	5.58	11.19
0.3	5-bin	6.18	10.51
	6-bin	6.10	11.59

A close look at Table 3.3 implies that the increase in batch performance is roughly linear with respect to the size of the SC. However, when the SC-4 and SC-8 architectures are compared, we see that the percentage improvement in revenue can increase by over 2.9x (Table 3.2). These results show that optimizing for performance alone may lead to different conclusions when compared to optimizing for revenue/profit

along with performance. Hence, these results motivate the use of revenue/profit when making architectural decisions.

3.5 Summary

Efficient binning under process variations has become a significant challenge for chip manufacturers. A considerable amount of effort is being made to save chips from excessive delay and market them properly to increase the profit margin. In this work, we evaluated cache resizing schemes and like One-Way Sizing (OWS) and Multi-Way Sizing (MWS). The extra circuitry needed for these schemes is very small and the newly resized cache causes minimal reduction in the instruction-per-cycle (IPC) rates: 0.02% and 0.08% on average for the most aggressive OWS and MWS, respectively. As an alternative to these resizing schemes we introduced a new cache architecture called the Substitute Cache (SC), which is aimed at maximizing the revenue obtained from a particular line of chips with the same process technology. Our scheme has no performance overhead and works by storing critical words of the data array in a separate structure. Extra circuitry needed for this technique is minimal and the modified L1 cache augmented with SC has no impact on the system performance. Moreover, to evaluate our architectural technique in the context of profit, we introduced models for estimating the price of processors from their architectural configurations and showed

that the estimation error rates are below 2% on average. Based on these models, we showed that the most aggressive SC scheme increases chip revenue by 13.1%.

Chapter 4

Mitigating the Effects of Process Variations: Power in CMPs

In this chapter of the thesis, we try to investigate the impact of process variations on CMPs and mitigate the power inefficiencies by using single/multiple voltage islands. Particularly, we make the following contributions:

- We develop an extensive model, which encompasses process variations for a CMP using statistical estimations and the detailed floor plan for Alpha EV7-like cores.
- We develop a variation-aware scheme for power optimization using single/multiple voltage islands across different cores in a CMP.
- We analyze varying voltage island granularities and show that depending on the technology, even a single voltage island can reduce the power consumption significantly.
- Finally, we formulate an analytical model that can be used to estimate the advantages of voltage islands for different manufacturing processes. Overall, our

results show that the multiple voltage island scheme results in up to 36.2% power reduction in our target architectures. A single voltage island, on the other hand, can save up to 31.5% of the dynamic power.

4.1 Multiple Voltage Island Scheme: Methodology

This section presents an overview of the power-aware multiple voltage islands scheme for CMPs. When processors are manufactured, they operate at a voltage level set during the design of the processor. This voltage level called the nominal voltage is usually chosen at the design time. However, under process variations setting a constant level for all the manufactured chips is considerably inefficient. First, different chips will have different latency slacks, which can be taken advantage of by customizing the voltage level for each chip. In addition, if we consider C2C variations, different cores will tend to have different latencies. In such cases, the operating frequency of the whole chip is determined by the maximum latency across all cores. It is known that the dependency of delay (D) or latency on the supply voltage is given by:

$$D \propto \frac{Vdd}{(Vdd-Vth)^{\propto}} \qquad (4)$$

where V_{th} is the threshold voltage and α is technology constant varying between 1 and 2. Equation 4 implies that cores which have latency lower than that of the slowest core (nominal delay), can increase their latencies by scaling down the V_{dd} in steps till they reach some minimum value. We refer to this voltage as the minimum stable supply

voltage (V_{opt}). Beyond this point the circuit operation fails. On the other hand, nominal supply voltage can be defined as the voltage set during design time which gives the desired latency for the set of manufactured chips. Our experiments indicate that for most cases the supply of one or more cores can be reduced below the nominal V_{dd} value. This optimization can significantly cut down static and dynamic power dissipation, hence lowering the energy of the whole system. Thus, in a multicore system a single core or a group of cores can be clustered on the basis of critical latencies and assigned a custom supply voltage. Such clusters with a customized V_{dd} can be referred to as a voltage islands. In general, if there are *k* voltage islands in a system having a nominal clock frequency of f_{clk} and a corresponding supply voltage V_{dd}, the dynamic and leakage power savings can be denoted by:

$$\Delta P_{dynamic} = C_{load} \cdot f_{clk} \cdot \sum_{k=1}^{n} (V_{dd}^2 - V_k^2) \qquad (5)$$

$$\Delta P_{statis} = \sum_{k=1}^{n} I_{leak_k} \cdot (V_{dd} - V_k) \qquad (6)$$

Where I_{leak_k} and C_{load} represents the average leakage current and load capacitance for each voltage island. Since our target CMP includes 16 cores, k can take values from 1 to 16. In the former case, the entire multicore system operates on a customized V_{dd}, while in the latter case each core has a different supply.

Since voltage islands can contain a single core or a collection of cores, we divide the possible variable voltage islands into the following cases. On one extreme, each core can be allocated an individual V_{dd} and thus the chip will have 16 different voltage

islands. On the other extreme, the chip can be assigned a single customized V_{dd} to optimize for power, thus having one voltage island. The other possibilities can be to divide the chip into 2, 4 and 8 islands. Figure 4.1 depicts some possible voltage island schemes for our 16-core CMP model. Note that for 2 and 8 voltage island configurations, we have two options. The cores can be selected horizontally or vertically to form the islands. For example, for the 2 islands case, the upper 8 and lower 8 cores can form the 2 islands (2-horizontal) or the left and right 8 cores can form the voltage island (2-vertical). In Section 4.3, we present results with both types of orientations.

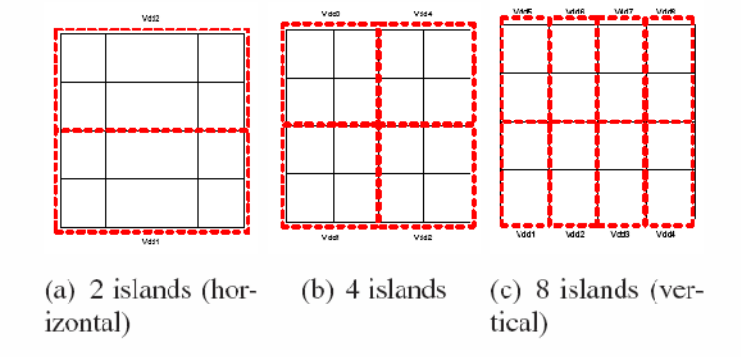

Figure 4.1. Various voltage island schemes.

One way of implementing the multiple-voltage-island scheme is to have a configurable DC-DC voltage converter in each voltage island. Once the chip has been tested the voltage levels for each island will be set once; in that way a dynamic adjustment can be avoided. Besides, several software tools allow user-level voltage

control to change the supply particularly in mobile processors [1]. This concept can similarly be extended to multicore systems. The extra overhead is going to be in the form of a supply voltage table keeping the voltage specifications for each island or core. The kernel is going to use this table during the boot operation. Alternatively, hardware mechanisms like [19-21] can be easily adapted for this purpose.

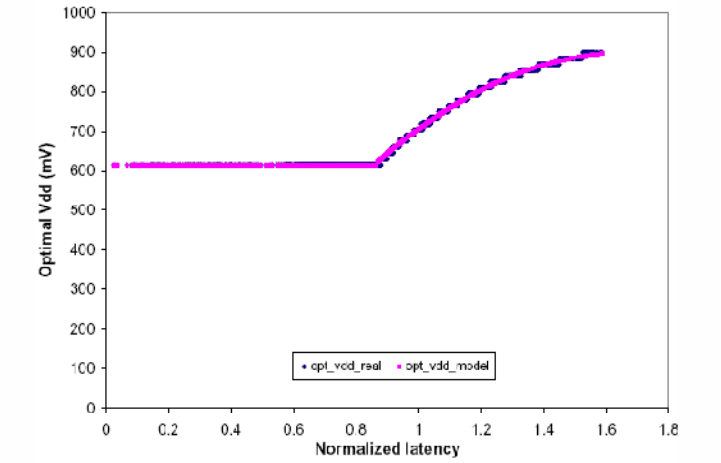

Figure 4.2. Optimum voltage for different voltage islands given as a function (h) of latency.

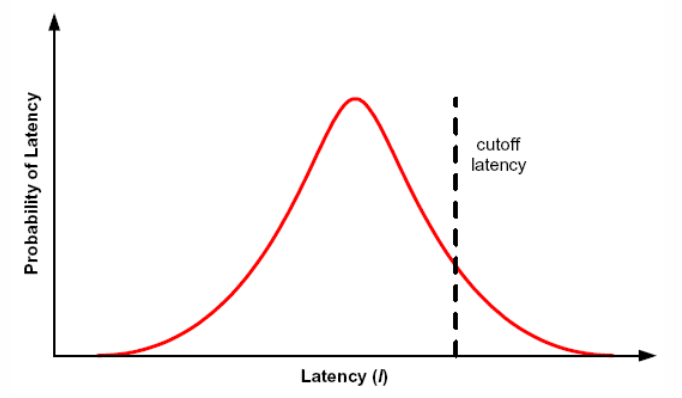

Figure 4.3. An example latency distribution curve (g(l)).

4.2 Modeling Power Optimization

In this section, we develop a model that can predict the amount of power savings for a given manufacturing technology. In the core of the model lies the observation that latency and voltage levels are correlated. For example, if a circuit operates at 8ns and our frequency requires 10ns operation, we can reduce the supply voltage until the latency is 10ns. Thus for a particular initial latency value (l), a corresponding minimum stable voltage (henceforth called optimal supply voltage V_{opt}) level exists that guarantees correct operation and results in the minimal power consumption. Since this metric depends on the latency, we first have to extract the relation between the latency and optimal supply voltage: V_{opt} = **h**(l). Note that this function is circuit-specific. For our target architecture, we have first plotted the latency (l) versus the corresponding V_{opt} values as shown in Figure 4.2. Then using curve fitting techniques the function **h** is found to be:

$$\boldsymbol{h}(l) = \begin{cases} 615 & if\ \ l < 2.18ns \\ 66.8l^2 - 566.8l - 1202.2 & if\ 2.18ns \le l < \ l_{cutoff} \\ 900 & otherwise \end{cases} \quad (7)$$

There are two important aspects of function **h**. First, our analysis of our circuit revealed that it does not work below 615mV (note that the nominal voltage level is 900mV). In addition, function **h** depends on the cutoff point set by the designer. This cutoff point corresponds to the frequency that the processor will run at and will be set

by the designer. Thus using function **h** we can compute the value of V_{opt}. Since dynamic power is proportional to the square of the supply voltage, from Equation 5 we get dynamic power savings for a chip with latency *l* as:

$$\Delta P_{dynamic} = 1 - \left[\frac{\boldsymbol{h}(l)}{Vdd_{nominal}}\right]^2 \qquad (8)$$

We also need the latency distribution for the batch of chips manufactured to be able to understand the advantages of a voltage island scheme. Assuming that this distribution is Gaussian (Figure 4.3), we can formulate the probability of a chip having a certain latency by **g** such that:

$$\boldsymbol{g}(l) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(1-\mu)^2}{2\sigma^2}} \qquad (9)$$

where μ and σ are the mean and standard deviation of the latency (*l*) distribution. Note that these values can be estimated for a given manufacturing technology. Thus the average dynamic power dissipation (**P**) for a batch of chips with latency distribution ***g(l)*** can be given as:

$$\boldsymbol{P} \propto \int_{l=0}^{l=cutoff} \boldsymbol{g}(l) \,.\, [\boldsymbol{h}(l)]^2 \,.\, dl \qquad (10)$$

Hence, given μ and σ of a distribution, Equation 10 can be used as an analytical model to compute dynamic power consumption with V_{opt}. In Section 4.3, we show that this model is highly accurate to estimate the optimized power consumption levels for our studied manufacturing technologies. Note that for a different technology, this model can be used by only providing the μ, σ, and cutoff values, which are easily available.

4.3 Experimental Results

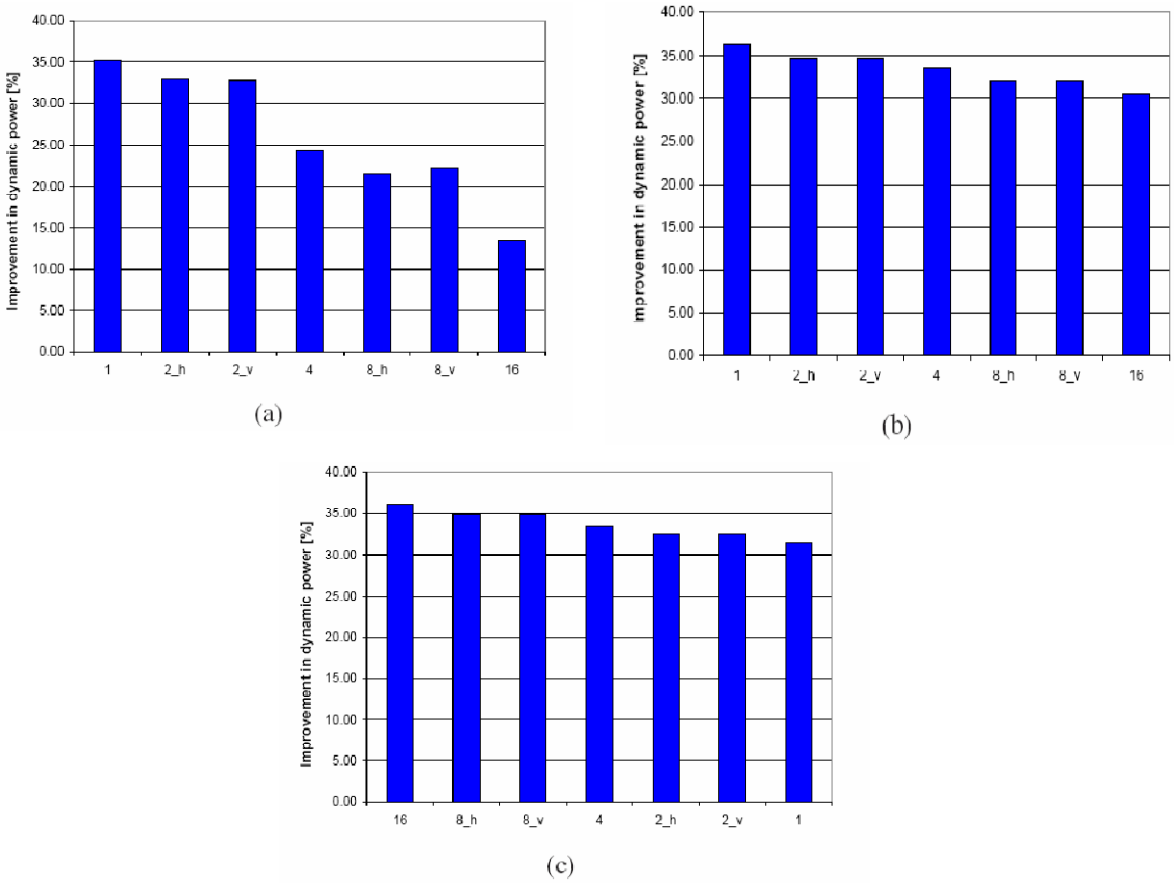

Figure 4.4. Power savings for 1, 2, 4, 8 and 16 voltage islands for (a) φ = 0.3, (b) φ = 0.5 and (c) φ = 0.7.

In this section we present the power optimization results for different voltage island schemes and also analyze how accurately can the power consumption with voltage islands be predicted using our model. We conducted SPICE simulations on the circuit model described in Chapter 1, Section 3.3. Since the randomness of a parameter changes with φ, we evaluate 7 different voltage islands schemes on 1000 multicore

chips each with φ values of 0.3, 0.5, and 0.7. The voltage islands can have a granularity of 1, 2, 4, 8, and 16 cores.

Figure 4.4 illustrates the power savings for different voltage island schemes with different amounts of randomness in variation. It shows the percentage of the power saving compared to the processor that uses nominal voltage (900mV) in all its cores. For a highly random case (φ = 0.3), the dynamic power improvement can lie between 13.5% and 35.1%. For the highest correlated variations (φ = 0.7), on the other hand, the improvements range between 31.5% and 36.2%. For the φ = 0.5 model, we see that the dynamic power reduction is between 30.5% and 36.2%. We can reach two important conclusions from these results. First, customized supply voltage levels can be an attractive mean to reduce the power consumption in CMPs under process variations. Particularly, we see that the dynamic power consumption of the chip can be reduced by as much as 36.2% on average, which is achieved when each core is individually controlled (16 voltage islands). Second, depending on the manufacturing technology, even a single customized voltage for the whole chip can reduce the power consumption significantly. Particularly, for the φ = 0.5 and φ = 0.7 models, we see that a single customized voltage level can reduce the power consumption by 30.5% and 31.5%, respectively. Only when the spatial correlation is diminishing (φ = 0.3), we need individual control of the cores: for the φ = 0.3 model, a scheme that uses 16 voltage

islands can save 35.1% of the dynamic power while the single voltage island scheme reduces the power consumption by only 13.5%.

Another interesting trend we observe in the results is that voltage islands having the same number of cores have almost same energy savings. For example, 2-vertical and 2- horizontal voltage islands have similar power savings.

Accuracy of the model: We compare the results obtained from the analytical model (Equation 10) with the empirical data from our experiments. The average error in **P** for φ values 0.3, 0.5, and 0.7 are found to be 0.01%, 0.30%, and 0.44%, respectively. Thus our model gives highly accurate estimations of the dynamic power consumption.

4.4 Summary

In this work we analyzed the effects of parameter variations on CMPs with an emphasis on the power dissipation. We presented a variation modeling technique which involves five different variation parameters affected by both systematic and random variations. We have first described an accurate model that can be used to estimate the advantages of forming voltage islands. Our simulations indicate that a custom supply voltage is more effective than a predetermined nominal V_{dd} for the entire chip. Particularly, application of multiple voltage islands with a latency constraint cuts the power dissipation of CMPs by as much as 36.2%. We also show that for most

manufacturing technologies, even a single customized supply voltage for the whole chip can reduce the power consumption substantially.

Chapter 5

Power due to On-chip Interconnect:

Advances in process technology enable exponentially more cores on a single die with each new process generation, leading to a commensurate increase in cache sizes to supply all these cores with data. To combat the increasing on-chip wire delays as the core counts and cache sizes grow, future multicore architectures become distributed: the last-level on-chip cache (LLC) is divided into multiple cache slices, which are distributed across the die area along with the cores [26, 83]. To facilitate data transfers and communication among the cores, such processors employ elaborate on-chip interconnection networks. Theon-chip interconnection networks are typically optimized to deliver high bandwidth and low latency.

However, such on-chip interconnects come at a steep cost. Recent studies show that on-chip networks consume between 20% to 36% of the power of a multicore chip[36, 47]and significantly raise the chip temperature [64] leading to hot spots, thermal emergencies, and degraded performance. As core counts continue to scale, the impact of the on-chip interconnect is expected to grow even higher in the future.

The flurry of recent research to minimize the power consumption of on-chip interconnects is indicative of the importance of the problem. Circuit-level techniques to

improve the power efficiency of the link circuitry and the router microarchitecture [76], dynamic voltage scaling [62] and power management [36, 63], and thermal-aware routing [64] promise to offer a respite, at the cost of extensive re-engineering of the interconnect circuitry and routing protocols. Yet, prior works miss one crucial observation: a large fraction of the on-chip interconnect traffic stems from packets sent to enforce data coherence, rather than from packets absolutely required to facilitate data sharing.

The coherence requirement is a consequence of performance optimizations for on-chip data. To allow for fast data accesses, the distributed cache slices are typically treated as private caches to the nearby cores [8, 12, 83],forming tiles with a core and a cache slice in each tile [5, 26]. Private caches allow the replication of shared data, which, in turn, require a mechanism to keep the data coherent in the presence of updates. To allow scaling to high core counts and facilitate coherent data sharing, modern multicores employ a directory structure, which is typically address-interleaved among the tiles [8, 26, 83].

However, address interleaving is oblivious to the data access and sharing patterns; it is often the case that a cache block maps to a directory in a tile physically located far away from the accessing cores. To share a cache block, the sharing cores need to traverse the on-chip interconnect multiple times to communicate with the directory, instead of communicating directly between them. These unnecessary network traversals increase traffic, consume power, and raise the operational temperature with detrimental

consequences. Ideally, from a power-optimization standpoint, the directory entry for a cache block would be co-located with the most active sharing core of the block, rather than a seemingly random one.

In this chapter, we observe that a large fraction of the on-chip interconnect traffic stems from the data-access-oblivious placement of directory entries. Based on this observation, we propose a distributed cache architecture that cooperates with the operating system to place directory entries close to the most active requestors of the corresponding cache blocks, eliminating unnecessary network traversals and conserving energy and power. The mechanisms we propose exploit already existing hardware and operating system structures and events, have negligible overhead, they are easy and practical to implement, and can even slightly improve performance.In summary, the contributions of this paper are:

- We observe that a large fraction of the on-chip interconnect traffic stems from the data-access-oblivious placement of directory entries.
- We propose Power-Aware Directory placement (PAD), a mechanism to co-locate directory entries with the most active requestors of the corresponding cache blocks, eliminating unnecessary network traversals and conserving energy and power.
- Through trace-driven and cycle-accurate simulation of large scale multicore processors running a range of scientific [78] and Map-Reduce [55] workloads, we show that PAD reduces the interconnect energy and power by

up to 37% (22% on average for the scientific workloads and 8% on average for Map-Reduce) with a 1.4%performance improvement on average.

5.1 Background on NUCA Caches

We assume a tiled multicore, where each tile consists of a processing core, a private split I/D first-level cache (L1), a slice of the second-level cache (L2), and a slice of the distributed directory. Figure 5.1 depicts a typical tiled multicore architecture. We assume a private NUCA organization of the distributed L2 cache [26], where each L2 slice is treated as a private L2 cache to the local core within the tile.

On-chip distributed caches in tiled multicores typically use a directory-based mechanism to maintain coherence [8, 26, 83]. To scale to high core counts, the directory is also distributed among the tiles in an address-interleaved fashion (i.e., the address of a block modulo the number of tiles determines the directory location for this block). In the ideal case, the directory has the capacity to hold coherence information for all the cache blocks across all the tiles in all cases (i.e., a full-map directory). Techniques like sparse directories reduce the capacity requirements of full-map ones. However, the investigation of directory capacity-management mechanisms is beyond the scope of this paper. Without loss of generality, and similarly to most relevant works, we assume a full-map directory for the baseline and PAD architectures. In general, PAD uses the same directory mechanisms as the baseline architecture.

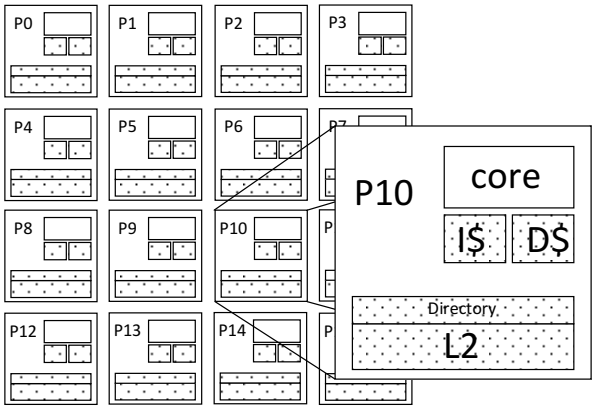

Figure 5.1. Baseline tiled architecture of a 16-core CMP. Each tile has core, split I/D L1, L2 and directory slice.

Address interleaving does not require a lookup to extract the directory location; all nodes can independently calculate it using only the address of the requested block. However, address-interleaved placement statically distributes the directories without regards to the location of the accessing cores leading to unnecessary on-chip interconnect traversals. Figure 5.2-a shows an example of the drawbacks of static address-interleaved directory placement. Tile 7 requests a data block, currently owned by Tile 1, with its directory entry located at Tile 5 as determined by address interleaving. To access the block, Tile 7 first has to access the directory at Tile 5, which forwards the request to the owner Tile 1, which then sends the data to Tile 7. As the directory placement is oblivious to the location of the sharing cores, most on-chip data transfers will require similar 3-hop messages. Ideally, the directory would be co-located with the sharer at Tile 1 (Figure 5.2-b), which would eliminate two unnecessary

network messages and result in reduced power consumption and faster data access. Such placement is the goal of PAD.

A similar message sequence is generated upon requests than result in off-chip misses. In such a case, even if the accessed data are private, the requesting core first contacts the corresponding directory, which then sends a message to the memory controller. When the data is available, the memory controller sends a message to the directory node and the data to the requestor. If the directory is co-located with the requesting core using PAD, one or two messages are eliminated (in an aggressive design, the data and directory replies may be combined into a single message, as they both go to the same destination tile).

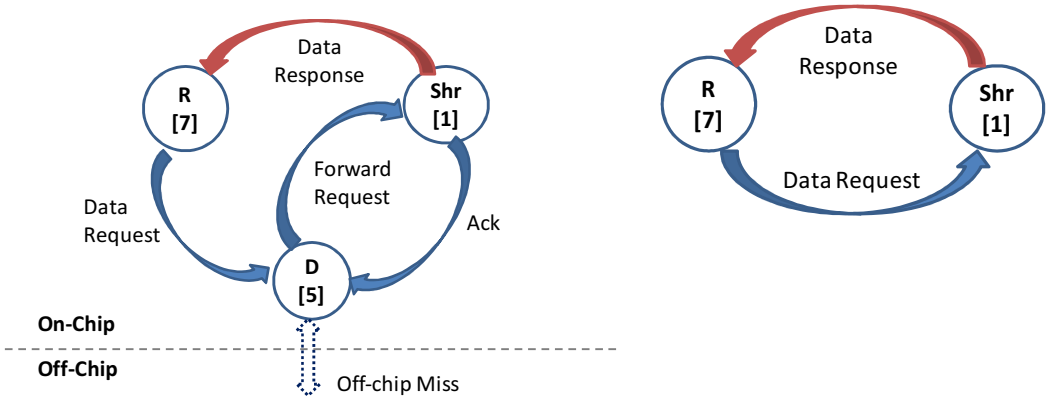

Figure 5.2. (a) Sequence of on-chip network messages following a request by tile 7 for a block owned by tile 1, with its directory at tile 5. (b) The same when the owner tile 1 also holds the directory entry.

The fewer the sharers of a block, the higher the impact of intelligent directory placement. A block that is private to a core and has its directory entry within the same tile can be accessed without any intermediate directory nodes participating in resolving

local cache misses. A block with a couple of sharers and its directory co-located with one of them can be shared through direct communication among the sharers, also without the need to access an intermediate node. At the far end of the sharing spectrum, a universally-shared block that all cores access with similar frequency cannot benefit from the intelligent placement of its directory entry, because any location is as good as any other. Thus, applications with a large fraction of data with one sharer (private) or with few sharers (2-4) will benefit the most from PAD. In Section 5.2 we show that there is a considerable fraction of accesses to blocks with one or few sharers in a variety of parallel applications, rendering traditional address-interleaved directory placement inefficient.

Note that in an *N*-tile multicore with address-interleaved distributed directory, the probability of a particular tile holding the directory entry for a block is *1/N*. This is the probability with which a requesting core can access a directory within its own tile. As the number of tiles increases, the probability of hitting a local directory diminishes. Thus, traditional address-interleaved directory placement becomes increasingly inefficient in future technologies, as the core counts grow. This has a twofold impact, both on the power and the performance of the chip. The extra hops to a remote directory increase the on-chip network power usage, and the extra hop latency increases the access penalty for a block that missed locally. Our proposal, **Power-Aware Directory placement (PAD)**, promises to mitigate both effects.

Table 5.1. Description of workloads.

Benchmark	Application	Description
SPLASH-2	*unstructured*	Computational fluid dynamics application
	ocean	Simulates large-scale ocean movements based on eddy and boundary currents
	dsmc	Simulates the movement and collision of gas particles
	appbt	Solves multiple independent systems of equations
	watersp	Simulates the interactions of a system of water molecules
	moldyn	Molecular dynamics simulation
	barnes	Barnes-Hut hierarchical N-body simulation
	tomcatv	A parallelized mesh generation program
	fmm	Simulates particle interactions using the Adaptive Fast Multipole Method
Phoenix	*lreg*	Linear regression to find best fit line for a set of points
	hist	Histogram plot over a bitmap image file
	kmeans	K-Means clustering over random cluster points and cluster centers
	pca	Principal component analysis over a 2D-matrix
	smatch	String matching in a large text file
	wcount	Word count in a large text file

Table 5.2. System parameters for the simulated framework.

CMP Size	16 cores
Processing Cores	UltraSPARC III ISA; 2GHz, in-order cores, 8-stage pipeline, 4-way superscalar
L1 Caches	split I/D, 16KB 2-way set-associative, 2-cycle load-to-use, 3 ports 64-byte blocks, 32 MSHRs, 16-entry victim cache
L2 NUCA Cache	private 512KB per core, 16-way set-associative, 14-cycle hit
Main Memory	4 GB memory, 8KB pages, 45 ns access latency
Memory Controllers	one controller per 4 cores, round-robin page interleaving
Interconnect	2D folded torus [8,28], 32-byte links, 1-cycle link latency, 2-cycle router
Cache Coherence Protocol	Four-state MOSI modeled after Piranha [7]

5.2 Overview of Power-Aware Directory Placement (PAD)

At a high level, PAD utilizes the virtual address translation mechanism to assign directories to tiles at page granularity. The first time a page is accessed, the tile of the accessing core becomes the owner of the directories for that page (directory information is still maintained at cache block granularity; only the placement of the entries to tiles is done at the page level). This information is stored in the page table and propagated to the TLB. If another core accesses the same page, the directory location is provided along with the physical address of the page. Therefore, a second core accessing the same page can directly contact the directory entries for blocks in that page. Thus, PAD decouples the address of a block from the physical location of its directory, allowing the directory to be placed anywhere on chip without complicating lookup. In the remainder of this section, we motivate the deployment of PAD through an analysis of the sharing patterns of SPLASH-2 and Phoenix Map-Reduce applications.

5.2.1 Experimental Methodology

We evaluate PAD on two different benchmark suites, SPLASH-2 [78] and Phoenix [55], which are described in more detail in Table 5.1. SPLASH-2 consists of a mixture of compute-intensive applications and computational kernels. Phoenix consists of data-intensive applications that use Map-Reduce.

We analyze the data sharing patterns across our application suite by collecting execution traces of each workload using SimFlex [77], a full-system cycle-accurate

simulator of multicores with distributed non-uniform caches. The traces cover the entire execution of the Map phase for Phoenix applications (which constitutes the majority of execution time) and threecomplete iterations for SPLASH-2 applications. The workloads execute on a 16-core tiled CMP supported by a 2D folded torus interconnect similar to [26]. The architectural parameters for our baseline configuration are depicted in Table 5.2.

5.2.2 Analysis of Sharing Patterns

A core first searches for data in its local L2 cache. If it misses, then a directory access for the corresponding block follows. For each workload, Figure 5.3 shows the percentage of local L2 misses (i.e., directory accesses) on blocks that are accessed by only one core during the execution of the program (1 shr, i.e., private blocks), accessed by few cores (2-4 shr), accessed by a large number of cores (5-15 shr), and blocks that are universally shared (16 shr).

As described in Section 5.3, placing the directory of private blocks in the same tile with the core accessing these blocks will eliminate two control messages for every local L2 miss.In contrast, conventional address-interleaved directory placement will co-locate the directory and the requestor only a small fraction of the time. For the cases where the accesses are to blocks with afew sharers (2-4), co-locating the directory with one of the requesting cores will significantly increase the probability that the directory and the requester are in the same tile, which will also lead to the elimination of two messages.

As the number of sharers increases, this probability decreases; in the case of universal sharing (16 shr), conventional address-interleaved directory placement will always co-locate the directory with one of the sharers, hence our proposed scheme will provide no additional benefit.

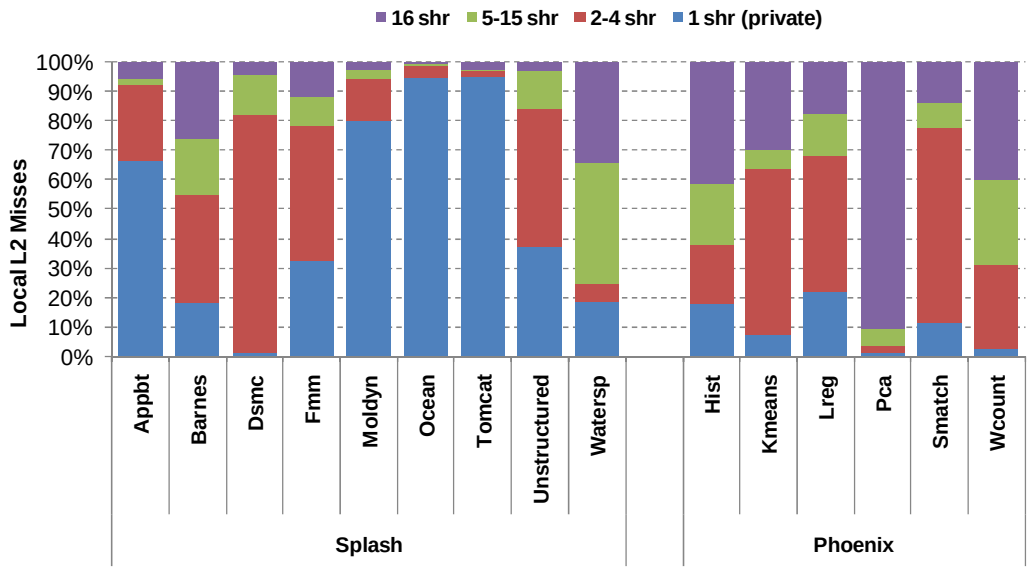

Figure 5.3. Access sharing pattern at the block level based on number of sharers per block.

Figure 5.3 shows that Phoenix and SPLASH-2 applications exhibit a significant fraction of directory accesses for blocks that are private or have a few sharers. Averaged across all 15 workloads, 35% of the directory accesses are for private data and 33% of the accesses are for data shared among 2-4 cores. However, there are some exceptions to this behavior: *pca* exhibits a large fraction of universally shared data. Nevertheless, our analysis suggests that in a large majority of applications,the most accessed

directoriesare either for private data or for data with a few sharers, motivating the use of PAD.

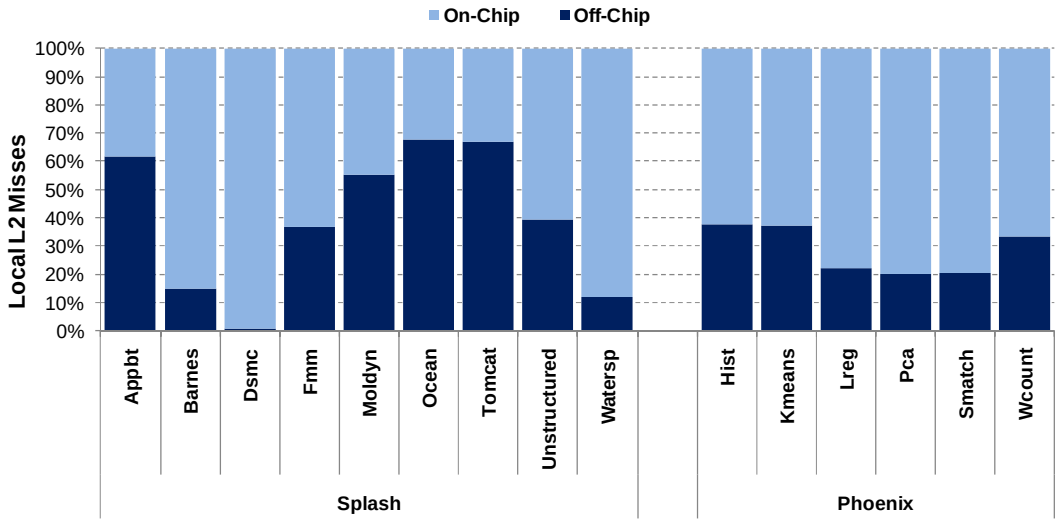

Figure 5.4. Accesses breakdown by off-chip and on-chip accesses.

Figure 5.4 illustrates the breakdown of local L2 misses based on whether the access is made to an off-chip block or to a block that resides in some remote tile on chip. Both types of accesses have to access the directory first. In the case of an off-chip miss, the directory sends a message to the memory controllerto fetch the block. In an aggressive protocol, the memory controller sends the data directly to the requestor, and a reply acknowledgement to the directory. In a conservative protocol, the memory controller sends the data to the directory, which then forwards it to the requestor. In either case, the directory is informed about the cache fill. If the requestor and the directory entry are

in different tiles, four messages are generated; PAD could eliminate two of them, by placing the directory entry together with the requestor.

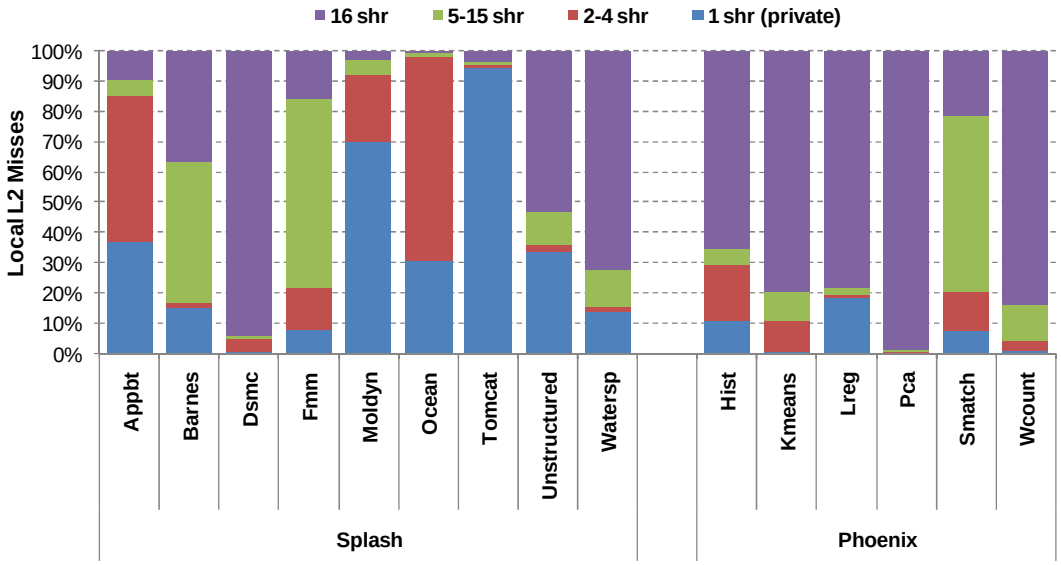

Figure 5.5. Access sharing pattern at the page level based on number of sharers per page.

In the case of a local miss to a block that resides on chip, the directory sends a request to the owner of the block, which then sends the data to the requestor and an acknowledgement to the directory, so that it can update the coherence state of the block and finalize the transaction. Hence, for cache-to-cache transfers, a total of four messages are generated (one of which will be avoided if the directory resides with the requesting core, and two will be avoided if the directory resides with the owner of the block).

PAD determines the placement of a directory at the page granularity (i.e., all the directory entries for the blocks within in a page are located in the same tile). Hence, the

sharing pattern at the page granularity determines the overall performance of our scheme. Similar to Figure 5.3, Figure 5.5 shows the percentage of local L2 misses (i.e., directory accesses) on blocks that are within pages accessed by some number of cores during the execution of the workload. Averaged across all 15 applications, 23% of the accesses are on pages that are private and 13% of the accesses are on pages with 2-4 sharers. Thus, although working at the page granularity introduces false sharing, the change is not drastic as compared to block granularity.

5.3 Power-Aware Directory Placement: In Detail

Power-Aware Directory placement (PAD) reduces the unnecessary on-chip interconnect traffic by placing directory entries on tiles with cores that share the corresponding data. To achieve this, for every page, PAD designates an owner of the directory entries for the blocks in that page, and stores the owner ID in the page table. By utilizing the already existing virtual-to-physical address translation mechanism, PAD propagates the directory owner location to all cores touching the page. There are three important aspects of this scheme: the classification of pages by the OS, the directory placement mechanism, and the distribution of directory owners among cores. We investigate these aspects in the following sections.

5.3.1 Operating System Support

To categorize pages and communicate their directory location to the cores, PAD piggybacks on the virtual-to-physical address translation mechanism. In modern systems, almost all L2 caches are physically accessed. Thus, for all data and instruction accesses, a core translates the virtual address to a physical one through the TLB before accessing L2. Upon a TLB miss (e.g., the first time a core accesses a page, or if the TLB entry has been evicted) the system locates the corresponding OS page table entry and loads the address translation into the TLB.

We implement PAD by slightly modifying this process. When a page is accessed for the first time ever by any of the cores, the first accessor becomes the owner of the corresponding directory entries (this is called *first-touch directory placement* and we discuss its effects in the next section). This information is stored in the page table. Upon a TLB fill, the OS (or the hardware page walk mechanism) provide this owner information to the core along with the translation, and store it in its TLB. Thus, any subsequent accessor of the page is also notified of the directory location for the blocks in the page. This mechanism guarantees that the directory is co-located with one of the sharers of the page. If the page is privately accessed, the tile of the accessing core will hold the directory entries for all the blocks in the page.

5.3.2 Discussion

Directory placement can be done at different granularities. For example, instead of designating one tile as the owner for the directory entries of all the blocks in the page, we could designate different owners for the directory entry of each block individually (or any granularity in between). Such a fine-grain placement would require considerable changes in the overall system operation. First, each TLB entry would have to store multiple directory owners (one per placement-grain). In turn, this would require a separate TLB trap for each sub-section of the page that is accessed to extract the directory location for it. Our results indicate that the system behaves well enough at the page granularity that employing finer-grain techniques is unjustified. Nevertheless, in the next section, we provide hypothetical energy savings of such an approach; our results indicate that, for most applications, finer granularity provides negligible benefits.

Directory placement could be achieved by simply guiding the selection of physical addresses for each virtual page (i.e., some bits of the physical address will also designate the directory owner). However, such a technique would couple the memory allocation with the directory placement. As a result, forcing the use of specific address ranges could lead to address space fragmentation with detrimental consequences in performance, and may complicate other optimizations (e.g., page coloring for L1) that pose conflicting address translation requests. PAD avoids these problems by fully decoupling page allocation from directory placement.

While pathological cases are possible, we didn't see any in our workloads, and we don't expect to see any in commercial workloads either: their data are typically universally shared with finely interleaved accesses [26], so the pages should distribute evenly. It is important to note here that it is simple to turn off PAD in pathological cases: one bit per page could indicate whether these entries are managed by PAD or a traditional method. Finally, in the case of heavily-migrating threads, the corresponding directory entries could either stay in the original tile and be accessed remotely by the migrating thread (similar to the baseline), or move along with it, or we could simply turn off PAD as described above. While dynamic directory migration is possible under our scheme, the complexities it entails may overshadow its benefits. Hence, we leave the investigation of on-chip directory migration schemes to future work.

5.4 Experimental Results

5.4.1 Methodology

We evaluate PAD using the SimFlex multiprocessor sampling methodology [77]. Our samples are drawn over an entire parallel execution (Map phase) of the Phoenix workloads, and three iterations of the scientific applications (SPLASH-2). We launch measurements from checkpoints with warmed caches, branch predictors, TLBs, on-chip directories, and OS page tables, then warm queue and interconnect state for 100,000 cycles prior to measuring performance for 200,000 cycles. We use the aggregate number of user instructions committed per cycle as our performance metric, which is

proportional to overall system throughput [77]. The architectural parameters are described in Section 5.2.1.

5.4.2 First Touch Directory Placement

To evaluate the effectiveness of the first-touch directory placement policy, we compute the number of page accesses by the core that was the first ever to access the page (FirstAcc), and compare it against the accesses issued by the most frequent accessor for the same page (MaxAcc). From a power optimization standpoint and in the absence of directory migration, MaxAcc would be the ideal directory location for that page. As Figure 5.6 shows, first-access directory placement is a good approximation of the ideal scheme: the number of accesses issued by the first accessor are very close to the maximum accessor's.

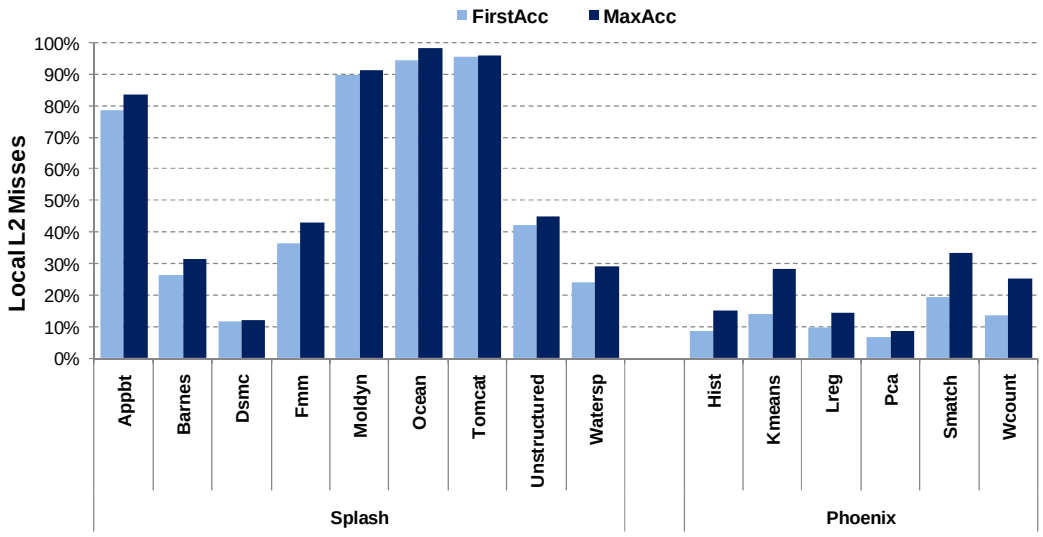

Figure 5.6. Effectiveness of the first-touch directory placement policy.

5.4.3 Distribution of Directory Entries Across Tiles

The first-touch directory placement policy may result in an imbalanced distribution of directory entries to tiles, in contrast to the almost even distribution under traditional address interleaving. If some tiles are assigned vastly more directory entries than others, they would require a disproportionately large area for the directory, or could result in traffic hotspots that degrade performance.

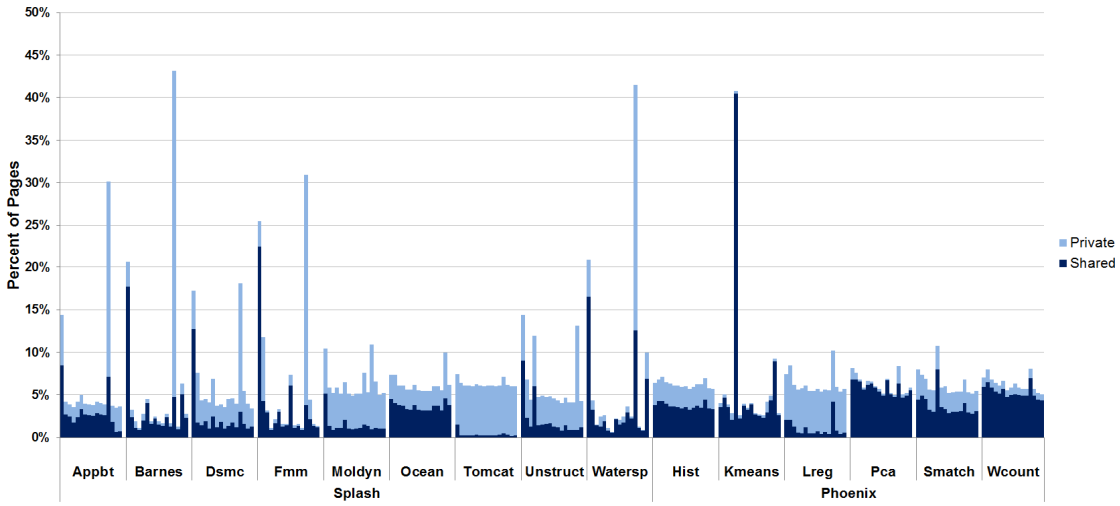

Figure 5.7. Distribution of directory entries for pages across tiles under the first-touch placement policy.

Figure 5.7 presents the distribution of directory entries under the first-touch placement policy across tiles for private and shared pages. The red line indicates the level of a hypothetical uniform distribution. The uneven distribution of directory entries is an artifact of the first-touch directory placement policy and could be minimized. First, only the shared pages matter; private data are accessed by only one core so they are always coherent, obviating the need for a directory. Thus, PAD can defer the directory

entry allocation until a page is accessed by a second core, ensuring that directories exist only for shared pages and conserving directory area. With the exception of Kmeans and Fmm, shared entries for the remaining applications are mostly evenly distributed (in the remaining applications, a tile gets at most 18% of the total entries). Second, PAD can minimize the uneven distribution by utilizing a "second-touch" placement policy (i.e., the second sharer being assigned the entries) when the first-touching tile is overloaded. Third, the uneven distribution of pages is not a direct indicator of increased traffic hotspots, as some pages are colder than others, and the baseline may also exhibit imbalanced traffic.

To investigate hotspots, we analyzed the accesses to each directory tile under PAD and baseline[1]. Our results indicate that PAD reduces the number of network messages for most applications. In some cases, it even cuts the number of control messages by almost half (appbt, moldyn, ocean, and tomcatv; see Figure 5.9). With the exception of Kmeans, Fmm and Dsmc, the remaining applications exhibit a slightly higher imbalance than baseline, with a tile receiving at most 16% more directory accesses from remote cores (8% more on average). These imbalances are relatively small and do not impact the overall performance (Figure 5.10). Applying PAD on Kmeans, Fmm, and Dsmc exacerbates already existing traffic imbalances. However, we find that even these hotspots have a negligible performance impact, due to the already small fraction of execution time these applications spend on the distributed L2 cache.

[1] We omit the graph due to space constraints, and instead present our findings in the text.

5.4.4 Energy Savings

Figure 5.8 presents the fraction of network energy saved by PAD. For each application, the left bar indicates the energy savings attained by PAD at cache-block granularity, while the right bar presents PAD for 8KB pages. PAD reduces the network energy by 20.4% and 16.1% on average for block- and page-granularity, respectively, mainly by reducing the network messages. As expected, the block-granularity shows higher energy savings compared to the page-granularity. However, as we describe in Section 5.3.2, such an implementation would complicate the design considerably (and will incur performance costs).

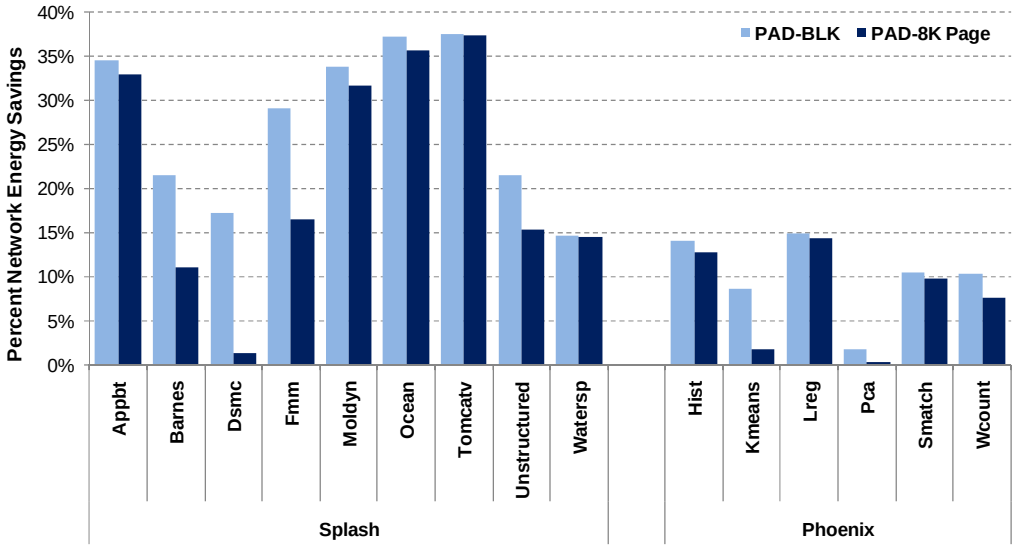

Figure 5.8. On-chip network energy savings obtained by block-grain and page-grain PAD.

In general, we note that the scientific applications attain higher energy savings compared to Phoenix. Phoenix applications exhibit a higher fraction of shared data

accesses (Section 5.2). As a result, our schemes are more useful for SPLASH-2 applications. In fact, we observe a strong correlation between the sharing distribution (Figure 5.5) and the energy reduction (Figure 5.8) for each of the studied applications.

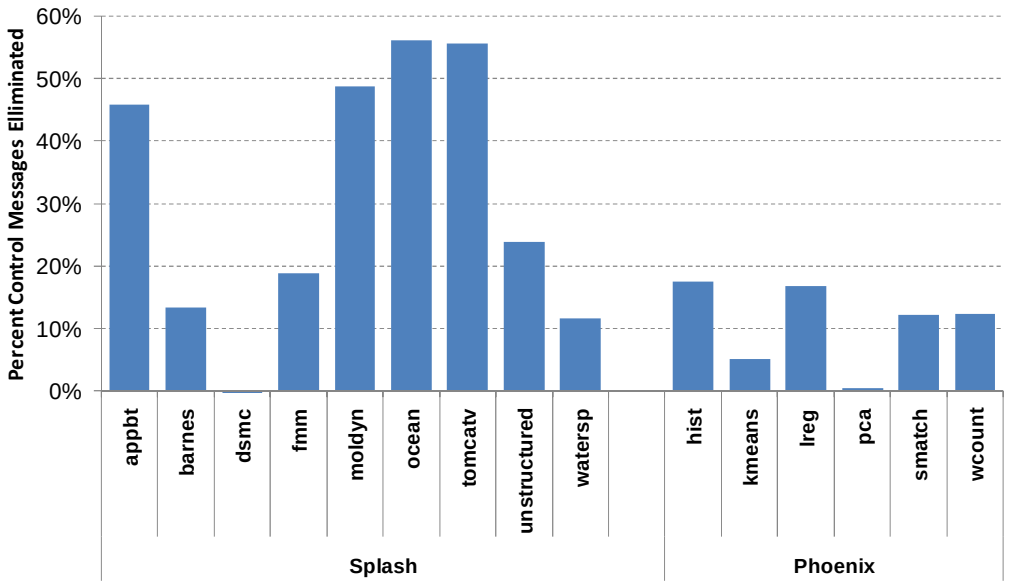

Figure 5.9. Reduction of network control messages attained by PAD with respect to Baseline.

5.4.5 Performance Impact

Figure 5.10 shows the overall speedup of PAD as compared to a baseline private NUCA architecture. Interestingly enough, we observe that PAD slightly increases performance in 7 out of 15 applications, and decreases performance in 2. PAD improves performance by up to 7% (*Ocean*), and by 1.3% on average, while the maximum performance slowdown is 1.3% (*Pca*). The performance is improved due to two reasons. First, PAD reduces the number of network packets which may eliminate

congestion and hence reduce the overall latency of network operations. Second, data transfers (on-chip and off-chip) are faster because the access to a remote directory is eliminated in many cases. Because the working set is large, PAD's savings are realized mostly by off-chip memory accesses. As the off-chip memory access latency is already large, saving a small number of cycles does not improve the performance considerably.

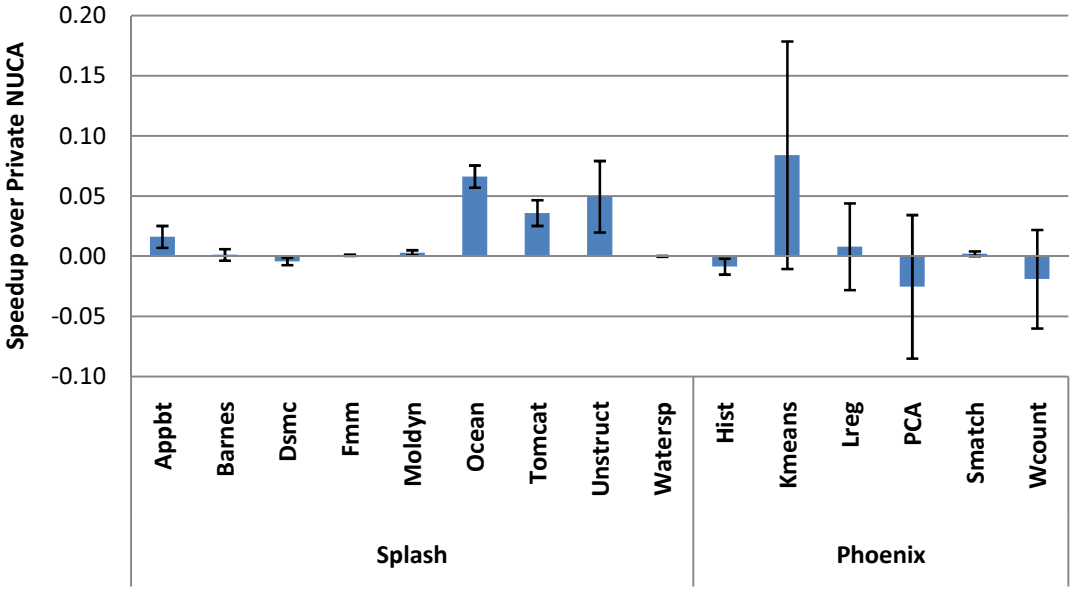

Figure 5.10. Speedup of PAD over the baseline private NUCA architecture.

The reason for the slowdown exhibited by a couple of the applications is attributed to the fact that PAD assigns directories for a whole page to one node. If it fails to reduce the number of network packets, this assignment can cause contention and hotspots. Especially for universally-shared pages, it is likely that blocks are accessed by different cores in nearly consecutive cycles, causing contention in the directory tile, and increasing the directory's response time. On average, we observe that the positive and

negative forces cancel each other out, and PAD has only a negligible overall performance impact.

5.5 Summary

As processor manufacturers strive to deliver higher performance within the power and cooling constraints of modern chips, they struggle to reduce the power and energy consumption of the most insatiable hardware components. Recent research shows that on-chip interconnection networks consume 20% to 36% of a chip's power, and their importance is expected to rise with future process technologies. In this paper, we observe that a large fraction of the on-chip traffic stems from placing directory entries on chip without regards to the data access and sharing patterns. Based on this observation, we propose Power-Aware Directory placement (PAD), a distributed cache architecture that cooperates with the operating system to place directory entries close to the most active requestors of the corresponding cache blocks, eliminating unnecessary network traversals and conserving energy and power. The mechanisms we propose exploit already existing hardware and operating system structures and events, have negligible overhead, and are easy and practical to implement. Through trace-driven and cycle-accurate simulation on a range of scientific and Map-Reduce applications, we show that PAD reduces the power and energy expended by the on-chip network by up to 37% (16.4% on average) while attaining a small improvement in performance (1.3% on average). Thus, we believe PAD is an appealing technique that shows great promise

in reducing the power and energy of the on-chip interconnect, with negligible overheads.

Chapter 6

Conclusions and Future Work

Future technological trends will intensify the problem of power and reliability in modern multi-core processors. On one hand device parameter variability gives rise to reliability concerns. On the other hand, the exponential growth of power in chips necessitates the needs for architectural approaches that need to mitigate the power problem. The goal of this thesis is provide microarchitectural level solutions to these problems. Particularly this thesis makes the following contributions:

- Analyzes the impact of process variations on power and performance of processors
- Proposes efficient cache resizing and redundancy schemes to optimize the frequency binning under process variations which leads to high batch performance and larger profitability
- Proposes power-aware voltage/frequency island scheme to reduce overall dynamic power dissipation in multi-core chips
- Propose power-aware directory co-location policy in private NUCA caches that effectively reduce network traversals and hence save active power.

While several researchers are studying novel cache and directory management schemes for future generation architectures to optimize for power and reliability, this

thesis only proposes few architectural solutions to these problems. Several areas of future work related to this work can be categorized as but are not limited to the following:

- Evaluation of directory co-location scheme for multi-programmed workloads and other parallel software
- Modeling power dissipation in a multicore platform and evaluating the power-savings from directory management schemes as described above
- Other schemes such as physical thread assignment in multicore processors that can increase spatial data locality and hence can optimize coherence misses.

In summary, this thesis opens up several interesting problem in the multicore cache/memory design space.

Bibliography

[1] *Notebook Hardware Control, personal edition.* Available: http://www.pbus-167.com

[2] A. Agarwal, B. C. Paul, H. Mahmoodi, A. Datta, and K. Roy, "A Process-Tolerant Cache Architecture for Improved Yield in Nanoscale Technologies," *IEEE Trans. Very Large Scale Integrated Systems,* vol. 13, pp. 27-38, 2005 2005.

[3] D. Albonesi, "Selective Cache Ways: On-demand Cache Resource Allocation," in *Intl. Symposium on Microarchitecture,* Haifa, Israel, 1999, pp. 248 - 259.

[4] B. S. Amrutur and M. A. Horowitz, "Speed and Power Scaling of SRAM's," *IEEE Trans. on Solid-State Circuits,* vol. 35, pp. 175-185, Feb. 2000 Feb. 2000.

[5] M. Azimi, N. Cherukuri, D. Jayasimha, A. Kumar, P. Kundu, S. Park, I. Schoinas, and A. Vaidya, "Integration challenges and trade-offs for tera-scale architectures," *Intel Technology Journal,* vol. 11, pp. 173-184, 2007.

[6] J. Balfour and W. Dally, "Design tradeoffs for tiled CMP on-chip networks," 2006, p. 198.

[7] L. A. Barroso, K. Gharachorloo, A. Nowatzyk, R. McNamara, R. Stets, S. Smith, S. Qadeer, B. Sano, and B. Verghese, "Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing," 2000, pp. 282-282.

[8] B. Beckmann, M. Marty, and D. Wood, "ASR: Adaptive selective replication for CMP caches," 2006, pp. 443-454.

[9] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, "Parameter Variations and Impact on Circuits and Microarchitectures," in *Proc. of the Design Automation Conference,* Anaheim, CA, 2003, pp. 338-342.

[10] W. Bryg and J. Alabado. The UltraSPARC T1 Processor - Reliability, Availability, and Serviceability.

[11] Y. Cao, T. Sato, D. Sylvester, M. Orshansky, and C. Hu, "New Paradigm of Predictive MOSFET and Interconnect Modeling for Early Circuit Design," in *Custom Integrated Circuits Conference,* Orlando, FL, 2000, pp. 201-204.

[12] J. Chang and G. S. Sohi, "Cooperative Caching for Chip Multiprocessors," *SIGARCH Comput. Archit. News,* vol. 34, pp. 264-276, 2006.

[13] M. Chaudhuri, "PageNUCA: Selected policies for page-grain locality management in large shared chip-multiprocessor caches," 2009, pp. 227-238.

[14] Z. Chishti, M. D. Powell, and T. N. Vijaykumar, "Optimizing Replication, Communication, and Capacity Allocation in CMPs," presented at the Proceedings of the 32nd annual international symposium on Computer Architecture, 2005.

[15] S. H. Choi, B. C. Paul, and K. Roy, "Novel Sizing Algorithm for Yield Improvement Under Process Variation in Nanometer Technology," in *Proc. of the Design Automation Conference*, ed San Diego, CA, 2004, pp. 454-459.

[16] P. Choudhary and D. Marculescu, "Hardware based frequency/voltage control of voltage frequency island systems," in *Proceedings of the 4th international conference on Hardware/software codesign and system synthesis (CODES+ISSS)*, New York, NY, 2006.

[17] O. Coudert, "Gate Sizing: A General Purpose Optimization Approach," in *European Design and Test Conference*, 1996, p. 214.

[18] A. Datta, S. Bhunia, J. H. Choi, S. Mukhopadhyay, and K. Roy, "Speed Binning Aware Design Methodology to Improve Profit Under Parameter Variations," in *Proc. of the Conf. on Asia South Pacific Design Automation*, Yokohama, Japan, 2006, pp. 712-717.

[19] S. Dhar, D. Maksimovi, and B. Kranzen, "Closed-loop adaptive voltage scaling controller for standard-cell asics," in *ISLPED*, New York, NY, 2002, pp. 103–107.

[20] M. Elgebaly and M. Sachdev, "Efficient adaptive voltage scaling system through on-chip critical path emulation," in *International Symposium on Low Power Electronics and Design (ISLPED)*, 2004.

[21] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation," in *Proc. of the Intl. Symposium on Microarchitecture*, San Diego, CA, 2003, p. 7.

[22] B. Falsafi and D. Wood, "Reactive NUMA: a design for unifying S-COMA and CC-NUMA," *ACM SIGARCH Computer Architecture News*, vol. 25, p. 240, 1997.

[23] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge, "Drowsy Caches: Simple Techniques for Reducing Leakage Power," in *International Conference on Computer Architecture (ISCA)*, Anchorage, Alaska, 2002, pp. 148 - 157.

[24] P. Friedberg, Y. Cao, J. Cain, R. Wang, J. Rabaey, and C. Spanos, "Modeling Within-Die Spatial Correlation Effects for Process-Design Co-Optimization," in *Proc. of the Intl. Symposium on Quality of Electronic Design*, San Jose, CA, 2005, pp. 516-521.

[25] Z. Guz, I. Keidar, A. Kolodny, and U. Weiser, "Utilizing shared data in chip multiprocessors with the Nahalal architecture," 2008, pp. 1-10.

[26] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Reactive NUCA: near-optimal block placement and replication in distributed caches " in *Proceedings of the 36th annual international symposium on Computer architecture* Austin, TX, 2009.

[27] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. Keckler, "A NUCA substrate for flexible CMP cache sharing," *IEEE transactions on parallel and distributed systems*, vol. 18, 2007.

[28] E. Humenay, D. Tarjan, and K. Skadron, "Impact of Parameter Variations on Multi-Core Chips," in *Workshop on Architectural Support for Gigascale Integration*, 2006.

[29] E. Humenay, D. Tarjan, and K. Skadron, "The impact of systematic process variations on symmetrical performance in chip multi-processors," in *Design, Automation and Test in Europe (DATE)*, 2007.

[30] Intel. (2006, *Intel Processor Pricing*. Available: http://www.intel.com/intel/finance/pricelist/processor_price_list.pdf?iid=InvRel+pricelist_pdf

[31] L. Jin, H. Lee, and S. Cho, "A flexible data to L2 cache mapping approach for future multicore processors," 2006, p. 101.

[32] N. P. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers," *ACM SIGARCH Computer Architecture News* vol. 18, pp. 364 - 373, 1990 1990.

[33] M. Kandemir, F. Li, M. Irwin, and S. Son, "A novel migration-based NUCA design for chip multiprocessors," 2008, pp. 1-12.

[34] S. Kaxiras, Z. Hu, and M. Martonosi, "Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power," in *International Conference on Computer Architecture (ISCA)*, Göteborg, Sweden, 2001.

[35] C. Kim, D. Burger, and S. W. Keckler, "An Adaptive, Non-uniform Cache Structure for Wire-delay Dominated On-chip Caches," in *Proc. of the Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, San Jose, CA, 2002, pp. 211-222.

[36] J. S. Kim, M. B. Taylor, J. Miller, and D. Wentzlaff, "Energy characterization of a tiled architecture processor with on-chip networks," in *International Symposium on Low Power Electronics and Design* Seoul, Korea 2003.

[37] N. S. Kim, D. Blaauw, and T. Mudge, "Leakage Power Optimization Techniques for Ultra Deep Sub-Micron Multi-Level Caches," in *International Conference on Computer Aided Design*, 2003, p. 627.

[38] H. Lee, S. Cho, and B. R. Childers, "Exploring the Interplay of Yield, Area, and Performance in Processor Caches," in *International Conference on Computer Design (ICCD)*, Lake Tahoe, CA, 2007.

[39] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," in *MICRO 2009*, New York City, NY, 2009.

[40] X. Liang and D. Brooks, "Mitigating the Impact of Process Variations on CPU Register File and Execution Units," in *International Symposium on Microarchitecture*, Orlando, FL, 2006.

[41] X. Liang, R. Canal, G.-Y. Wei, and D. Brooks, "Process Variation Tolerant 3T1D-Based Cache Architectures," in *40th International Symposium on Microarchitecture*, Chicago, IL, 2007.

[42] X. Liang, G.-Y. Wei, and D. Brooks, "ReVIVaL: A Variarion-Tolerant Architecture Using Voltage Interpolation and Variable Latency," in *Internation Symposium on Computer Architecture*, Beijing, China, 2008.

[43] D. Marculescu and S. Garg, "System-level processdriven variability analysis for single and multiple voltagefrequency island systems," in *International Conference on Computer-Aided Design (ICCAD)*, 2006.

[44] D. Marculescu and E. Talpes, "Variability and energy awareness: a microarchitecture-level perspective.," in *Proceedings of the 42nd annual conference on Design Automation (DAC)*, New York, NY, 2005.

[45] J. Merino, V. Puente, P. Prieto, and J. Gregorio, "Sp-nuca: a cost effective dynamic non-uniform cache architecture," *ACM SIGARCH Computer Architecture News*, vol. 36, pp. 64-71, 2008.

[46] M. Miller. (Sep. 2004, *Manufacturing-aware Design Helps Boost IC Yield*. Available: http://www.eetimes.com/news/design/features/showArticle.jhtml;?articleID=47102054

[47] S. Mukherjee, P. Bannon, S. Lang, A. Spink, and D. Webb, "The Alpha 21364 network architecture," *IEEE micro*, vol. 22, pp. 26-35, 2002.

[48] S. Naffziger, G. Colon-Bonet, T. Fischer, R. Riedlinger, T. Sullivan, and T. Grutkowski, "The Implementation of the Itanium 2 Microprocessor," *IEEE Journal of Solid State Circuits*, vol. 37, November 2002 2002.

[49] S. R. Nassif, "Modeling and Analysis of Manufacturing Variations," in *IEEE Conference on Custom Integrated Circuits*, San Diego, CA, May 2001, pp. 223-228.

[50] S. Natarajan, M. A. Breuer, and S. K. Gupta, "Process Variations and their Impact on Circuit Operation," in *International Symposium on Defect and Fault Tolerance in VLSI Systems*, 1999, p. 73.

[51] K. Niyogi and D. Marculescu, "Speed and voltage selection for gals systems based on voltage/frequency islands," in *Proceedings of the 2005 conference on Asia South Pacific design automation (ASP-DAC)*, New York, NY, 2005.

[52] S. Ozdemir, D. Sinha, G. Memik, J. Adams, and H. Zhou, "Yield-Aware Cache Architectures," in *International Symposium on Microarchitecture*, Orlando, FL, 2006.

[53] M. Powell, S.-H. Yang, B. Falsa, K. Roy, and T. Vijaykumar, "Gated-Vdd: A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories," in *ACM/IEEE Intl. Symposium on Low Power Electronics and Design*, Rapallo, Italy, 2000, pp. 90-95.

[54] S. Raj, S. B. K. Vrudhula, and J. Wang, "A Methodology to Improve Timing Yield in the Presence of Process Variations," in *Proc. of the Conf. on Design Automation*, San Diego, CA, 2004, pp. 448-453.

[55] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis, "Evaluating mapreduce for multi-core and multiprocessor systems," *In HPCA*, vol. 7, pp. 13–24.

[56] R. Rao, D. Blaauw, D. Sylvester, and A. Devgan, "Modeling and Analysis of Parametric Yield under Power and Performance Constraints," *IEEE Des. Test*, vol. 22, pp. 376-385, 2005 2005.

[57] R. Rao, A. Srivastava, D. Blaauw, and D. Sylvester, "Statistical Estimation of Leakage Current Considering Inter- and Intra-Die Process Variation," in *ISLPED '03*, Seoul, Korea, 2003.

[58] A. Raychowdhury, S. Ghosh, S. Bhunia, D. Ghosh, and K. Roy, "A Novel On-chip Delay Measurement Hardware for Efficient Speed Binning," in *Intl. Online Testing Symposium*, France, Jul. 2005.

[59] R. Ricci, S. Barrus, D. Gebhardt, and R. Balasubramonian, "Leveraging bloom filters for smart search within NUCA caches," 2006.

[60] B. F. Romanescu, M. E. Bauer, D. J. Sorin, and S. Ozev, "A Case for Computer Architecture Performance Metrics that Reflect Process Variability," Duke University, Dept. of ECEMay 2007 2007.

[61] A. Ros, M. Acacio, and J. García, "DiCo-CMP: Efficient cache coherency in tiled CMP architectures," 2008, pp. 1–11.

[62] L. Shang, L.-S. Peh, and N. K. Jha, "Dynamic Voltage Scaling with Links for Power Optimization of Interconnection Networks," in *Proceedings of the 9th International Symposium on High-Performance Computer Architecture (HPCA)*, Anaheim, CA, 2003.

[63] L. Shang, L. Peh, and N. Jha, "PowerHerd: dynamic satisfaction of peak power constraints in interconnection networks," in *ICS*, 2003, p. 108.

[64] L. Shang, L. Peh, A. Kumar, and N. Jha, "Thermal modeling, characterization and management of on-chip networks," in *IEEE MICRO*, 2004, pp. 67-78.

[65] T. Sherwood, B. Calder, and J. Emer, "Reducing cache misses using hardware and software page placement," 1999, p. 164.

[66] P. Shivakumar, S. Keckler, C. Moore, and D. Burger, "Exploiting microarchitectural redundancy for defect tolerance," in *International Conference on Computer Design (ICCD)*, 2003, pp. 481-488.

[67] SimpleScalarLLC, "The SimpleScalar Tool Set," ed, 2001.

[68] G. S. Sohi, "Cache Memory Organization to Enhance the Yield of High Performance VLSI Processors," *IEEE Trans. Comput.*, vol. 38, pp. 484-492, 1989.

[69] SPEC, "Spec CPU2000: Performance Evaluation in the New Millennium v1.1," Dec. 2000.

[70] Sun. *OpenSPARC T1*. Available: http://opensparc-t1.sunsource.net/index.html

[71] S. H. Tadas and C. Chakrabarti, "Architectural approaches to reduce leakage energy in caches," in *International Symposium on Circuits and Systems*, 2002.

[72] D. Tam, R. Azimi, L. Soares, and M. Stumm, "Managing shared L2 caches on multicore systems in software," 2007.

[73] R. Teodorescu and J. Torrellas, "Variation-Aware Application Scheduling and Power Management for Chip Multiprocessors," in *International Symposium on Computer Architecture (ISCA)* Beijing, China, 2008.

[74] The_R_Foundation. *The R Project for Statistical Computing*. Available: http://www.r-project.org/

[75] A. Tiwari, S. R. Sarangi, and J. Torellas, "ReCycle: Pipeline Adaptation to Tolerate Process Variation," in *International Symposium on Computer Architecture*, San Jose, CA, 2007.

[76] L. Wang, A. Nichelatti, H. Schellevis, C. de Boer, C. Visser, T. Nguyen, and P. Sarro, "High aspect ratio through-wafer interconnections for 3D-microsystems," in *MICRO*, 2003, pp. 634-637.

[77] T. Wenisch, R. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. Hoe, "SimFlex: statistical sampling of computer system simulation," *IEEE MICRO*, vol. 26, p. 18, 2006.

[78] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: characterization and methodological considerations," presented at the Proceedings of the 22nd annual international symposium on Computer architecture, S. Margherita Ligure, Italy, 1995.

[79] S. Yang, M. Powell, B. Falsafi, K. Roy, and T. N. Vijaykumar, "An integrated circuit/architecture approach to reducing leakage indeep-submicron high-performance I-caches," in *International Symposium on High-Performance Computer Architecture*, 2001, pp. 147-157.

[80] S. Yang, M. D. Powell, B. Falsafi, and T. N. Vijaykumar, "Exploiting choice in resizable cache design to optimize deep-submicron processor energy-delay," in *International Symposium on High-Performance Computer Architecture*, 2002, pp. 151- 161.

[81] T. Yeh and G. Reinman, "Fast and fair: data-stream quality of service," 2005, p. 248.

[82] J. Zebchuk, V. Srinivasan, M. Qureshi, and A. Moshovos, "A tagless coherence directory," 2009, pp. 423-434.

[83] M. Zhang and K. Asanovic, "Victim replication: Maximizing capacity while hiding wire delay in tiled chip multiprocessors," in *ISCA*, 2005, pp. 336-345.

[84] Z. Zhang and J. Torrellas, "Reducing remote conflict misses: NUMA with remote cache versus COMA," 1997, p. 272.