EXPLOITING NATURAL ON-CHIP REDUNDANCY FOR ENERGY EFFICIENT MEMORY AND COMPUTING

Author:
ALEXANDRA FERRERÓN LABARI

Supervisors:
DR. JESÚS ALASTRUEY BENEDÉ
DR. DARÍO SUÁREZ GRACIA

DISSERTATION
Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Universidad de Zaragoza

Grupo de Arquitectura de Computadores
Departamento de Informática e Ingeniería de Sistemas
Instituto de Investigación en Ingeniería de Aragón
Universidad de Zaragoza

October, 2016

Universidad Zaragoza

ABSTRACT

Power density is currently the primary design constraint across most computing segments and the main performance limiting factor. For years, industry has kept power density constant, while increasing frequency, lowering transistors supply (V_{dd}) and threshold (V_{th}) voltages. However, V_{th} scaling has stopped because leakage current is exponentially related to it. Transistor count and integration density keep doubling every process generation (Moore's Law), but the power budget caps the amount of hardware that can be active at the same time, leading to *dark silicon*. With each new generation, there are more resources available, but we cannot fully exploit their performance potential. In the last years, different research trends have explored how to cope with dark silicon and unlock the energy efficiency of the chips, including Near-Threshold voltage Computing (NTC) and approximate computing.

NTC aggressively lowers V_{dd} to values near V_{th}. This allows a substantial reduction in power, as dynamic power scales quadratically with supply voltage. The resultant power reduction could be used to activate more chip resources and potentially achieve performance improvements. Unfortunately, V_{dd} scaling is limited by the tight functionality margins of on-chip SRAM transistors. When scaling V_{dd} down to values near-threshold, manufacture-induced parameter variations affect the functionality of SRAM cells, which eventually become not reliable.

A large amount of emerging applications, on the other hand, features an intrinsic error-resilience property, tolerating a certain amount of noise. In this context, approximate computing takes advantage of this observation and exploits the gap between the level of accuracy required by the application and the level of accuracy given by the computation, providing that reducing the accuracy translates into an energy gain. However, deciding which instructions and data and which techniques are best suited for approximation still poses a major challenge.

This dissertation contributes in these two directions. First, it proposes a new approach to mitigate the impact of SRAM failures due to parameter variation for effective operation at ultra-low voltages. We identify two levels of natural on-chip redundancy: cache level and content level. The first arises because of the replication of blocks in multi-level cache hierarchies. We exploit this redundancy with a cache management policy that allocates blocks to entries taking into account the nature of the cache entry and the use pattern of the block. This policy obtains performance improvements between 2% and 34%, with respect to block disabling, a technique with similar complexity, incurring no additional storage overhead. The latter (content level redundancy) arises because of the redundancy of data in real world applications. We exploit this redundancy compressing cache blocks to fit them in partially functional cache entries. At the cost of a slight overhead increase, we can obtain performance within 2% of that obtained when the cache is built with fault-free cells, even if more than 90% of the cache entries have at least a faulty cell.

Then, we analyze how the intrinsic noise tolerance of emerging applications can be exploited to design an approximate Instruction Set Architec-

ture (ISA). Exploiting the ISA redundancy, we explore a set of techniques to approximate the execution of instructions across a set of emerging applications, pointing out the potential of reducing the complexity of the ISA, and the trade-offs of the approach. In a proof-of-concept implementation, the ISA is shrunk in two dimensions: *Breadth* (i.e., simplifying instructions) and *Depth* (i.e., dropping instructions). This proof-of-concept shows that energy can be reduced on average 20.6% at around 14.9% accuracy loss.

PUBLICATIONS

Some ideas and figures have appeared previously or are currently under review in the following publications:

[1] A. Ferrerón. *Exploiting Redundancy for Efficient Cache Operation at Near-threshold Voltages*. ACM Student Research Competition at Grace Hopper Celebration. 2015.
Acceptance ratio 19/79 (24%), Bronze Medal.

[2] A. Ferrerón, D. Suárez-Gracia, J. Alastruey-Benedé, T. Monreal-Arnal, and V. Viñals-Yúfera. "Block Disabling Characterization and Improvements in CMPs Operating at Ultra-low Voltages." In: *26th Int. Symp. on Computer Architecture and High Performance Computing*. 2014, pp. 238–245. DOI: 10.1109/SBAC-PAD.2014.12.
B CORE 2014; Google h5-index 12; Acceptance ratio 32.6%.

[3] A. Ferrerón, D. Suárez-Gracia, J. Alastruey-Benedé, T. Monreal-Arnal, and V. Viñals-Yúfera. "Low Complexity Improvements for CMPs Shared Caches at Ultra-low Voltages." In: *10th International Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems*. HiPEAC, 2014, pp. 73–76.

[4] A. Ferrerón, J. Alastruey-Benedé, D. Suárez-Gracia, T. Monreal-Arnal, P. Ibáñez, and V. Viñals-Yúfera. "A Fault-aware Cache Management Policy for CMPs Operating at Ultra-low Voltages." Submitted to IEEE Trans. on Parallel and Distributed Sytems. July 2016.
IF: 2.661; JCR 2015: Q1 (9/105) Computer Science, Theory & Methods; Google h5-index 76 (1st in Computing Systems).

[5] A. Ferrerón, J. Alastruey-Benedé, D. Suárez-Gracia, and U. R. Karpuzcu. "AISC: Approximate Instruction Set Architecture." Submitted to 2017 Design, Automation & Test in Europe Conf. Exhibition. Sept. 2016.
B CORE 2014; Google h5-index 39 (9th in Computer Hardware Design).

[6] A. Ferrerón, D. Suárez-Gracia, J. Alastruey-Benedé, T. Monreal-Arnal, and P. Ibáñez. "Concertina: Squeezing in Cache Content to Operate at Near-Threshold Voltage." In: *IEEE Trans. on Computers (Special Section on Defect and Fault Tolerance in VLSI and Nanotechnology Systems)* 65.3 (2016), pp. 755–769. DOI: 10.1109/TC.2015.2479585.
IF: 1.723; JCR 2015: Q1 (12/51) Computer Science, Hardware & Architecture; Google h5-index 45 (4th in Computer Hardware Design).

[7] A. Ferrerón, J. Alastruey-Benedé, D. Suárez-Gracia, T. Monreal-Arnal, P. Ibáñez, and V. Viñals-Yúfera. "Gestión de contenidos en caches operando a bajo voltaje." In: *XXVII Jornadas de Paralelismo*. 2016, pp. 497–506.

ACKNOWLEDGMENTS

I want to start by thanking my supervisors, Chus and Darío, for their help and guidance during these years, and with whom during this time I have built not only a professional, but also a relationship of friendship. This dissertation had not seen the light without Pablo Ibáñez devoted guidance and work, who came to the rescue in the time I most needed someone like him. Thanks also to my co-authors and to the rest of the members of the computer architecture group. To all of you, I hope you enjoyed working with me as much as I enjoyed working with you.

Thanks to the external reviewers Rekai (Rodrigo Gónzalez-Alberquilla) and Adrian Sampson. I really appreciate the time you dedicated to thoroughly read my dissertation and your helpful suggestions.

I also want to thank to the supervisors of my research stays at Edinburgh, Minnesota, and Cambridge, and their teams, who welcomed me and made me feel like home. Mike O'Boyle gave me some great advice that has helped me to arrive to where I am, taking difficult decisions on the way. Ulya Karpuzcu has been an example of work, tenacity, and self-improvement, and became a great support during the last part of my PhD. Roxana Rusitoru, thanks for making my stay at ARM a great and really fun experience.

On the technical part, I have to thank Jorge Albericio, Marta Ortín, and Javier D. Maag for their help with the simulation infrastructure and simulation methodology.

Special thanks to all my friends and colleagues who have been there during all these years, supporting and encouraging me at every step of the way. I hope you forgive me for not naming you all here. María and Alejandro, you are family to me; thanks for being always there.

Finally, I have to thank my family, the ones that are and the ones that left, for all these years of support and patience, always believing in me. And Markus, who has shared many sacrifices, including the pain of reading this dissertation.

And with all my heart, I dedicate this work to Rosa, whose light slowly faded, and who I will always miss; wherever I go, I will always have the memory of your love with me.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Part I

PRELIMINARIES

1 INTRODUCTION

It would appear that we have reached the limits of what it is possible to achieve with computer technology, although one should be careful with such statements, as they tend to sound pretty silly in 5 years.

JOHN VON NEUMANN, 1949

1.1 RATIONALE

For the last 50 years, industry has enjoyed a long run of continuous technology improvements fueled by Moore's Law [114] and Dennard Scaling [46]. Scaling until 65 nm was rather straightforward, improving performance and power, while providing, at the same time, economic benefits. With each new generation, chips became smaller, more powerful, and cheaper to produce. However, semiconductor design has gotten more complicated, and after 22 nm, double patterning [138], 3D transistors [47], or new substrate materials such as FD-SOI [63] came into play, raising the production costs without yielding obvious power or performance benefits. With high-volume manufacturing at 7 nm scheduled for 2018 to 2019—and still not a clear substitute for CMOS—industry faces a complete paradigm shift: from designing for performance, to designing to satisfy the thermal design power of the end product.

Dennard Scaling Law has driven the semiconductors industry for the last decades. It states that as transistors get smaller, their power density stays constant, as both voltage and current scale down with length. In particular, the performance of a transistor can improve while preserving its operational characteristics if the parameters of a device (i.e., dimensions, voltages, and doping concentration densities) are scaled by a dimensionless factor S (Table 1.1). This technique is also called *constant field scaling* because as both voltage and distance shrink, the electric fields remain the same.

Industry generally scales process generations with $S = \sqrt{2}$ (also called a 30% shrink). This way, the area of a transistor is reduced by a factor of 2. A 30% shrink with Dennard scaling translates into a 40% improvement in clock frequency, and it cuts power consumption per gate by a factor of 2. This power decrease allows to integrate more transistors in accordance to Moore's Law, keeping the same power budget. Over the last four decades, microprocessors featured size has improved more than two orders of magnitude, transistor budgets have multiplied by more than five orders of magnitude, and clock frequencies have multiplied more than three orders of magnitude.

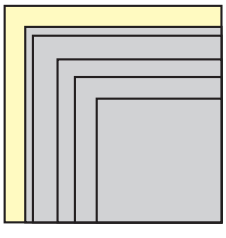
Unfortunately, CMOS scaling may be ultimately limited by leakage power. Scaling supply voltage (V_{dd}) usually entails scaling the threshold voltage (V_{th}) to keep performance[1]. But as V_{th} decreases, subtreshold leakage power

1 V_{th} is the minimum transistor's gate-to-source voltage differential needed to create a conducting path between the source and drain terminal; the gate delay time decreases as the ratio V_{th}/V_{dd} does.

increases exponentially [156]. In the post-Dennard regime, smaller devices are more prone to leakage, and subtreshold leakage has changed from being negligible, to being a substantial fraction of the total power, reaching the same magnitude as the circuit's dynamic power [86]. Thus, V_{dd} and V_{th} are no longer scaling parameters, rather they are set by the results of a power optimization. V_{dd} and V_{th} have remained fairly constant (around 1 V and 300 mV, respectively) since the 65 nm node to limit leakage. In addition to this, variability increases as transistor dimensions shrink, as described by Pelgrom's model: the variance on the threshold voltage, the current factor, and the substrate factor are inversely proportional to the transistor area [121]. With each new process generation the complexity increases, reliability issues exacerbate, and new problems arise.

At the 65 nm node and beyond, scaling benefits, in terms of performance and power, started to diminish, but the engineering costs keep escalating. Scaling still provides a competitive advantage in a cutthroat industry, and vendors have taken advantage of the extra transistors integrating several processor cores on the chip in the form of homogeneous and heterogeneous chip multiprocessors (CMPs). However, integrating more resources on-chip does not mean that all the resources can be used simultaneously, because the power budget caps the amount of hardware that can be active at the same time. Hence, an exponentially increasing fraction of the chip area remains underclocked or dark—hence the term *dark silicon* [53, 147].

Table 1.1 describes the differences between Dennard and post-Dennard scaling. In the post-Dennard scaling era, for each new process generation, the total chip utilization for a fixed power budget drops by S^2, as a consequence of non-scaling V_{dd} and V_{th} parameters. The effect is illustrated in Figure 1.1, which shows ARM's prediction on dark silicon (assuming no design mitigation) for the next process generations [165]. The shaded regions on Figure 1.1 represent the percentage of dark silicon (i.e., transistors that cannot be turned on) expected at different technology nodes, with respect to a 28 nm reference process chip's power budget. Dark silicon is predicted to reach 80% of the chip's silicon at the 5 nm process[2]. Real products are likely to achieve better results through technology and design innovations, but power consumption is clearly a severe design constraint.



Transistor Property	Dennard	Post-Dennard
$\triangle$ Quantity	S^2	S^2
$\triangle$ Frequency	S	S
$\triangle$ Capacitance	$\frac{1}{S}$	$\frac{1}{S}$
V_{dd}^2 & V_{th}^2	$\frac{1}{S^2}$	1
$\triangle$ Power = $\triangle QFCV^2$	1	S^2
$\triangle$ Utilization = $\frac{1}{Power}$	1	$\frac{1}{S^2}$

Figure 1.1 & Table 1.1: Dark silicon percentage with no design mitigation ([165]) as a consequence of post-Dennard scaling.

Improving system efficiency in the dark silicon era, without neglecting performance improvements, has led designers toward heterogeneous solutions [139]. These solutions include CMPs with multiple voltage domains [62,

2 Dark silicon should not be understood as "useless" silicon, but as silicon that it is not used all the time or at its full frequency. For example, caches are dark silicon friendly: they contribute to the performance of the system, consuming power only on special situations.

129], integrated graphics processing units (GPUs) [22, 45], heterogeneous multi-cores with specialized accelerators [59], and intensive industry and academia research in novel technologies, including novel memory structures [112] and 3D integration [71]. We can expect to see an increase in the amount of resources dedicated to accelerators, and architectural changes focus on exploiting the characteristics of the emerging applications, such as approximate computing [113], accelerators for machine learning [97], or processing in memory [99]. In this dissertation, we focus on two trends to improve the energy efficiency of present and future systems: near-threshold voltage computing and approximate computing.

A very effective way of reducing the on-chip power consumption is reducing the supply voltage. Reducing V_{dd} to values near the threshold voltage—a technique known as *Near-Threshold voltage Computing* (NTC)—can decrease energy per operation up to 10x, at the cost of slower devices [30, 49]. The resulting power reduction could be used to activate more chip resources and potentially achieve performance improvements. Unfortunately, one of the main limitations of V_{dd} scaling is the tight margins of the on-chip Static Random Access Memory (SRAM) cache transistors. Excessive parameter variations in SRAM cells limit the voltage scaling of memory structures to a minimum voltage, below which SRAM cells may not operate reliably, and show a behavior consistent with a hard-fault. In this context, micro-architectural techniques can alleviate the performance and energy degradation that appears as a consequence of the reduction in the number of functional cache entries. In particular, the organization of the memory hierarchy on-chip and the applications running on the CMP themselves are a natural source of redundancy, offering a great opportunity to counteracting the performance degradation that appears as a result of the cache capacity degradation. SRAM reliability at near-threshold voltages is the focus of Part II of this dissertation, where we present micro-architectural techniques to mitigate the impact of SRAM cell failure when running at voltages near the threshold, based on exploiting the intrinsic redundancy of the memory hierarchy and the applications.

At the same time, as semiconductor industry advances and circuits become more sensitive to parameter variations and faults, it is highly energy inefficient to ensure fault-free computations. Fortunately, an increasingly large number of emerging applications, which are commonly referred as Recognition, Mining, and Synthesis (RMS), feature an intrinsic error-resilience property [37]. These applications often process noisy inputs (e.g., from sensors), run iterative probabilistic algorithms, and they usually do not require to compute a unique valid output, rather many outputs might be "acceptable" or "precise enough" [35]. Thus, *approximate computing* has drawn significant attention over the last years [113, 161]. The idea is to allow certain approximation or occasional violation of the specifications of a given algorithm, exploiting the gap between the level of accuracy required by the application and that provided by the computing system, potentially unlocking large energy gains. The main obstacle for approximate computing is effectively selecting which parts of the application code and data can be approximated [133]. Wrong decisions might lead to unacceptable results, which need to be re-computed, diminishing any potential energy benefit, or even worse, applications might crash or incur in security violations. Approximate computing is the focus of Part III of this dissertation, where we explore the possibilities of approximating the Instruction Set Architecture

(ISA) itself, and analyze the impact of several ISA-shrinkage techniques on energy and accuracy.

1.2 OBJECTIVES AND DISSERTATION OVERVIEW

The overall goal of this dissertation is to explore new approaches to improve the energy efficiency of future systems in the dark silicon era. We explore sources of natural on-chip redundancy and take advantage of them in the context of computing at voltages near the threshold and approximate computing. The rest of this dissertation is organized as follows:

- Part II presents our micro-architectural proposals to mitigate the impact of SRAM cell failures in last-level caches caused by process variations. It starts with an introduction to near-threshold voltage computing, discussing the reliability issues associated with it and the related work in Chapter 2. Chapter 3 details our methodology. Part II also includes two proposals, both based on taking advantage of the natural redundancy of content in the cache hierarchy. Chapter 4 explores how to take advantage of the memory hierarchy organization to minimize the performance loss in the presence of faulty cells, with minimal changes in the coherence protocol, by means of an intelligent cache management policy; Chapter 5 presents a novel approach based on compressing cache blocks, so they fit into cache entries with faults.
- Part III explores approximation at the ISA level to improve the energy efficiency of processors. As Part II, it begins with an introduction to approximate computing in Chapter 6, where we discuss the reasons of the increasing popularity of approximate computing and cover the related work. The methodology is discussed in Chapter 7. We explore the possibilities of approximating the ISA in several dimensions, e.g., simplifying instructions (Breadth) or even dropping them (Depth), and present a proof-of-concept in Chapter 8.
- Part IV concludes and discusses future research lines in Chapter 9.

1.3 CONTRIBUTIONS

At the time this dissertation is being written, a large part of the work presented here has been published in peer-reviewed national and international conferences and journals, or is currently under review process.

The main contributions of this dissertation are:

- Contributions on SRAM reliability at near-threshold voltages (Part II):
  - Based on block disabling techniques [142], we introduce a low complexity mechanism for inclusive cache hierarchies that permits cache blocks to use the non-functional last-level cache (LLC) data entries, by keeping operational the tag array. Thus, blocks allocated to disabled entries can be present in the private caches. We propose a fault-aware cache management policy that predicts the usefulness of a block based on its use pattern, and guides block allocation to faulty and non-faulty cache entries, accordingly.
    This work has been published in the *26th International Symposium on Computer Architecture and High Performance Computing (October*

2014). An extended version of this work has been submitted to *IEEE Transactions on Parallel and Distributed Systems (submitted July 2016)*.

  - We provide the first analysis of requirements for compression techniques for shared LLCs running at ultra-low or near-threshold voltages. We propose the use of compression to improve cache utilization at ultra-low voltages, with a low-complexity combination- and remapping-free mechanism, maintaining the regular nature of SRAM cells. We present and evaluate cache management policies that match compressed blocks with faulty cache entries.
    This work has been published in *IEEE Transactions on Computers, Special Section on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (March 2016)*.

- Contributions on approximate computing (Part III):
  - We propose to reduce the complexity of the ISA by shrinking it in two dimensions (Breadth and Depth) and approximate the execution of emerging RMS applications, exploiting the intrinsic noise tolerance of their algorithms. We present a proof-of-concept implementation and analyze the sensitivity of these applications to Breadth and Depth ISA shrinkage.
    This work has been developed in collaboration with the Altai group of the University of Minnesota, and it has been submitted to *Design, Automation, and Test in Europe 2017 (submitted September 2016)*.

1.4 PROJECT FRAMEWORK

The realization of this dissertation was possible thanks to the support of the following awards and fellowships:

- 4-year PhD FPI grant from Spanish *Ministerio de Ciencia e Innovación*, including two complementary short stay grants:
  - Compiler and Architecture Design Group, University of Edinburgh, United Kingdom (April - October 2013).
  - Altai Lab, University of Minnesota, Twin Cities, United States of America (May - August 2015).
- 4-month research stay at ARM Research, co-funded by HiPEAC PhD industrial internship program (October 2015 - January 2016)
- Google Anita Borg Memorial Scholarship[3], including €7,000 donation.
- Travel and assistance grants: Grace Hopper Celebration (2015, ABI, ACM, & Google), ISCA Conference (2015, IEEE), SBAC-PAD Conference (2014, U. Zaragoza), PACT Conference (2013, IEEE), ARCS Conference (2013, U. Zaragoza), HiPEAC Conference (2012, HiPEAC), JJ-PAR Conference (2012, U. Zaragoza) ACACES Summer School (2011 and 2014, HiPEAC).

Part of the work of this dissertation has been awarded the 3rd position at the ACM Student Research Competition[4] at the Grace Hopper Celebration of Women in Computing (October 2015), which included a $300 donation.

3 http://www.google.com/anitaborg/emea/
4 http://src.acm.org/

This dissertation was realized in the framework of the following projects: *Interconexión y Memoria en Computadores Escalables* (TIN2010-212911) and *Memoria, Interconexión y Aplicaciones para Computadores Eficientes* (TIN2013-46957). HiPEAC has been a great supporter funding two Summer Schools, a conference grant, and partially funding an internship at ARM Research.

Part II

EXPLOITING REDUNDANCY FOR EFFICIENT CACHE OPERATION AT NEAR-THRESHOLD VOLTAGES

2

FROM DARK TO DIM SILICON: COMPUTING AT VOLTAGES NEAR THE THRESHOLD

One of the main obstacles to voltage scaling to values near the threshold is the minimum voltage supply of the SRAM on-chip structures. At lower voltages, devices are more sensitive to process variations, which impact circuit functionality, and eventually limit supply voltage reduction. SRAM cells are specially vulnerable because they are aggressively sized to meet high density requirements. In this chapter, we introduce near-threshold voltage computing and its challenges, including variability on SRAM cells at low supply voltage. Then, we present our reference SRAM reliability model and cover the related work.

2.1 INTRODUCTION

Voltage scaling is one of the most effective ways to reduce the chip power consumption, as dynamic power scales quadratically with voltage. Modern chips implement Dynamic Voltage and Frequency Scaling (DVFS) schemes to run at different predefined voltage points and frequencies, trading off power consumption and performance, with a fixed (non-scaling) V_{th} [65]. However, concerns about robustness and performance set the lower bound of supply voltage around 70% of the nominal V_{dd} in commercial applications. As a matter of fact, CMOS circuits operate at very low voltages, and remain functional even at voltages below the threshold ($V_{dd} < V_{th}$)—this is called subthreshold computing—, but due to the large performance loss, their application is restricted to niche markets [85, 141, 154].

Figure 2.1 shows the energy per operation and delay when supply voltage scales down. By reducing V_{dd} from a nominal 1 V (V_{nom} in Figure 2.1) to 400-500 mV (V_{nth}), we can obtain as much as 10x energy efficiency gain without incurring in the exponential performance degradation of subtreshold computing ($V_{dd} < 300$ mV). This technique is referred to as Near-Threshold voltage Computing (NTC) [49]. While reducing V_{dd} below V_{th} reaches the minimum point of energy per operation, the consequent performance degradation may be unacceptable for the vast majority of applications [23].

Researchers have looked at x86-based NTC implementations [67] and 3D many-core ARM-based implementations [48], but several key challenges have prevented the general adoption of NTC, namely: i) 10x loss in performance due to the frequency degradation, ii) 5x increase in performance variation due to parameter variations, and iii) five orders of magnitude increase in functional failure rate of memory as well as increased logic failures.

To compensate for the performance loss, the power budget resulting from scaling V_{dd} can be used to switch on more cores. Throughput applications, such as graphics-like workloads with minimal emphasis on each thread, or parallel applications, such as high-performance computing ones, are the strongest matches for NTC. These applications can take advantage of a big pool of cores, even if they run at slower frequencies. As power consumption decreases more from operating at low V_{dd} than it increases from operating

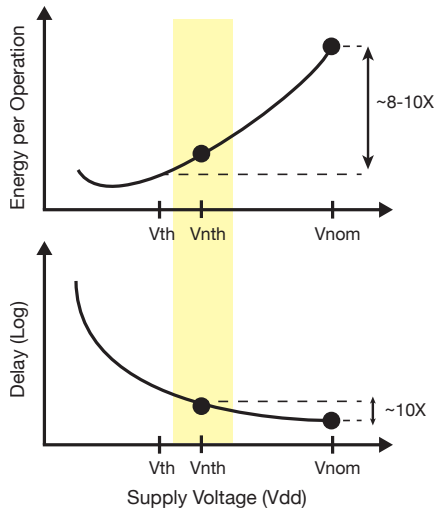


Figure 2.1: Energy per operation and delay with respect to supply voltage.

more cores in parallel, the result is still a power reduction [30]. Single-thread performance could still be achieved with voltage/frequency boosting of a low-power core, or by a high-performance core added to the system [122].

Parameter variations—the deviation of device parameters from their nominal specifications—are already present at nominal V_{dd} [20]. The sources of parameter variations include manufacturing-induced sources (process variations) and environmental ones (supply voltage and temperature). CMOS gates are robust, and they generally work correctly when parameters vary enormously. However, in the presence of parameter variations, chips have regions that run at different speed and have different power consumption. Designers usually account for parameter variations inserting conservative margins or guardbands to cover against the worst-case variation, increasing nominal voltage up to 20% [70]. This worst-case guided design incurs in large performance and power losses, as the worst case might be far from the nominal specifications, and the worst case condition can be severe, but infrequent [126]. At NTC the problem exacerbates, because the same amount of variation causes larger changes in the device speed and power [106], and relying on the worst-case operating margins is not practical, as frequency is already low. Architectural and circuit solutions can provide variation tolerance and adaptivity [49, 78, 79].

Next, we cover the main causes of functional failure of on-chip memory structures when running at near-threshold voltages.

2.2 PROCESS VARIATIONS IN SRAM CELLS

Static Random Access Memory (SRAM) is the most widely used form of on-chip memory. They are faster than their dynamic counterparts (DRAM), avoiding the periodic refresh phases, but require more area per bit. SRAMs use a memory cell with internal feedback that retains its value as long as a power is applied. A traditional 6T SRAM cell, such as the one shown in Figure 2.2, consists of four transistors that form two cross-coupled inverters (PL-NL and PR-NR), and two access transistors (AXL and AXR).

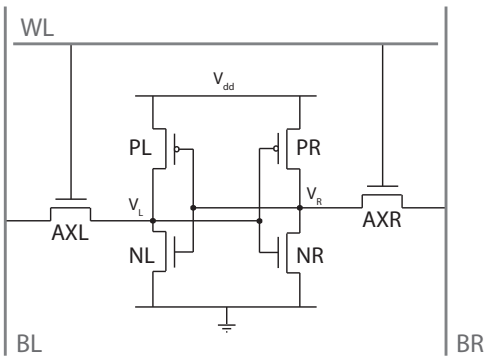


Figure 2.2: 6T SRAM cell architecture.

A 6T SRAM cell is susceptible to five types of failures: *read upset*, during read operation, the cell flips its content; *read access*, during read operation, the time needed to produce a voltage difference between the two bitlines, BL and BR, exceeds the period that the wordline, WL, stays high; *write stability*, during write operation, even if the write duration is extended to infinity, the cell cannot change its logic state; *write timing*, during write operation, the cell is unable to change its logic state by the end of the designated time duration; and *hold*, the value of the cell is flipped by excessive leakage on the constituent transistors. To ensure both read stability and writability, the transistors must satisfy ratio constraints. The stability and writability of the cell are quantified by the hold margin, the read margin, and the write margin, which are determined by the static noise margin of the cell in its various modes of operation. The hold margin increases with V_{dd} and V_{th}; read margin improves by increasing V_{dd} or V_{th}, or by reducing the WL voltage relative to V_{dd}; the write margin improves as the access transistor becomes stronger, the pull-up becomes weaker, or the wordline voltage increases. Therefore, write and read improvements are conflicting with each other.

SRAM structures are especially vulnerable to process variations [18], since they are aggressively sized to meet high density requirements, and due to the vast number of cells that comprise the on-chip memory structures, which nowadays account for even more than half of the chip area [26]. Process variations can be classified into inter-die and intra-die. Inter-die variations eventually cause performance differences across chips, and they can be handled by adding enough margin and rejecting chips that do not meet certain specifications. Intra-die variations are gaining importance as technology scales, because chips integrate millions or billions of transistors, and, very likely, some of them will fail outside the specification corners. Intra-die variation has a systematic and a random component. The systematic component is typically caused by lithographic irregularities, mainly affecting channel length. The random component is caused by varying dopant fluctuations (Random Dopant Fluctuation or RDF), and it is the main cause of V_{th} variation [146]. Specifically, V_{th} is determined by the number and location of dopant atoms implanted in the transistor channel region. The stochastic nature of ion implantation leads to different V_{th} values across the chip.

The resultant threshold mismatch reduces the available cell noise margins, creating a distribution of read, write, and hold margins across the SRAM

structure, and some of the cells become unstable. An unstable cell flips during a read operation, fails to switch during a write operation, or loses its state during hold, causing a circuit behavior consistent with a hard fault. Because the static noise margins depend on V_{dd}, SRAMs have a minimum voltage at which they can reliably operate. This voltage is $V_{dd_{min}}$ and it is typically of the order of 0.7–1.0 V in current technology nodes, when 6T cells are employed, and presents an obstacle to continued voltage scaling.

Modern CMPs implement multi-level on-chip SRAM cache hierarchies, usually composed of two or three cache levels, with different cell design objectives, depending on the importance of performance and leakage. In general, first-level caches occupy little area and their access time often determines the processor cycle time; circuit techniques such as larger 8T SRAM cells [29] are utilized to minimize the impact of variation [91]. Last-level caches (LLCs), on the other hand, have larger size and associativity, and occupy a great percentage of the chip area; smaller and less leaky cells are preferred to achieve more density, but the structure is more prone to errors due to variability [18].

To allow circuit operation at lower V_{dd}, some commercial processors have different voltages for logic and memory. For instance, Intel® Medfield operates the L2 cache at 1.05 V, while the CPU cores can scale down to 0.7 V [166]. Separate voltage domains complicate chip design [105], and scaling will likely increase the differences between voltage domains for logic and memory, eventually increasing the number of voltage domains required, and diminishing the power reduction benefit of operating chips at lower voltages. Our objective is to reduce the supply voltage of the whole chip and mitigate the impact of process variations on the SRAM cells of the large on-chip cache structures (i.e., the LLC). We achieve this objective through micro-architectural techniques, which take into account the organization of the memory hierarchy within a CMP, and the nature of the applications running on them. We identify two levels of natural on-chip data redundancy: cache level and content level. The first arises because of the natural replication of blocks in inclusive cache hierarchies, designed to exploit the temporal locality of the data. The latter arises because of the redundancy of data in real world applications. We exploit these two levels of redundancy through fault-aware cache management policies, relying on the underlying coherence protocol and replacement policy.

2.2.1 *Reference SRAM Reliability Study*

Many authors have extensively analyzed the robustness of SRAM cells under the $V_{dd_{min}}$ range [3, 4, 24, 27, 32, 77, 83, 167]. Zhou *et al.* describe six 6T SRAM cells of different sizes in 32 nm technology, and how their probabilities of failure change with the voltage supply and the size of the constituent transistors [170]. According to Zhou *et al.*, at 0.5 V (our target near-threshold voltage) the probability of failure (P_{fail}) of a standard SRAM cell ranges between 10^{-3} and 10^{-2}. Larger cells have a lower probability of failure because non-uniformities in channel doping average out with larger transistors, resulting in more robust devices, but at the price of larger area, and higher energy consumption. Table 2.1 describes the six SRAM cells of Zhou's study (C1, C2, C3, C4, C5, and C6) with their areas relative to the smallest cell (C1), as well as the percentages of non-faulty entries of a cache implemented with

the cells at 0.5 V, assuming 64-byte cache entries[1]. An entry is considered faulty if it contains at least one defective bit. Failures are randomly distributed, but uniform within a given chip [2, 36], so they can be modeled following the Bernouilli distribution with $p = p_{fail}$. As Table 2.1 shows, only 10% of the cache entries are non-faulty for the small C2 cell at our target voltage of 0.5 V. If the cache is implemented with the more robust C6 cells, the percentage of non-faulty cache entries rises to 60%, but at the cost of a 41.1% increase in area (relative to C2) and the consequent increase in leakage, which is not a viable option for a large structure such as the on-chip LLC.

Table 2.1: Area relative to cell C1 and percentage of non-faulty 64-byte entries in a cache operating at 0.5 V, for the six bit cells introduced in [170].

Cell type	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**
Relative area	1.00	1.12	1.23	1.35	1.46	1.58
% non-faulty	0.0	9.9	27.8	35.8	50.6	59.9

In this dissertation, we use Zhou's study as SRAM reliability model to test our proposals on a wide range of failure probabilities. We consider C2 to C6 operating at 0.5 V, as at such voltage, a cache built with C1 cells would have all its capacity compromised.

2.3 RELATED WORK

Solutions proposed to date to address the variability in SRAM cells at ultra-low voltages can be arranged into two groups: circuit and micro-architectural techniques.

2.3.1 *Circuit Solutions*

Circuit solutions include proactive methods that improve the bit cell characteristics by increasing the SRAM cell size or adding assist/spare circuitry.

Larger transistors reduce V_{th} variability, since nonuniformities in channel doping average out, and result in more robust devices with a lower probability of failure [170]. Another approach to reducing variability is to add assist read/write circuitry by increasing the number of transistors per SRAM cell. Some examples are 8T [29], 10T [24], or Schmitt Trigger-based (ST) SRAM cells [90]. Increasing the SRAM cell size or the number of transistors per cell comes at the cost of significant increases in the SRAM area (lower density) and in energy consumption. For example, the use of ST SRAM cells doubles the area of the SRAM structure, which is not practical for large structures such as on-chip LLCs.

Spare rows/columns can be used to replace faulty rows/columns and improve yield [137], but this technique has obvious limitations in the amount of faulty rows/columns it can handle, due to resource limitations, area scaling rate, and design complexity. For example, Intel includes 2 bits per cache entry to replace defective bits [26].

1 In this dissertation, we assume 64-byte cache entries without any loss of generality; all techniques described here could be easily applied to other cache entry sizes.

A different approach is to provide separate voltages for logic and memory [166], but separate voltage domains complicate chip design [92, 105]. Moreover, scaling will likely increase the differences between voltage domains for logic and memory, which eventually increases the number of voltage domains required, and diminishes the potential benefit of operating chips at low voltages.

2.3.2 *Micro-architectural Solutions*

From the micro-architectural perspective, excessive parametric variability can cause circuit behavior consistent with a hard fault. Several runtime methods have been proposed to mitigate its impact, including redundancy through Error Correcting Codes (ECCs), disabling techniques, and the combination of faulty resources to recreate functional ones.

ECCs are extensively employed to protect designs against soft errors, and they have also been studied in ultra-low voltage contexts to protect against hard errors [6, 39]. To store an ECC, the capacity of the cache must be extended or part of its effective capacity has to be sacrificed. ECCs are usually optimized to minimize their storage requirements, at the cost of more complex logic to detect and correct errors. Thus, correcting more than two errors requires a high latency overhead or encodings with more check bits as described in [39], where half of the cache capacity is dedicated to storing the ECC. To reduce storage overhead, Chen *et al.* propose to store ECCs in the unused fragments that appear as a result of compression [33]. Finally, it is worth noting that the use of ECCs to protect against hard errors will jeopardize resilience to soft errors.

A simple approach to mitigating hard faults is to disable faulty entries[2] (cache entries with faulty bits, which cannot store a complete cache block) [142]. This technique, called *block disabling*, is already implemented in modern processors to protect against hard faults [26, 44]. Block disabling has been studied at ultra-low voltages because of its easy implementation and low overhead (1 bit per entry sufficing). However, the large number of faults at low operating voltages implies a large percentage reduction in cache capacity and associativity. For instance, for C2, although defective bits represent around 0.5% of the total, only 9.9% of the capacity is available (Table 2.1). Lee *et al.* examine performance degradation by disabling cache lines, sets, ways, ports, or the complete cache in a single processor environment [94]. To compensate for the associative loss, Ladas *et al.* propose the implementation of a victim cache in combination with block disabling [93].

Ghasemi *et al.* propose the use of heterogeneous cell sizes, so when operating at low voltages, ways or sets of small SRAM cells are deactivated if they start to fail [56]. Khan *et al.* propose a mixed-cell memory design, where a small portion of the cache is implemented with robust cells, which store dirty cache blocks, and the remainder with non-robust cells [81]. They modify the replacement policy to guide the allocation of blocks based on the type of request (load or store). Zhou *et al.* combine spare cells, heterogeneous cell sizes, and ECCs into a hybrid design to improve on the effectiveness obtained by any of the techniques applied alone [170].

2 In this dissertation we make the distinction between cache block and cache entry: block is the transfer unit, the content *per se*; entry is the physical set of cells that store a block.

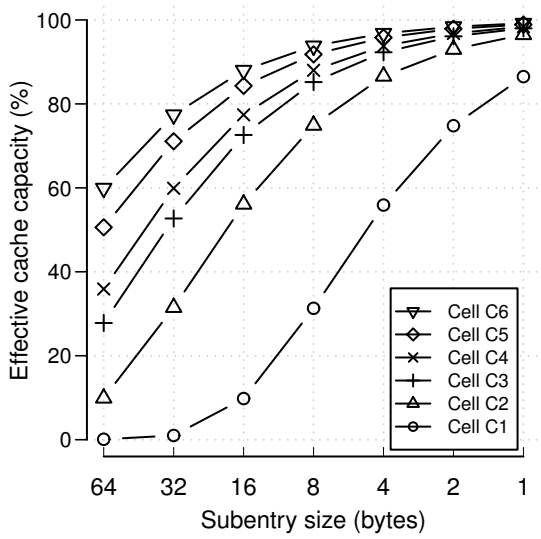


Figure 2.3: Potential cache capacity that can be used by tracking faults at finer granularities for different SRAM cell sizes.

The granularity of the disabled storage might be finer, but at the cost of a larger overhead. Cache entries can be divided into subentries of a given size. A defective cell implies disabling just the subentry which it belongs to, rather than the whole entry. Figure 2.3 shows for the six SRAM cells from Zhou's study, the potential cache capacity that can be used by tracking faults at finer granularities (different subentry sizes). For example, a cache implemented with cells C3-C6 could potentially use more than 80% of the cache capacity by disabling 8-byte subentries.

Previous proposals take advantage of this observation. Word disabling tracks defects at word-level granularity, and then combines two consecutive cache entries into a single fault-free entry, halving both associativity and capacity [157]. Abella *et al.* bypass faulty subentries rather than disabling full cache lines, but this technique is suitable only for the first-level cache, where accesses are word wide [1]. Palframan *et al.* follow a similar approach, patching faulty words from other microprocessor structures, such as the store queue or the miss status holding register [118].

More complex schemes couple faulty cache entries using a remapping mechanism [12, 17, 88, 104]. They rely on the execution of a complex algorithm to group collision-free cache entries from the same or different cache banks, and on additional structures to store the mapping strategy. For example, Archipelago divides the cache into autonomous islands, where one entry is sacrificed to allocate data portions of other entries; they use a configuration algorithm (based on minimum clique covering) to create the islands [12]. The remapping mechanism adds a level of indirection to the cache access (increasing its latency), and the combination of cache entries to recreate a cache block adds complexity.

All of these combination or remapping strategies have a major inconvenience: to reconstruct cache blocks several cache accesses are needed, increasing the energy consumption and/or the latency per block access. Unlike the aforementioned proposals, an ideal fault-tolerant mechanism would not compromise cache capacity, associativity, or latency.

2.3.2.1 *Cache Management Techniques*

Our first main contribution relies on a smart insertion and replacement policy to handle faulty and non-faulty cache entries. In the context of ultra-low voltages, Keramidas *et al.* use a PC-indexed spatial predictor to orchestrate the replacement decisions among fully and partially usable entries in first-level caches [80]. We base our allocation predictions in the reuse patterns, which simplifies the hardware, and we do not consider the use of partially faulty entries.

Regarding the implementation of our techniques, it is worth referring to the work of Jaleel *et al.* [68]. In inclusive hierarchies, the private caches filter the temporal locality and hot blocks (i.e., blocks being actively used by the processor) are degraded in the replacement stack of the LLC to be eventually evicted. The authors address this problem by protecting blocks present in the private caches and preventing their replacement in the LLC through several techniques, including: sending hints to the LLC, identifying temporal locality via early invalidation, or querying the private caches about the presence of blocks. We also protect private copies on all our replacement policies using the coherence information and assuming non-silent evictions of clean blocks.

Albericio *et al.* base the replacement decisions on the block reuse locality [9]. They propose the *Not-Recently Reused* (NRR) algorithm, which protects blocks present in the private caches and blocks that have shown reuse in the LLC. Their simple yet efficient implementation improves the performance of more complex techniques such as RRIP [69]. We build our cache management policy on top of this algorithm.

2.3.2.2 *Compression Techniques*

Our second main contribution relies on compressing cache blocks to fit into the still available LLC capacity. This approach avoids complex cache block remapping, and the combination of faulty cache entries to recreate fully functional ones.

Several compression mechanisms have been proposed in the literature to increase the effective storage capacity of all the on-chip cache levels, potentially reducing misses and improving performance [8, 34, 51, 87, 119, 135, 162]. In general, these proposals seek to maximize the compression ratio either to store more than one block in each cache entry, or to reduce the energy consumption of each LLC access. However, ultra-low voltage operation sets different requirements for the compression scheme: most of the blocks need to be compressed to fit into faulty entries, but the compression ratio is relatively small. In an ideal case, where all blocks could be compressed to fit into the available space, neither capacity nor associativity would decrease. Similar requirements may be observed in the context of Non-Volatile Memories (NVM) to enlarge their durability [57, 160].

We combine the compression mechanism with a smart allocation/replacement policy, which assigns cache blocks to faulty entries based on their compressibility. Two recent studies have indicated the importance of taking compression information into account for a replacement policy tailored to on-chip caches implementing compression [14, 120]. Both studies are based on the fact that several blocks are allocated to the same cache entry and, therefore, prioritizing the replacement of large blocks might allow the allocation of several smaller blocks to the same space. In ultra-low voltage

operation, the compression ratio must be taken into account in the replacement decision as well, but in order to even out the pressure on the entries of a cache set.

3 EXPERIMENTAL FRAMEWORK

This chapter presents the experimental framework used during Part II of the dissertation, including the modeled system, the experimental set-up, and the metrics to quantify the impact of the proposed techniques. Specific details might change for the evaluation of some of the ideas of this dissertation. In that case, those details are properly explained when required.

3.1 OVERVIEW OF THE MODELED SYSTEM

Our baseline system consists on a tiled chip multiprocessor (CMP), with an inclusive two-level cache hierarchy, where the second level or last-level cache (LLC) is shared and distributed among the processor cores. Tiles are interconnected by means of a two dimensional mesh, in a similar way as Phytium's Mars [76]. Each tile has a processor core with a private first level cache (L1) split in instructions and data, and a bank of the shared LLC, both connected to the router (Figure 3.1). The cache write policies are write-back and write-allocate. LLC banks are interleaved by cache line address, and inclusion is enforced at the LLC; i.e., when a cache line is inserted in L1, it is also inserted in the LLC, and when a line is evicted from the LLC, it is replaced from the private levels, too. Two memory controllers are distributed on the edges of the chip. Table 3.1 shows the parameters of the baseline processor, memory hierarchy, and interconnection network. During the next chapters, when necessary, we will detail the parameters that differ from the baseline configuration system.

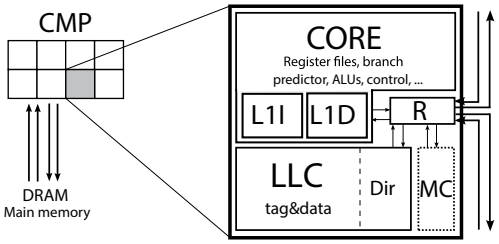


Figure 3.1: Tiled CMP with 8 processors.

At lower voltages near to the threshold voltage, frequency degrades about 5 to 10x [49]. We assume a frequency of 1 GHz at 0.5 V, our target V_{dd}. Note that the DRAM module voltage is not scaled as the rest of the system. Thus, the relative speed of main memory with respect to the chip gets faster as the voltage decreases. This model is consistent with prior work [12, 157].

Our baseline coherence protocol relies on a full-map directory with MESI states (Modified, Exclusive, Shared, and Invalid). We use explicit eviction notification of both shared and exclusively owned blocks. L1 caches are built with robust SRAM cells and, therefore, they can run at lower voltages without suffering from parameter variations, while LLC banks are built with conventional 6T SRAM cells, so they are sensitive to failures [91].

Table 3.1: Main characteristics of the modeled CMP system.

Cores	8, Ultrasparc III Plus, in-order, 1 instr/cycle, single-threaded 1 GHz at V_{dd} 0.5 V
Coherence Protocol	MESI, directory-based full-map dist. among LLC cache banks
Consistency	Sequential
L1 cache	Private, 64 KB data and inst. caches, 4-way, 64 B block size, LRU 2-cycle hit access time
LLC cache	Shared, 1 bank/tile, 1 MB/bank, 16-way, 64 B block size, pseudo-LRU[a] Inclusive, interleaved by line address 8-cycle hit access time (4-cycle tag access)
Memory	2 memory controllers, distributed on the edges of the chip Double Data Rate (DDR3 1333 MHz), 2 channels, 8 Gb/channel 8 banks, 8 KB page size, open page policy Raw access time 50 cycles[b]
NoC	Mesh, 2 Virt. Networks (VN): requests and replies 16-byte flit size, 2-stage routers, 1-cycle latency hop

[a] Chapter 4 assumes NRU [145], while Chapter 5 assumes LRU; in both cases, the baseline replacement policy protects the private copies of the blocks [68].

[b] When the CMP runs at 1 GHz/0.5 V_{dd}.

3.2 SIMULATION FRAMEWORK

Regarding our experimental set-up, we use *Simics*, a multiprocessor full-system simulator, running the *Solaris 10* operating system [102]. To model a detailed and cycle-accurate on-chip memory hierarchy and interconnection network, we use the *ruby* plugin from GEMS Multifacet's multiprocessor simulator toolset [107]. During this dissertation, we enlarge the ruby module to faithfully model our different proposals. In order to get a detailed performance and energy DDR3 DRAM model, we incorporated DRAMSim2 into our simulation infrastructure [128]. To get timing, area, and energy consumption we use McPAT framework [95] for the on-chip components, and DRAMSim2 for the DRAM module.

3.3 WORKLOADS AND METHODOLOGY

We consider two sets of multiprocessor workloads exhibiting different kinds of parallelism. On the one hand, multiprogrammed workloads are a combination of several sequential programs of different nature and behavior running at the same time in the system; i.e., there are as many independent tasks as processor cores. These workloads are representative of, for example, desktop systems running different applications or servers running different user requests. We use the 29 programs from the SPEC CPU 2006 benchmark suite [61]. SPEC CPU 2006 is an industry-standardized, CPU-intensive benchmark suite, stressing a system's processor, memory subsystem, and compiler. It contains two suites that focus on two different types of compute intensive performance: the CINT2006 (integer) suite measures compute-intensive integer performance, and the CFP2006 (floating point) suite measures compute-intensive floating point performance.

On the other hand, parallel workloads are shared-memory applications; i.e., one task is divided along the available processor cores. Parallel applications are mainstream in today's CMPs. We use applications from the PARSEC benchmark suite [19]. The PARSEC suite focuses on emerging workloads and was designed to contain a diverse selection of applications that is representative of next-generation shared-memory programs for CMPs.

Next, we present the selected workloads and methodology in depth.

3.3.1 *Multiprogrammed Workloads*

To select the appropriate number of workloads, we use a set of 100 workloads built as random combinations of the 29 SPEC CPU 2006 programs, with no special distinction between integer and floating point. The applications appear from 16 to 35 times, the average number of occurrences is 27.6, with a standard deviation of 4.5.

In order to locate the end of the initialization phase, all the SPEC binaries with the reference inputs were run until completion with hardware counters enabled. To ensure that no application was in its initialization phase, each multiprogrammed mixed was run for as many instructions as the longest initialization phase, and a checkpoint was created at that point. From that point, we run cycle-accurate simulations including 300 million cycles warm-up of the memory hierarchy and 700 million cycles of statistics collection.

To reduce the experimental workload, highly loaded with the Monte Carlo simulations discussed in the following sections, we compute the minimum number of mixes needed to obtain statistically representative results. Following statistical sampling [153], and assuming a confidence interval of $(1-\alpha) = 0.95$ and error tolerance $\varepsilon = 5\%$, we can compute the minimum number of mixes n as:

$$n \geqslant \left(\frac{z * V_x}{\varepsilon}\right)^2 \tag{3.1}$$

where z is the $100[1-\frac{\alpha}{2}]$ percentile of the standard normal distribution, and V_x is the coefficient of variation of the population. As V_x of the entire population is not available, we use sample $\hat{V}_x$. Our initial sample size is $n = 100$ (mixes). We obtain the coefficient of variation $\hat{V}_x = 0.10901$, based on the total number of instructions executed for each mix. As $z = 1.96$, the minimum number of mixes is $n \geqslant 18.26$.

Thus, we select 20 workloads built as random combinations of the SPEC CPU 2006 programs. Each application appears on average 5.5 with a standard deviation of 2.5.

To evaluate the behavior of multiprogrammed workloads for the different proposals, we use LLC Misses Per Kilo Instruction (MPKI) and speedup. Speedup for a given mix is computed as in Equation 3.2, where IPC_i^A is the number of executed instructions per cycle of program i when it runs in system A, and IPC_i^{Base} is the number of executed instructions per cycle when it runs in the $Base$ system (baseline). Note that as we always run a

fixed number of cycles, this is equivalent to using the number of instructions executed.

$$\text{Speedup} = \sqrt[n]{\prod_{i=1}^{n} \frac{IPC_i^A}{IPC_i^{Base}}} \tag{3.2}$$

As the different mixes are run for equal parts on the baseline system, as well as on the different configurations, the arithmetic mean of individual speedups is used when summarizing results for all the workloads [72].

3.3.2 *Parallel Workloads*

We use a selection of shared-memory parallel applications from PARSEC with a significant memory footprint when using the *sim-large* input (i.e., $MPKI_{LLC} \geqslant 1$ in the baseline system, Table 3.2).

Table 3.2: LLC MPKI for parallel workloads (baseline system).

	canneal	ferret	streamcluster	vips
$MPKI_{LLC}$	4.26	1.59	1.00	1.19

Parallel applications should run until completion in order to compare the performance of different design alternatives. However, in our applications, no OS activity appeared when parallel applications were run, and the ratio of load of work among the different threads was practically constant between simulations. Thus, following the same approach as [10], we use sampling, and the same way than for multiprogrammed workloads, we run 300 million cycles of warm-up of memory structures once the parallel phase has started, and then collect statistics for 700 million execution cycles.

The speedup is computed as in Eq. 3.3, where I_i^A is the number of executed instructions of thread i when it runs in system A, and I_i^{Base} is the number of executed instructions when it runs in the $Base$ system.

$$\text{Speedup} = \frac{\sum_{i=1}^{n} I_i^A}{\sum_{i=1}^{n} I_i^{Base}} \tag{3.3}$$

3.4 SRAM FAILURE MODEL

Random intra-die variations are modeled as random variables and used as inputs to determine the failure probability or P_{fail} of a given SRAM cell. We can assume that failures are randomly distributed, but uniform within a given chip [2, 36]. Thus, we create fault maps with different distribution of failures for each P_{fail} point. We follow Zhou *et al.* study, described in Chapter 2, and modeled five configurations (C2-C6) with 6T SRAM cells of different sizes (i.e., P_{fail} points) [170]. To create the fault maps for each configuration, we compute the faultiness of each memory cell randomly and independently of other cells, obtaining configurations with similar number of faults, but different locations, and run Monte Carlo simulations as follows.

For each design point, we run N configurations (fault maps) and compute the sample mean ($\bar{X}$). The accuracy of the sample mean will improve as we increase the number of samples (number of fault maps or N), but if we set the number of samples too large, we will do unnecessary computation without gaining meaningful accuracy.

To obtain representative results, we follow the methodology summarized below. Our target error is $\varepsilon = 5\%$ with a confidence level $(1-\alpha) = 0.95$.

Step 1. We set an initial value of samples N and collect results to compute the sample mean $\bar{X}$ and the sample variance of a selected metric S^2 according to equation 3.4[1].

Step 2. Given the confidence level $(1-\alpha) = 0.95$ and error tolerance $\varepsilon = 5\%$, we check if equation 3.5a is satisfied[2]. If so, we claim that the stopping criteria has been achieved and stop the simulation. Otherwise, we increase the number of samples N′ according to equation 3.5b, and repeat from Step 1.

$$\bar{X} = \frac{x_0 + x_1 + \cdots + x_{N-1}}{N} \tag{3.4a}$$

$$S^2 = \frac{\sum_{i=0}^{N-1} (x_i - \bar{X})^2}{N-1} \tag{3.4b}$$

$$\left(\frac{t_{[N-1,\frac{\alpha}{2}]} \times S}{N \times \bar{X}} \right)^2 \leqslant \varepsilon \tag{3.5a}$$

$$N' \geqslant \left(\frac{t_{[N-1,\frac{\alpha}{2}]} \times S}{\varepsilon \times \bar{X}} \right)^2 \tag{3.5b}$$

Note that we could use different metrics in Step 1 to compute the sample mean and sample variance to check if we have reached our target error. Nevertheless, as we are interested in comparing throughput by means of speedups, we also consider speedup to compute the error.

For the sake of clarity, we avoid plotting error bars in the different results graphs of this dissertation, providing that all the results are within the error and confidence levels here described.

1 Note that we use $(N-1)$ to compute the unbiased S^2 (Bessel's correction).

2 $t_{[N-1,\frac{\alpha}{2}]}$ is obtained from the t-distribution function with $N-1$ degrees of freedom.

4

EXPLOITING REDUNDANCY AT CACHE LEVEL

This chapter presents a cache management policy to enable efficient last-level cache (LLC) operation at low or near-threshold voltages. We base our proposal on block disabling, a micro-architectural technique that tolerates faults by deactivating faulty cache entries. Block disabling's main drawback is that, given the random component of SRAM cell faults, both cache associativity and capacity rapidly degrade as the number of defective cells increases. We take advantage of the on-chip coherence and replacement mechanisms to enhance block disabling performance, without any additional cost. For that, we exploit the natural redundancy of multi-level inclusive cache hierarchies and extend the LLC with a fault-aware cache management policy, which maps blocks with a higher probability of being used to operative cache entries. Our evaluation shows that this fault-aware management can reduce MPKI by up to 37.3% for multiprogrammed workloads, and by 54.2% for parallel ones, with respect to block disabling. This translates into performance enhancements of up to 13% and 34.6% for multiprogrammed and parallel workloads, respectively.

4.1 INTRODUCTION

Modern chip multi-processors (CMPs) include several cache levels to reduce the gap between the speed of the processor and the access latency to the main memory. On-chip caches are usually built with SRAM cells, which are sensitive to process variations, specially when operating at low or near-threshold voltages [18]. First-level caches are usually private, occupy little area, and their access time often determines the processor cycle time. Commercial processors, such as the Intel Nehalem family, use robust 8T SRAM cells to build reliable first-level caches, since it represents an affordable overhead [91]. On the contrary, LLCs are usually shared and have larger size and associativity, accounting for much of the die area [26]. Thus, minimum-geometry 6T cells are preferred to achieve higher density, with the consequent higher sensitivity to failures.

Block disabling is a simple micro-architectural technique that disables a cache entry when a defective bit is found [142]. Its main disadvantage is that due to the random distribution of defective cells at low voltages, the capacity of the cache is rapidly compromised. In this chapter, we propose a new approach to mitigate the impact of SRAM failures in LLCs based on block disabling, relying on the underlying structures already present in CMPs. We identify a natural source of on-chip data redundancy that arises because of the replication of blocks in inclusive multi-level cache hierarchies and exploit this redundancy through a smart fault-aware cache management policy.

Our fault-aware cache management policy is able to decrease the LLC MPKI up to 37.3%, with respect to block disabling, which translates into speedup improvements between 2% and 13% for multiprogrammed workloads. For parallel workloads, the MPKI improvement ranges between 5% and 54.2%, with respect to block disabling, for the different SRAM cells considered, improving performance up to 34.6%.

The rest of this chapter is organized as follows. Section 4.2 comments on block disabling and its impact on large cache structures. In Section 4.3, we present how to take advantage of the coherence infrastructure to operate at low V_{dd}. Section 4.4 introduces a fault-aware cache management policy for LLC operating at low voltages. Section 4.5 presents our evaluation. Section 4.6 discusses the system impact, and Section 4.7 concludes.

4.2 IMPACT OF BLOCK DISABLING ON LARGE SHARED CACHES AT LOW VOLTAGES

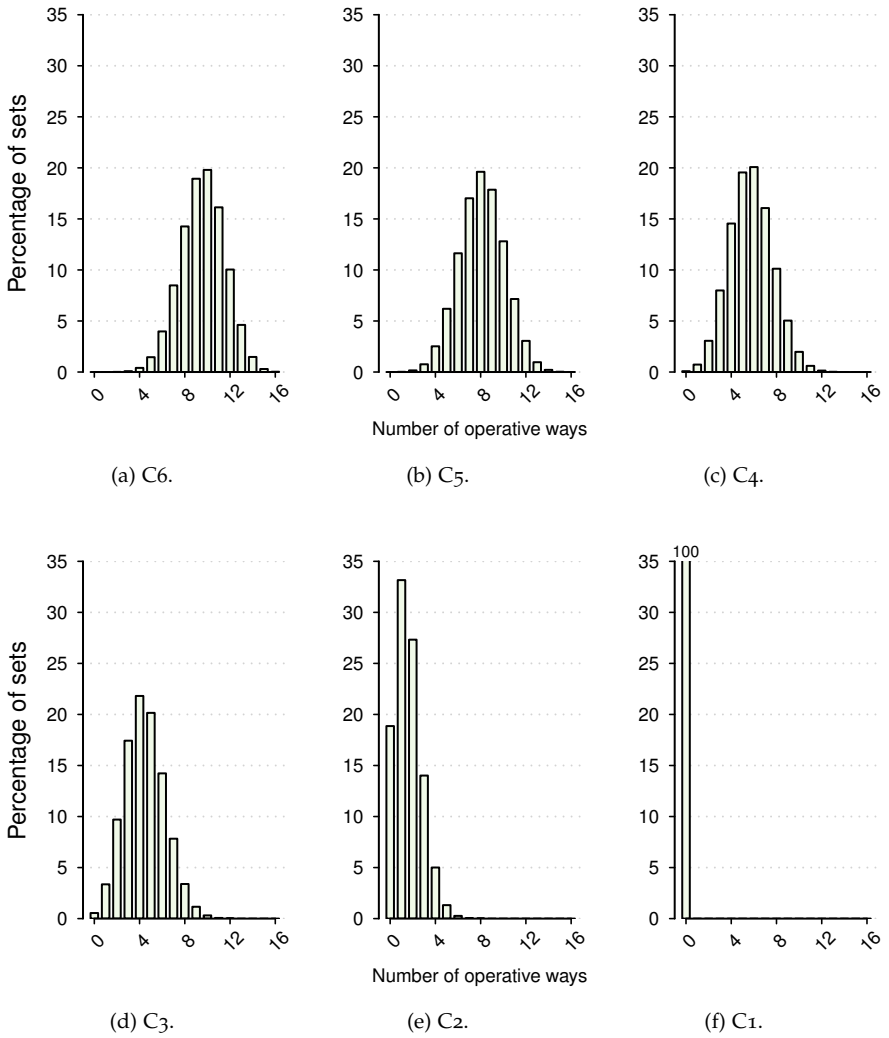
A simple approach to tolerate hard faults is disabling faulty resources. Block disabling (BD) deactivates resources at block (cache entry) granularity: when a fault is detected at a given cache entry, that entry is marked as defective and it can no longer store a cache block [142]. This technique is implemented in modern processors to bear hard faults [26].

BD has also been studied for operation at low voltages because of its easy implementation and low overhead [93]. From the implementation perspective, only one bit per entry suffices to mark the entry as faulty. Its main drawback is that the amount of capacity dramatically falls when the probability of failure increases, as shown in Table 2.1. Although the total count of faulty cells in the cache is less than 1%, the effective cache capacity is strongly affected because of the random distribution of faulty cells. BD results in caches with variable associativity per set, determined by the number and distribution of faults in the cache.

The interaction between BD and the cache organization of the system also plays an important role. Modern commercial processors, such as the Intel Core™ i7, implement inclusive hierarchies to ease the coherence management. Inclusive hierarchies require that all the blocks cached in the private levels are stored in the shared LLC, too. The directory information is embedded in the LLC; i.e., each block is augmented with sharing state and a bit vector to represent the current sharers. To enforce inclusion, when a block is evicted from the LLC, explicit back-invalidations are required to remove the copies of the private cache blocks, if present (inclusion victims) [16].

Inclusive hierarchies perform poorly when the aggregated size of the private caches is similar to the size of the LLC [68]. BD exacerbates the problem because of the substantial associativity and capacity degradation in the LLC. Figure 4.1 shows the available associativity of a 16-way set associative cache bank with 64-byte blocks, when built with cells C6-C1 (Table 2.1) operating a 0.5 V. The number of faulty ways per set follows a binomial distribution $B(n, p)$, where n is the associativity, and p denotes the failure probability of a cache entry. Figure 4.1 shows how the associativity degrades as more faulty cells appear on the cache structure. On average, 50% of the ways are faulty if the cache is built with C5 cells, and this percentage rises to 85% when using C2 cells; for C1, 100% of the cache entries are faulty at 0.5 V and, therefore, we will not consider it in our study. The associativity loss directly translates into a significant increase in the number of inclusion victims. For instance, the number of invalidations in a cache built with C3 cells is 10x larger than in a cache implemented with fault-free SRAM cells.

This last evidence suggests that inclusive hierarchies are not well suited for systems that implement BD in presence of a significant number of faults. However, from the coherence management perspective, only directory inclusion is required: blocks present in the private levels have to be tracked only



(a) C6. (b) C5. (c) C4.

(d) C3. (e) C2. (f) C1.

Figure 4.1: Available associativity of a 16-way set associative block disabling cache (64-byte block) made up of cells C6-C1 operating at 0.5 V.

in the shared level tag array, without the need of having a replica in the data array [16]. This observation is the basis for the techniques we propose in this chapter.

Note on Figure 4.1 that, when using cell types C3 and C2, 0.6% and 18.9% of the sets have no operative ways, respectively. To be able to offer a complete comparison with BD, we assume that at least one of the ways in each set is non-faulty, although the techniques we present in this chapter do not have this requirement, and the LLC could operate even when all the ways of a set are faulty.

4.3 EXPLOITING INCLUSIVE HIERARCHIES TO ENABLE LOW VOLTAGE OPERATION: BDOT

The BD scheme simply assumes one extra bit per entry to identify faulty cache entries in the data array (one or more faulty cells). Faulty data entries

are excluded from tag search and replacement, involving a net reduction in associativity, and the consequent increase in inclusion victims. However, from the coherence management perspective, tracking blocks in the shared level tag array suffices to enforce directory inclusion. This is the basis for the first technique that we call *Block Disabling with Operational Tags* (BDOT).

Assuming a two level inclusive hierarchy, to keep directory inclusion, we turn on the tags of faulty entries in the LLC, including them in the regular actions of search and replacement. The tag of a faulty entry, if valid, tracks a cache block that might be present in the private caches, but that cannot be stored in the shared cache. Enabling the tags of the faulty entries restores the associativity of the shared cache as seen by the first-level private caches, removing the increase in the number of inclusion victims caused by the associativity loss.

In this situation, two kinds of LLC entries have to be distinguished: tag-only (T), where the associated data entry is faulty and only the tag is stored, and tag-data (D), where the associated data entry is non-faulty and both tag and data are stored. From the implementation perspective, still one resilient bit suffices to indicate whether the entry is faulty or not. The coherence protocol needs to be adapted to this new situation, where a T entry only stores the block tag and directory state. Whenever a request to a block stored in a T entry arrives to the LLC bank, the request needs to be sent to the next level (in this case, off-chip) to recover the block, and the same occurs with dirty blocks, which need to be written back to memory after being evicted from a private cache.

To fully exploit this scheme, no failures should occur in the cells of the tag array. This can be accomplished, for example, by using robust cells (e.g., increasing the number of transistors per cell) or increasing the strength of the ECC. Tags occupy very little area in comparison to the data array (around 6% for our configuration, see Table 3.1), and increasing the cell size by 33% (assuming 8T SRAM cells [29]) will end up increasing the total area of a cache bank by 2%. Since using sophisticated ECC could increase the access latency to the tag array, and using resilient tag cells involves little overhead, we will resort to the latter. This approach is also consistent with prior work [12, 157]. Besides, many of today's CPUs use different cell types for tag and data arrays [81].

4.3.1 *BDOT Limitations*

BDOT, as described above, has two potential limitations, both related to the allocation of blocks to faulty entries.

First, BDOT always forwards requests to blocks allocated to faulty entries to the off-chip memory. However, a block allocated to a faulty entry might be present on-chip, if it is being used by a private cache (L1). This situation is common in parallel workloads, which share data and instructions. In this case, the directory information can be used to orchestrate the cooperation among L1 caches. An L1 request to a shared block mapped to a T entry is forwarded to one of the sharers of the block (another L1). That L1 will serve the block through a cache-to-cache transfer.

Cache-to-cache transfers are already implemented in the baseline coherence protocol for exclusively owned blocks. Thus, no additional hardware is required and a slight modification of the directory protocol suffices to trig-

ger a shared block transfer. So from now on, we assume that BDOT includes this feature.

The second limitation comes from allocating blocks to LLC entries without taking into account their T or D nature. Unfortunately, this blind allocation can leave heavily reused blocks stuck to faulty entries. Indeed, if a particular block of the LLC is required repeatedly from an L1 cache (i.e., the block shows reuse), any replacement algorithm will tend to protect it, reducing its eviction chances. Thus, if a block with reuse is initially allocated to a T entry, unless replicated in other cores, all L1 cache misses will be forwarded off-chip by the LLC.

In the next section, we introduce a specific allocation and reallocation policy for BDOT caches that differentiates between T and D entries.

4.4 FAULT-AWARE CACHE MANAGEMENT POLICY FOR BDOT CACHES

Conventional cache management policies assume that every cache entry can store a block, while BDOT breaks this assumption: each set in an N-way set associative cache contains T entries that store only tag, and D entries that store tag and data. With the main goal in mind of improving the overall LLC performance under BDOT, this section introduces a fault-aware cache management policy that takes into account the distinct nature of T and D entries, and the reuse pattern of the reference stream. In particular, we propose to meet the following two goals:

1. Try to allocate blocks that are most likely to be used in the future to D entries.
2. Try to maximize the on-chip contents by giving more priority (higher chances to be allocated to D entries) to blocks that are not present in private cache levels.

Prior work has shown that *reuse* is a very effective predictor of the usefulness of a given block in the LLC [9, 31]. Reuse locality can be described as follows: lines accessed at least twice tend to be reused many times in the near future, and recently reused lines are more useful than those reused earlier [9]. Thus, regarding our first goal, we exploit reuse locality to predict which blocks should be allocated to D entries. With respect to our second goal, a request to a block allocated to a T entry and present in L1 can be serviced through a cache-to-cache transaction, whilst if the block is not present in L1, the request will be always forwarded to the off-chip memory, penalizing the time and energy of the access. Therefore, it is preferable to dedicate D entries to blocks not available on the L1 caches.

These goals may be added to any management policy. In this work, we will build on top of a state-of-the-art reuse-based replacement algorithm: Not-Recently Reused (NRR) [9]. Next, we describe the baseline replacement in some depth and then, we make it aware of the existence of faulty entries.

4.4.1 *Baseline NRR Replacement Algorithm*

The NRR algorithm requires four states per LLC block, as depicted in Figure 4.2. When a block not present in the LLC is requested by the processor (1^{st} *use: L1 request*), it is stored in the L1 and the LLC (to enforce inclusion), being its state in LLC `NR-C` (Non-Reused, Cached). When the block is

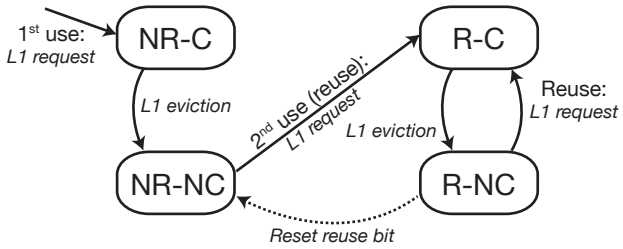


Figure 4.2: Reuse and inclusion states for a block in LLC. NR, R, C, and NC represent: Non-Reused, Reused, Cached (in L1), and Non-Cached (in L1), respectively. Replacement and coherence transitions are not shown.

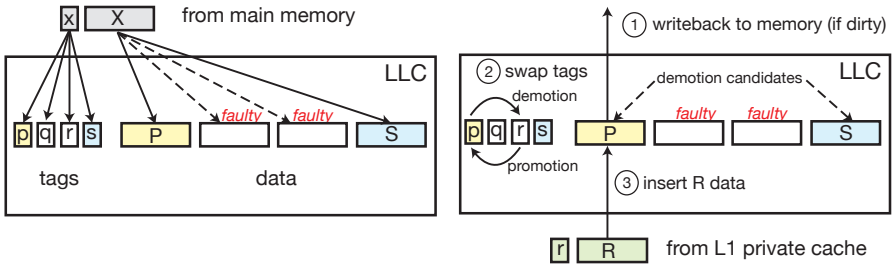
evicted from the private cache (*L1 eviction*), its LLC state changes to `NR-NC` (Non-Reused, Non-Cached). On a new request (*2nd use: L1 request*), a copy of the block is stored again in L1, and its LLC state is `R-C` (Reused, Cached). At this point, the block has shown reuse in the LLC and, very likely, it will be reused many times in the near future. Finally, when the block is evicted again from the L1, the state becomes `R-NC` (Reused, Non-Cached). Subsequent requests and evictions switch between the `R-NC` and `R-C` states.

Having LLC blocks classified this way, the replacement policy can exploit L1 temporal locality and LLC reuse. In an inclusive hierarchy, the replacement of a block in the LLC forces the invalidation of its copies in the private caches, if any, which usually implies a performance degradation, as long as blocks in L1 are being actively used [68]. Therefore, the highest priority (protection) is given to blocks stored in private caches. As a secondary objective, the highest priority is given to blocks that have shown reuse in the LLC. Hence, NRR selects victims in the following order: `NR-NC`, `R-NC`, `NR-C`, `R-C`. Reuse recency is taken into account by resetting the reuse bit, when all the non-cached blocks are marked as reused (transition from `R-NC` to `NR-NC`). This way, more recently reused blocks become more protected.

The implementation of NRR only requires one *reuse bit* per block. The protection of private copies can be implemented in different ways [68], but a simple solution uses the presence bit-vector of the coherence directory, assuming non-silent tag evictions of clean blocks.

4.4.2 *Reused-based and Fault-aware Management for BDOT Caches*

Seeking to guarantee that valuable blocks remain in the LLC, we devise a fault-aware management policy by distinguishing between T and D entries. One option is to *promote* blocks by reallocating them from T to D entries, if needed, to improve the overall cache performance. The design choices include *where* the promoted data comes from and *which* victim is chosen as a target of the consequent *demotion*. At the same time, we want to continue exploiting reuse in the simple and efficient way offered by an NRR-like replacement algorithm, which is unaware of faulty entries. Thus, our goal is to design a comprehensive cache management policy, merging reuse exploitation and faulty entries management. Below, we elaborate on the two key mechanisms, namely block insertion/replacement and block promotion/demotion.



(a) Insertion from main memory after LLC miss.

(b) Insertion from L1 after L1 eviction (promotion and demotion).

Figure 4.3: Insertion and promotion actions for a fault-aware cache management policy: example with a 4-way cache set with two faulty cache entries. Lowercase and capital letters indicate tag and data, respectively.

4.4.2.1 *Insertion and Replacement of Blocks*

On a first insertion (LLC miss), an incoming block has not shown reuse, so allocating it to a T entry seems a reasonable idea. Figure 4.3a shows an example of a cache block to be inserted in a 4-way cache set with two T entries (those storing `q` and `r` tags) and two D entries (those storing `p` and `s` tags and the corresponding `P` and `S` data). A victim is selected among the blocks allocated to T entries. The baseline replacement dictates which one among those blocks (`Q` and `R`) is selected for replacement. This is equivalent to predict that the incoming block `X` is not going to be reused. In case the reuse pattern of the block is mispredicted, block `X` should be reallocated to a D entry, to reduce its access time and transfer energy in future L1 misses. This would be solved using the promotion mechanism we detail in the next subsection.

Dealing with first insertions this way is very simple but has a clear inconvenience, related to the distribution of T and D entries, with respect to the percentage of reused and non-reused blocks. For example, if the number of T entries is small, the insertion policy would incur in a lot of pressure on the scarce T entries. Blocks would be unavoidably forced to leave the LLC before having enough time to show a reuse pattern, even though there are many available D entries. In an extreme case, when all the entries in a set are D type, this cache management policy could not be implemented. Solving this issue is not easy. We devised some adaptive mechanism in which some D entries are used as T. However, finding an optimal number of T entries is difficult, and it highly depends on the workload. After carrying out some experiments (not shown in Section 4.5, for the sake of brevity), the performance returns were limited given the required complexity.

With the right promotion mechanism to reallocate blocks from T to D entries and vice versa, we realized that the baseline NRR replacement itself suffices to achieve our initial goals. This is supported by the protection of reused blocks that guides the baseline replacement. Reused blocks are likely to be reallocated to D entries once they have shown reuse, so protecting them in the replacement algorithm (lower chances of eviction) naturally selects blocks in T entries as replacement victims. Hence, the initial insertion does not necessarily have to consider the nature of the entry, and our

implementation only relies on the baseline replacement policy to select the victim block.

4.4.2.2 *Promotion and Demotion of Blocks*

A blind allocation of blocks to cache entries may result in valuable blocks (i.e., those with reuse) being initially allocated to T entries, and vice versa. However, this undesirable situation can be tracked on the fly through the reuse footprint, and reverted by swapping the T entry with a D entry: when a block allocated to a T entry shows reuse, we will *promote* it to a D entry. Promotion involves a complementary *demotion* of the block stored in the selected D entry.

To select which block is demoted, we also rely on reuse and L1 presence information. Reused blocks should be kept in the LLC, but contrarily to the baseline replacement, *block demotion does not involve an LLC tag eviction*. Furthermore, if the block is present in L1, losing the content in the LLC is not critical, because there is at least one on-chip copy of the block, which can be supplied by a cache-to-cache transaction. Thus, to maximize the on-chip contents, the demotion algorithm will select the victim block among those present in L1. Among the blocks in L1, non-reused ones should have priority to be demoted.

Note that the promotion of a block can be performed in two different moments: at reuse detection (i.e., second L1 request to a block stored in a T entry) or after the second eviction from L1 (i.e., eviction after reuse). Performing the promotion after the second request from L1 duplicates the content, as a copy of the block is stored in a private cache, too, whilst performing the promotion after the L1 eviction meets the goal of maximizing the on-chip content. Thus, we resort to the latter and trigger promotions only after L1 evictions, being necessary non-silent block data evictions.

The promotion/demotion is illustrated in Figure 4.3b. When block `R`, which is stored in a T entry, is evicted from the L1 cache and selected for promotion (i.e., its reuse bit is set), we select a victim among the demotion candidates (`P` and `S` in Figure 4.3b). Once the victim is selected (`P` in our example), we swap the cache contents in three steps: ① discard the data entry `P`, writing back its content to memory, if dirty; ② swap `p` and `r` tags; and ③ copy the data (`R`) in the available D entry, which was occupied by the demoted block (`P`).

4.4.2.3 *Summary and Implementation*

Figure 4.4 illustrates the implementation of the aforementioned ideas. The states of the baseline replacement shown in Figure 4.2 are now *superstates* split into T and D states. The initial allocation of blocks (*1st use: L1 request* in Figure 4.4) does not take into account the nature of the entry, and it solely depends on the victim selection arising from reuse and L1 presence; i.e., it only depends on the baseline replacement algorithm. After insertion, blocks will move along `NR-C`, `NR-NC`, `R-C`, and `R-NC` superstates as they would do in a cache without considering faulty entries.

To guarantee that highly valuable blocks—those showing reuse—remain in the LLC, the policy reallocates them from T to D entries when they are evicted from the L1 and reside in a faulty LLC entry: state `R-C-T`. After *L1 eviction*, blocks in `R-C-T` trigger a promotion action, which results into the

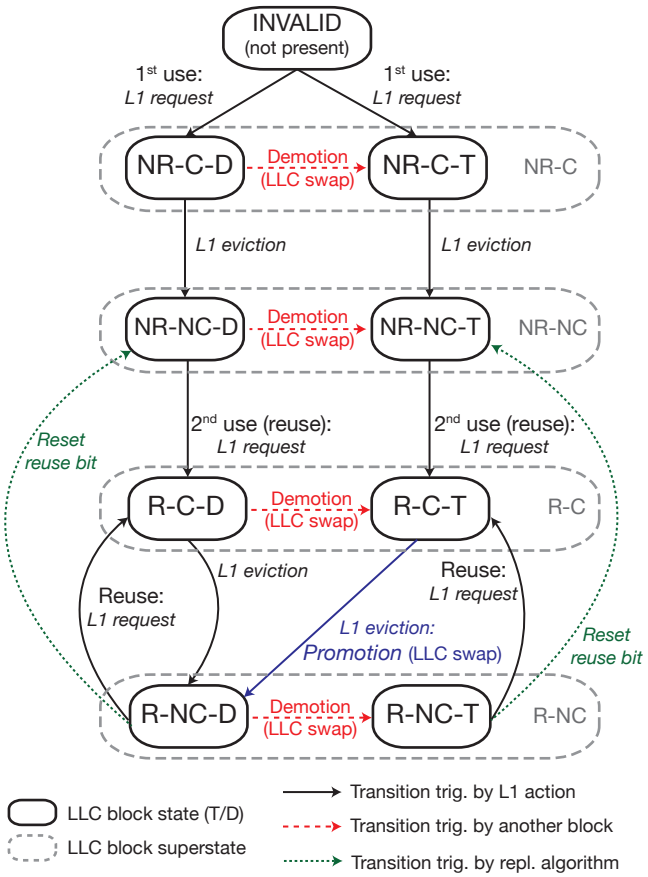


Figure 4.4: Reuse and inclusion states for a block in LLC with BDOT.

transition to `R-NC-D` state and reallocation to a D entry, with the consequent demotion of another block within the set to a T entry. A block being demoted can be in any of the superstates; according to the victim selection algorithm, we demote first blocks that are present in the private levels, in order to maximize the available content on the on-chip hierarchy. As a secondary objective, the policy attempts to first demote blocks without reuse. In particular, it selects blocks in the following order: `NR-C-D`, `R-C-D`, `NR-NC-D`, and `R-NC-D`.

This reuse-based, fault-aware policy adds no extra storage overhead to the baseline reuse-based replacement, as only the bit indicating reuse and the presence bit vector are needed to orchestrate the replacement and promotion decisions. Besides, swapping blocks only requires some extra control logic to perform the following actions: first, the logic reads the demoted victim and inserts the promoted block, same as for a conventional block insertion, and, then, it writes back the tag of the demoted block. Promoting blocks after L1 eviction implies non-silent eviction of data blocks. This overhead does not affect latency, as L1 replacements are not in the critical path, and has a negligible impact on the energy consumption.

The fault-aware cache management technique here presented could be implemented on top of other replacement algorithms (such as LRU or NRU). We decided to rely on NRR because of its simple, yet efficient implementation, and because it fits the general principles behind our ideas. Finally, and regarding the reallocation from T to D entries and vice versa, other policies

are also possible. For example, instead of relying on the reuse information of the blocks, a future use predictor [159] could be utilized to decide which blocks should be allocated to D entries, or a dead block predictor [82] could be used to indicate which blocks may be demoted to T entries, but these solutions add complexity to the cache logic as well as require more storage overhead.

4.5 EVALUATION

This section evaluates the effectiveness of the proposed BDOT management technique for LLC caches in terms of MPKI. Later on, Section 4.6 analyzes the impact on system performance, area, and energy.

To assess the effectiveness of our proposals, we include several additional configurations. First, as an upper bound in performance, a robust cache built with *unrealistically robust cells* (Robust); i.e., cells that operate at ultra-low voltages with neither failures nor power or area overhead, which corresponds to a perfect unattainable solution. Then, we also include block disabling (BD), as our proposal emerges from it. Finally, we add results for *word disabling* (WD) [157]. Word disabling is a more complex technique that combines consecutive faulty cache entries to recreate fully functional ones, at the cost of reducing the cache capacity. Section 4.5.3 presents a qualitative discussion of other techniques versus our proposals.

In summary, we consider the following configurations:

- *Robust*: reference system; the LLC is built with unrealistically robust cells. All data are presented with respect to this system.
- *BD*: system implementing block disabling, as presented in Section 4.2, with NRU replacement.
- *BDOT-NRU*: system implementing block disabling with operational tags, as presented in Section 4.3, with NRU replacement.
- *BDOT-NRR*: system implementing BDOT with NRR replacement, as presented in Section 4.4.1.
- *BDOT-NRR-FA*: system implementing BDOT with fault-aware NRR replacement, as presented in Section 4.4.2.
- *WD*: system implementing word disabling with NRU replacement [157].

As in the case of NRR, the NRU implementation also includes private copy protection. Our detailed results include multiprogrammed workloads (the 20 SPEC CPU 2006 mixes) and parallel workloads (the 4 selected PARSEC applications), for the five cell types considered (C6, C5, C4, C3, and C2).

4.5.1 *Multiprogrammed Workloads*

Figure 4.5a shows the LLC MPKI results for the multiprogrammed workloads.

BD is a valid solution for a cache with few defective entries, like one built with C6 cells, where the average MPKI penalization is 23.9%. However, this penalization increases rapidly with the number of faulty entries, reaching 136% for C2. Using the tags of the defective LLC entries to keep the coherence state of blocks stored in L1 allows BDOT-NRU to reduce the MPKI

increase with respect to BD for C2, but it does not offer any advantage (the MPKI increases) for the rest of the cells.

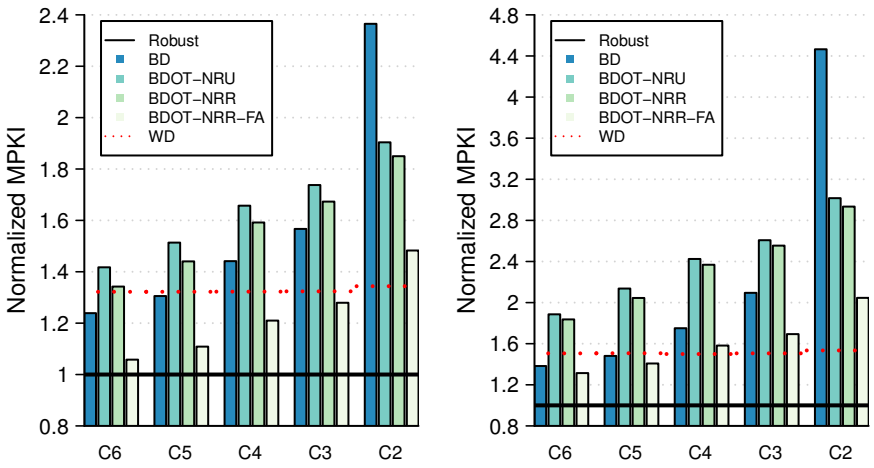
To differentiate and quantify the benefit of a reuse-based replacement and our fault-aware cache management policy, we first implement NRR on top of BDOT (BDOT-NRR), without taking into account the nature of cache entries (faulty or non-faulty). This naive implementation offers a slight improvement with respect to BDOT-NRU for all cell types, but it is still worse than BD, except for C2, as in the case of BDOT-NRU. The rationale of this behavior is the blind allocation of blocks to entries, without taking into account if the entry can store only the tag (T) or both the tag and the data (D). Allocating a block that shows reuse to a T entry implies that all the requests to that block are forwarded to the next level (in this case, off-chip). Besides, due to the reused-based policy, this block will remain in the defective entry of the LLC, protected by the replacement algorithm. However, blocks with reuse allocated to D entries are also protected from replacement, and that explains why the relative differences between BDOT-NRR and BDOT-NRU are larger when using larger cells (i.e., with less faults, like C6 and C5).

BDOT-NRR-FA addresses this issue, adding the information of defective entries to the cache management policy. The penalization in terms of MPKI decreases with respect to BD by 14.6%, 15.1%, 16%, 18.3%, and 37.3% for cells C6, C5, C4, C3, and C2, respectively. If we compare BDOT-NRR-FA with BDOT-NRR, the MPKI decreases over 20%, irrespective of the cell type, demonstrating the goodness of the design.

Regarding WD, although there are significant differences in terms of the number of defective entries among the cell types considered (Table 2.1), the MPKI for the different configurations is almost constant. Two reasons explain this behavior: i) a single defective cell forces the entry to be classified as faulty, and ii) the number of defective cells per entry is usually small (three on average for the smallest cell, C2, as we show in Chapter 5) and, therefore, very often blocks are successfully stored by combining two consecutive entries. Thus, the average number of ways per set in our system when implementing WD is 8 across the different cell configurations. In comparison with BD, WD obtains better results when the average number of defective entries is greater than half, which is the case of cells C4, C3, and C2, as shown in Table 2.1. BDOT-NRR-FA lowers the MPKI with respect to WD by 20%, 16.1%, 8.5%, and 3.4%, for C6, C5, C4, and C3, respectively. WD only beats BDOT-NRR-FA in caches with a high number of defective cells (C2, where on average 90% of the entries are faulty). However, BDOT-NRR-FA requires no additional overhead, whilst WD requires additional storage and logic to reconstruct blocks.

4.5.2 *Parallel Workloads*

Figure 4.5b shows the relative LLC MPKI for the parallel workloads, with respect to the baseline. As with multiprogrammed workloads, BDOT-NRR-FA has a lower average MPKI than BD and non fault-aware implementations of BDOT. In particular, BDOT-NRR-FA improves MPKI with respect to BD by 5%, 5%, 9.6%, 19.2%, and 54.2% on average for C6, C5, C4, C3, and C2, respectively. Comparing with the multiprogrammed workloads, the relative MPKI numbers shown in Figure 4.5b are larger, moving away from the Robust system to a greater extent for all cell types, even for the winning alternatives (WD and DBOT-NRR-FA). But it is worth noting that the absolute



(a) Multiprogrammed workloads.

(b) Parallel workloads.

Figure 4.5: Normalized MPKI (average) with respect to Robust for the different proposals and cell types.

MPKI values for the parallel applications considered are low (Chapter 3), which makes the relative increases appear more substantial.

Digging into the results shows some interesting observations. Figure 4.6 shows the LLC MPKI analysis per application for the different cell types. BD is better than plain BDOTs (BDOT-NRU, BDOT-NRR) in C6-C3 cells (C3 in `streamcluster` is an exception), while in cell C2 the trend clearly reverses. On the contrary, BDOT-NRR-FA is better than BD in most cases, being `vips` the only exception (cells C6-C3), and giving very noticeable reductions in the smallest cell C2. For `vips`, BDOT-NRR-FA only beats BD in C2 because its image processing algorithm shows very little reuse with a small working set. In such non-demanding environment, BD can store the `vips` working set.

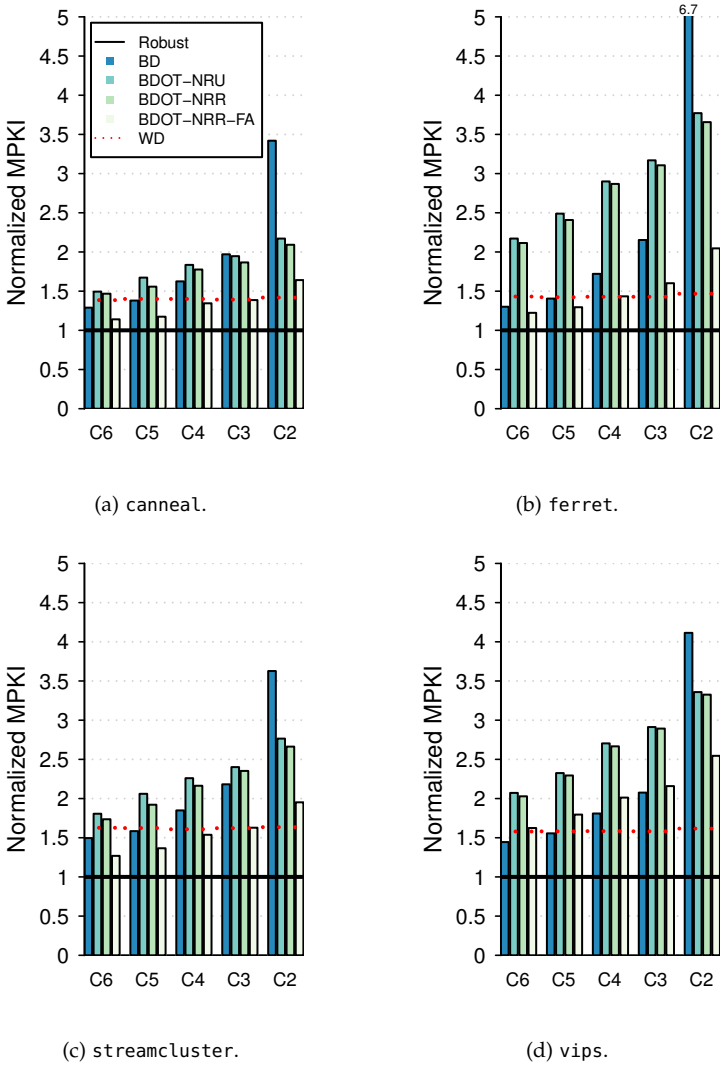
Finally, the costly WD shows a similar tendency than with multiprogrammed workloads, with a regular performance independently of the cell type. In this case, BDOT-NRR-FA beats WD when using C6 or C5, but it cannot reach WD performance for C4, C3, and C2, where MPKI increases by 5.5%, 12.4%, and 33.3%, respectively.

4.5.3 *Comparison with Prior Work*

Our proposal is orthogonal to the use of ECC to provide more functional entries (or any other technique that increases the number of functional entries), as it adapts seamlessly to the amount of functional and non-functional data entries in the cache.

Regarding block disabling improvements, Ladas *et al.* implement a victim cache to compensate for the BD associativity loss [93]. Our approach also relies on BD, but contrary to their work, it does not require any additional structure.

Several cache management techniques have been proposed in the context of caches built with heterogeneous cell sizes [56, 81]. As opposed to these techniques, we do not rely on the existence of robust ways and we guide



(a) `canneal`.

(b) `ferret`.

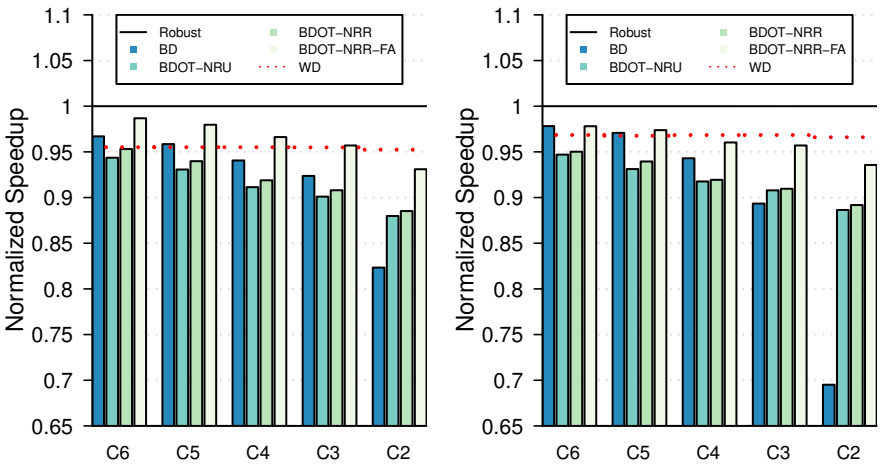
(c) `streamcluster`.

(d) `vips`.

Figure 4.6: Per-application normalized MPKI (PARSEC) with respect to Robust for the different proposals and cell types.

the allocation of blocks to faulty or operational LLC entries based on their reuse. Keramidas *et al.* use a PC-indexed spatial predictor to orchestrate the replacement decisions among fully and partially usable entries in first-level caches [80]. We base our allocation predictions in the reuse patterns, which simplifies the hardware, and we do not consider the use of partially faulty entries.

Complex re-mapping mechanisms such as [12, 88, 104] add a level of indirection to the cache access (increasing its latency), and the combination of cache entries to recreate a cache block adds complexity. Besides, several cache accesses are needed to obtain a fault-free cache block, increasing the energy consumption and/or the block access latency. We do not add any additional structure or re-mapping mechanism, only minor changes to the coherence protocol and replacement policy. We focus our detailed comparison on word disabling [157] as a representative technique of this group.



(a) Multiprogrammed workloads.

(b) Parallel workloads.

Figure 4.7: Normalized speedup (average) with respect to Robust for the different proposals and cell types.

4.6 SYSTEM IMPACT

This section analyzes the impact of our proposals on the system in terms of performance (instructions per cycle or IPC), area, and energy consumption. As in the previous section, we present results relative to the Robust configuration and compare against the BD and WD mechanisms.

4.6.1 *Performance*

Figure 4.7 shows the performance relative to Robust for both multiprogrammed and parallel workloads.

For multiprogrammed workloads (Figure 4.7a), performance follows the same trend as MPKI, being BDOT-NRR-FA the best design alternative except for C2, where WD outperforms BDOT-NRR-FA by 2.2%. In particular, BDOT-NRR-FA shows a performance degradation with respect to the reference Robust system of 1.3%, 2%, 3.4%, 4.3%, and 6.9% for C6, C5, C4, C3, and C2, respectively, or, in other words, a performance improvement with respect to BD of 2%, 2.2%, 2.7%, 3.6%, and 13.1%.

As in the case of multiprogrammed workloads, the speedup of parallel applications (Figure 4.7b) also follows the same trend as MPKI results, with a notable exception. For C3, BDOT-NRU and BDOT-NRR perform slightly better than BD on average, while in Figure 4.5b, the average MPKI of these techniques was larger than BD. As we already mentioned, the LLC MPKI of the parallel applications in the baseline system is small (Chapter 3), and small MPKI increases with respect to this system appear relatively large in Figure 4.5b. Nevertheless, for C3, `streamcluster` has a dramatic speedup degradation with BD. This is due to the large number of back invalidations to L1 blocks to force directory inclusion (inclusion victims). Namely, in this application, the number of invalidations to L1 blocks decreases 20x when implementing BDOT. The MPKI numbers are similar, but the number of instructions executed highly differ. For this application, we observe a perfor-

mance improvement of 6.1% when using BDOT-NRU (6.2% for BDOT-NRR), with respect to BD.

On average, BDOT-NRR-FA shows a similar performance than BD for C6 and C5, where the performance degradation with respect to the reference system is 2.2% and 2.9%, respectively, and improves its performance by 1.8%, 7.1%, and 34.6% for C4, C3, and C2, respectively. BDOT-NRR-FA and WD have similar performance (within 1%), except for C2, where WD achieves a 3.1% performance increase.

In summary, BDOT-NRR-FA is an excellent choice for caches with different number of defective entries, as it reaches the performance of more complex fault-tolerant techniques without adding any extra storage overhead to the cache.

4.6.2 *Area and Energy*

Larger SRAM cells have lower probability of failure, but at the cost of an increase in area and power consumption. Even the largest cell considered by Zhou's study (C6), which increases the area by 41.1% relative to C2, is far from reaching fully functional performance: 40.1% of the cache entries are faulty at 0.5 V (Table 2.1).

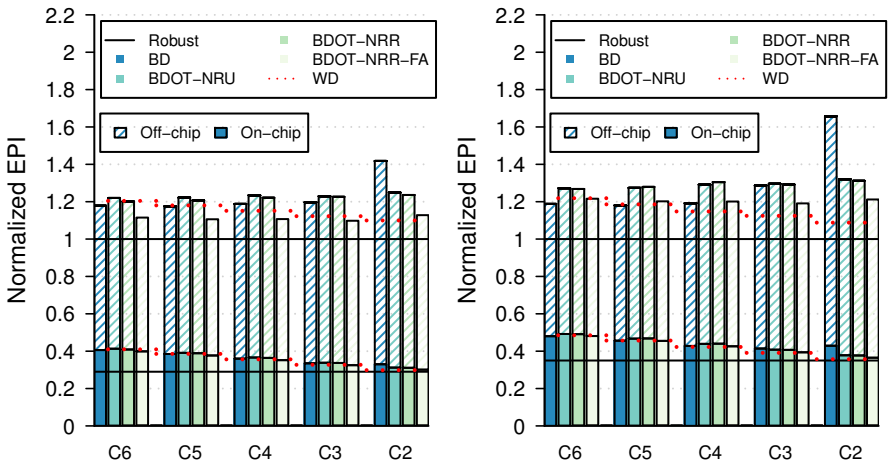
Our fault-aware mechanism has a minimal impact in area. Only two extra bits suffice to implement BDOT-NRR-FA: one bit marks entries as defective (as BD), and the other one stores the replacement policy (i.e., NRR) information. Thus, no extra storage overhead is added to the BD system.

Minimizing area helps to reduce energy in LLC. Signals travel smaller distances requiring less dynamic power for switching, and, most importantly, small cells consume less static power. To estimate the sub-threshold current, I_{sub}, causing the static consumption, we assume that I_{sub} is directly proportional to transistor width of the cells considered, and estimate it with respect to C2 [170]. For the unrealistically robust cell, we assume that its dimensions equal C2, but with a zero probability of failure. Energy consumption also includes the dynamic overhead of LLC block swaps and L1 clean data eviction required by the fault-aware BDOT policy. Finally, we account for both the on-chip power and the off-chip DRAM power.

Figure 4.8 shows the energy per instruction (EPI) for all the systems and cell types considered, both for multiprogrammed (Figure 4.8a) and parallel (Figure 4.8b) workloads, with respect to a system implemented with robust cells at 0.5 V, distinguishing between on-chip and off-chip consumption.

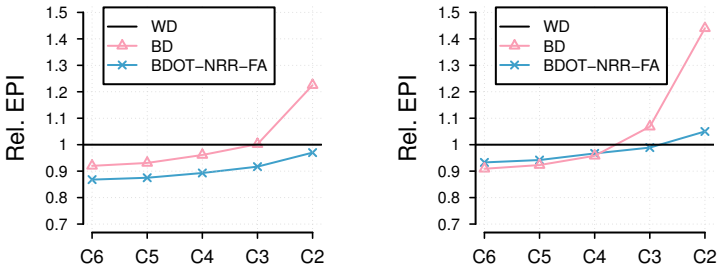
For BD, the 2.4× MPKI increment of C2 escalates the off-chip DRAM traffic, and in turn, significantly increases off-chip DRAM EPI for both multiprogrammed and parallel workloads. On average, BDOT-NRR-FA decreases the overall EPI with respect to BD by 5.4%, 5.8%, 6.8%, 8.2%, and 20.4% for C6, C5, C4, C3, and C2, respectively, for the multiprogrammed workloads. In the case of parallel workloads, the EPI of BDOT-NRR-FA is within 2% of BD for C6, C5, and C4, and it decreases by 7.4% and 26.8% for C3 and C2, respectively.

Regarding WD, the results show the same trend as performance: BDOT-NRR-FA EPI results improve by 7.5%, 9.8%, 7%, and 4%, for C6, C5, C4, and C3, respectively, when considering multiprogrammed workloads, while for parallel workloads the EPI values of both techniques are very similar for C6



(a) Multiprogrammed workloads. (b) Parallel workloads.

Figure 4.8: Normalized EPI (average) with respect to the unrealistically robust cell for the different proposals.



(a) Multiprogrammed workloads. (b) Parallel workloads.

Figure 4.9: Normalized EPI (average) with respect to word disabling, when implementing fine-grained block power gating.

and C5, but BDOT-NRR-FA cannot reach the efficiency of WD for the rest of the cell configurations.

The energy results shown above do not consider any block power gating technique [123]. Assuming a more aggressive approach, where fine-grained block power gating is affordable [58], the benefits of BD-based techniques in terms of power and energy will improve, as faulty entries do not consume static power during operation. Applying this technique, the EPI of BDOT-NRR-FA would decrease by 6.2%, 6.7%, 7.2%, 6.3%, and 5.5% for C6, C5, C4, C3, and C2, respectively, for the multiprogrammed workloads, with respect to the EPI values of Figure 4.8. The same tendency is observed in the parallel workloads results.

Figure 4.9 compares the EPI values of BD and BDOT-NRR-FA when implementing block power gating with respect to WD. We observe that for multiprogrammed workloads all the cell configurations achieve significant improvements in terms of EPI with respect to WD, and only the C2 configuration in the case of parallel workloads is not able to reach the WD efficiency.

4.7 SUMMARY AND CONCLUSIONS

Voltage reduction has been the primary driver to reduce power during last decades, but ultra-deep submicron technologies have suddenly stop this trend because of leakage and stability reasons. At lower voltages, manufacturing induced parameter variations make SRAM cells unstable, requiring a minimum voltage to operate reliably. SRAM cell failures can be tolerated by deactivating faulty cache entries. This technique is called block disabling (BD) and requires only one bit per tag. Unfortunately, as the number of defective entries increases, so does performance degradation, and the energy saved from decreasing V_{dd} does not pay off the extra energy of the additional main memory accesses.

The reduction in associativity and capacity experienced by inclusive LLCs extended with BD has two specific drawbacks in multicore systems. First, the number of inclusion victims in private L1 caches increases. Second, the MPKI figures also grow, increasing LLC miss latency and main memory energy consumption.

To cope with the first problem we propose Block Disabling with Operational Tags (BDOT), which uses robust cells to implement the LLC tag array. BDOT enables some cache blocks to be only in private levels by simply tracking their tags (T entries), and extends to clean blocks the existing cache-to-cache coherence service. Thus, with regard to inclusion victims, the LLC associativity is fully restored. BDOT requires a light extra control, and it adds no storage overhead to BD. Any replacement algorithm may work with BDOT, and we have tested NRU and NRR, two low-cost state of the art proposals for LLCs.

After the last copy L1 eviction of a block tracked by a T entry, a future reference to this block will involve an off-chip access, even if we know that reuse chances are high. So we tackle the second problem from the key observation that we can preserve the cached on-chip contents by exchanging the valuable, just evicted T entry block (promotion), with an L1-present D entry block (demotion). Furthermore, if all blocks allocated to D entries lack L1 copies, we can still resort to demotion, losing effective on-chip capacity, assuming that an incoming L1 block showing reuse (second L1 replacement) is more valuable than any older block allocated to a D entry. We have implemented these ideas in BDOT-NRR-FA, the fault-aware version of BDOT that selects for demotion a D entry victim block that has a backup copy in L1 (first criteria), and has not shown reuse in the LLC (second criteria). Compared to a BDOT LLC using NRR replacement, BDOT-NRR-FA improves performance and energy with no area overhead, because the required bits per block, namely presence vector, operative entry, and reuse are required, respectively, by coherence mechanism, BD, and conventional replacement.

We tested our proposals against a large range of multiprogrammed and parallel workloads under different P_{fail} situations. Our best proposal, BDOT-NRR-FA, beats BD, reducing MPKI up to 37.3% and 54.2% for multiprogrammed and parallel workloads, respectively. Those improvements translate into performance improvements of 13% and 34.6%, respectively. Regarding EPI, our proposal decreases it between 5.4% and 20.4% for multiprogrammed, and between 2% and 26.8% for parallel workloads. The larger savings come from LLCs with more faulty cells, making our proposal very suitable to operate multicore LLCs at low voltages for current and future technology nodes.

5 EXPLOITING REDUNDANCY AT CONTENT LEVEL

This chapter presents Concertina, an efficient and fault-tolerant last-level cache (LLC) that operates with unreliable SRAM cells at ultra-low voltages. Based on the observation that for many applications the LLC contains large amounts of null data, Concertina compresses cache blocks so that they can be allocated to cache entries with faulty cells. To distribute blocks among cache entries with different number of defective cells, Concertina also implements a compression- and fault-aware cache management policy that reduces the LLC miss rate and, consequently, the main memory accesses. Concertina reaches the performance of an ideal system implementing a LLC that does not suffer from parameter variations with a modest storage overhead. Specifically, performance degrades by less than 2%, even when using small SRAM cells, which implies over 90% of cache entries having defective cells, and this represents a notable improvement on previously proposed techniques.

5.1 INTRODUCTION

Concertina is an efficient and fault-tolerant LLC that operates with unreliable SRAM cells at ultra-low voltages. Unlike previous architectural schemes, Concertina enables all the cache entries, even the faulty ones. Our key idea is to compress cache blocks, in order that they fit within the functional elements of faulty entries. Since not all cache blocks can be evenly compressed and not all faulty entries can store the same amount of information, it is not possible to use existing cache management policies based on the premise that a block can be stored in any entry of a set. To address this issue, we study different insertion/replacement policies that are aware of the nature of both the incoming block (degree of compression) and the corresponding cache set entries (number of defective cells).

Compression has been proposed in related literature as a promising technique to increase the effective on-chip cache capacity [8, 34, 119, 162], or to reduce the energy consumption of the cache [33, 87], but to the best of our knowledge, it has not been explored as a way to enhance tolerance to cache faults in the context of ultra-low voltage operation. This application sets different requirements for the compression scheme. On the one hand, the probability of failure of a cache entry is high for conventional SRAM cells, and hence a large fraction of cache blocks have to be compressed. On the other hand, low compression ratios are good enough to recover a faulty cache entry, as the probability of having a high number of faulty cells in a given entry is very low. This latter observation, which is in line with the analysis of other authors [7], is also crucial to our proposal. Concertina implements null subblock compression, a fast, simple, and efficient scheme that takes advantage of the large amount of runs of zeroes present in the LLC blocks. Null subblock compression in combination with a smart compression- and fault-aware replacement policy results in performance within 2% of that of a system that implements the LLC with ideal defect-free cells.

The rest of this chapter is organized as follows. Section 5.2 explores compression requirements at ultra-low voltages and introduces our proposed

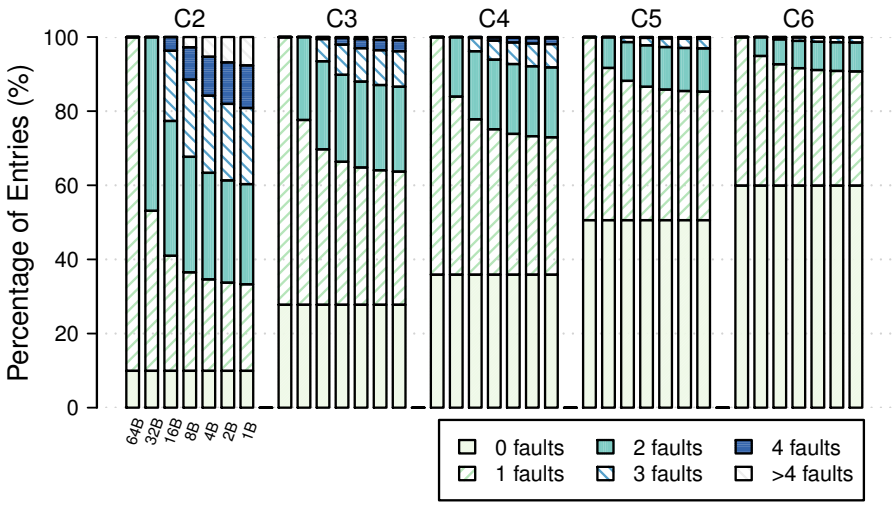


Figure 5.1: Distribution of faulty subentries for different subentry and cell sizes.

compression scheme. Section 5.3 presents the Concertina architecture. Section 5.4 presents the evaluation of our proposal. Section 5.5 discusses the system impact of Concertina, and Section 5.6 outlines our conclusions.

5.2 COMPRESSION FOR CACHES OPERATING AT LOW VOLTAGES

Data compression is a powerful technique for storing data in less space than originally required. In general terms, a compression mechanism seeks to be fast, especially at the decompression stage (decompression latency is on the critical path, while compression is done during replacement); and simple (the hardware and energy overhead should not overcome the benefit of the compression); as well as effective in saving storage capacity.

Compression is also an attractive idea for caches operating at ultra-low voltages because entries with defective bits could store compressed blocks, reducing the negative impact on capacity and associativity. To the best of our knowledge, this is the first time compression has been proposed for improving cache reliability at ultra-low voltages.

In this section, we first analyze the requirements imposed by near-threshold voltage operation for a compression scheme. Then, we present a simple yet effective compression scheme able to allocate most blocks to faulty cache entries.

5.2.1 *Compression Scheme Requirements for Caches Operating at Low Voltages*

State-of-the-art compression techniques aiming to increase the effective cache capacity focus on maximizing the compression ratio (uncompressed block size divided by compressed block size) rather than coverage (fraction of compressed blocks) [32, 119, 162]. In contrast, at ultra-low voltages, the main goal is to enable all the cache capacity. Hence, there are special requirements for a compression scheme related to its coverage and compression ratio, according to our parameters of interest, the cell and cache subentry sizes:

Coverage. The fraction of cache blocks that have to be compressed depends on the probability of cell failure. As faulty cells are spread across the cache structure, when the probability of cell failure increases (as cell size decreases), more blocks need to be compressed. Table 2.1 shows that, for the cell sizes considered, high coverage is required, ranging from 40.1% (C6) to 100% (C1).

Compression ratio. For a block to be stored in a faulty cache entry, its size has to be less than or equal to the available space at the cache entry (the size of its fault-free subentries). Otherwise, the matching is not possible. Due to the random component of the SRAM cell failures, some entries have more faulty subentries than others. Hence, the required compression ratio varies across entries. Assuming that we can track faults at different granularities (distinct subentry sizes), Figure 5.1 shows the distribution of defective subentries (from zero to four or more) for cells C2 to C6.

Irrespective of cell size, when the subentry size is reduced, the average number of faulty subentries per entry increases, as a subentry with multiple defective cells may spread across several smaller faulty subentries. However, as Figure 5.1 shows, even for the smallest subentry size (1-byte, 64 subentries) and the smallest cell considered (C2), only 7.6% of the cache entries have more than four faulty subentries. This fraction drops to 1% for the C3. Moreover, even with more faulty subentries, the disabled cache capacity drastically decreases when the subentry size is reduced (Figure 2.3).

Table 5.1 shows the compression ratio required to store a block in a cache entry for a range of defective subentries and subentry sizes. The compression ratio drops with decreasing subentry sizes. For instance, a 1.06 compression ratio suffices to allocate a block into a faulty cache entry with four faulty 1-byte subentries.

Summarizing, the main requirement of the compression scheme for caches operating at ultra-low voltages is *a high coverage, while the required compression ratio is very small*. In the rest of this chapter, we focus on cells C2, C3, and C4, as they are the most discouraging scenario (highest P_{fail}).

Table 5.1: Compression ratio required to store a 64-byte block in a cache entry with several faulty subentries of different sizes.

Subentry size (bytes)	**Compression ratio**			
	Faulty subentries			
	1	**2**	**3**	**4**
32	2.00	∞[a]	NA	NA
16	1.33	2.00	4.00	∞[a]
8	1.14	1.33	1.60	2.00
4	1.06	1.14	1.23	1.33
2	1.03	1.06	1.10	1.14
1	1.02	1.03	1.05	1.06

[a] A cache entry with two faulty 32-byte subentries or four faulty 16-byte subentries has no available space.

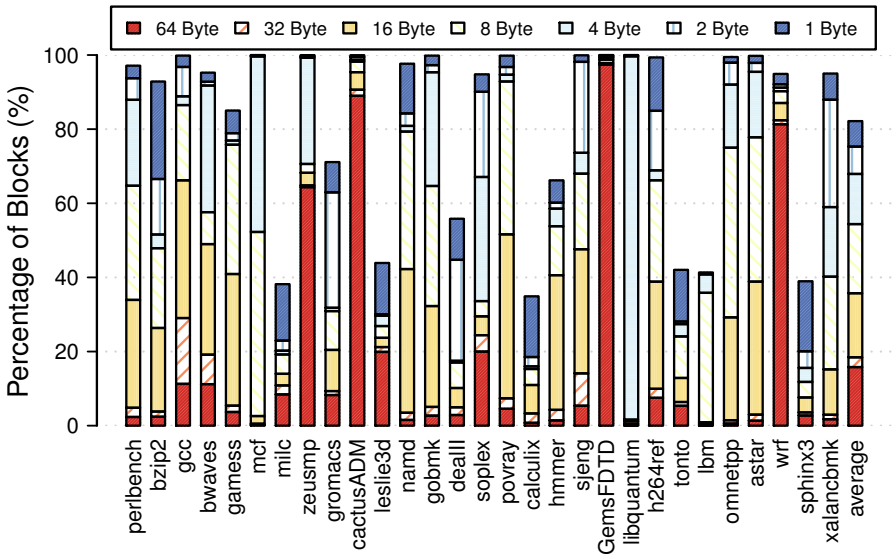


Figure 5.2: Average percentage of LLC blocks that have at least one null subblock for different granularities (SPEC CPU 2006).

5.2.2 *Exploiting Zero Redundancy to Compress Cache Blocks*

Content redundancy is prevalent among real-world applications. For example, zero is by far the most frequent value in data memory pages: It is used to initialize memory values, and to represent null pointers or false Boolean values, among others. Many compression schemes are based on compressing zeros or treating them as a special case [51, 119, 152, 162].

Exploiting zero redundancy can lead to a simple compression technique. We can compress aligned null subblocks[1] (subblocks whose content is all zeros) to effectively reduce the size of the original block. Compressing and decompressing null subblocks can be performed with low-complexity hardware: we just need to keep track of the locations of the null subblocks within a given block, and properly shift the non-null contents. This mechanism will be effective only if applications maintain a significant degree of compressibility during their execution. Hence, in this section we analyze the compressibility potential (number of null subblocks) of our target applications, and how it varies over the course of their execution.

To characterize null subblock occurrences in our applications, we conducted the following experiment. We ran the 29 SPEC CPU 2006 programs for one billion cycles (see Chapter 3 for methodology details) and inspected the contents of the LLC every one million cycles. Each application ran alone in the system, making use of the whole shared LLC (8 MB). Then we counted up the number of blocks that had at least one null subblock for different subblock sizes (from 64 to 1 byte).

Figure 5.2 shows the average percentage of blocks stored in the LLC that contain at least one null subblock of a given size. Notice that large null subblocks include several occurrences of smaller null subblocks (e.g., one null 64-byte subblock is equivalent to two null 32-byte subblocks). Different workloads show distinct behavior regarding subblock sizes and the amount

1 The same way we divide a cache entry into subentries, we divide a cache block into subblocks. Aligned subblocks simplify the design of the compression and decompression logic.

of null subblocks. While some workloads such as `GemsFDTD` and `cactusADM` show a significant percentage of blocks that have large null subblocks (64 bytes), most of them show a noticeable increase in the amount of null subblocks when reducing the subblock size. On average, downsizing subblock sizes from 8 bytes to 1 increases the average fraction of compressible blocks from 54.4% to 82.2%. A large set of benchmarks (including `gcc`, `povray`, `sjeng`, and `gobmk`, among others) reach almost 100% coverage when considering 1-byte subblocks. Regarding the temporal evolution of blocks, we checked that the percentage of compressible blocks is also maintained over the course of the execution of the applications, especially when considering small subblock sizes.

The conclusions extracted from our study are encouraging: not only can a significant percentage of blocks be compressed (high coverage), but also this percentage remains steady during program execution. This reinforces our belief that a null subblock compression mechanism, designed to be fast and simple, will also result in an effective technique to enable reliable LLC operation at ultra-low voltages.

5.3 CONCERTINA ARCHITECTURE

This section presents Concertina, our proposal to enable effective ultra-low voltage LLC operation by tolerating SRAM failures caused by process variations. Concertina implements null subblock compression and subblock rearrangement to enable otherwise non-usable cache entries when operating at voltages near the V_{th}. Concertina also incorporates a replacement policy to manage cache entries of different sizes, which may only contain compressed blocks. We first present the detailed implementation of Concertina. Then, we consider in detail the design of its replacement policy.

5.3.1 *Operation and Component Description*

Concertina consists of a conventional LLC plus the required logic to manage full blocks in faulty LLC entries (see Figure 5.3). To insert a block, either a refill from the main memory or a writeback from a lower cache level, Concertina performs two steps. First, it detects the location of its null subblocks if any. Then, it jointly uses this compressibility information and the location of defects in the corresponding cache entry to rearrange the non-null subblocks and store them in the functional subentries. To supply a block to a lower cache level or to evict a block, Concertina reconstructs the original block, rearranging the compressed block according to its metadata, which describes the location of null subblocks and defective subentries, and reintroduces the corresponding null subblocks.

To track the number and location of the defective subentries for each cache entry, and the number and location of null subblocks for each allocated block, Concertina uses two bit maps. Each map has as many elements as the number of subentries/subblocks considered; e.g., for a 64-byte block and a 16-byte subblock, the size of each map would be four bits. Each bit indicates whether the corresponding subentry is faulty (0) or not (1) for the fault map, FM, or if the subblock is compressed (0) or not (1) for the compression map, CM. For the sake of simplicity, we assume that the subentry and sub-

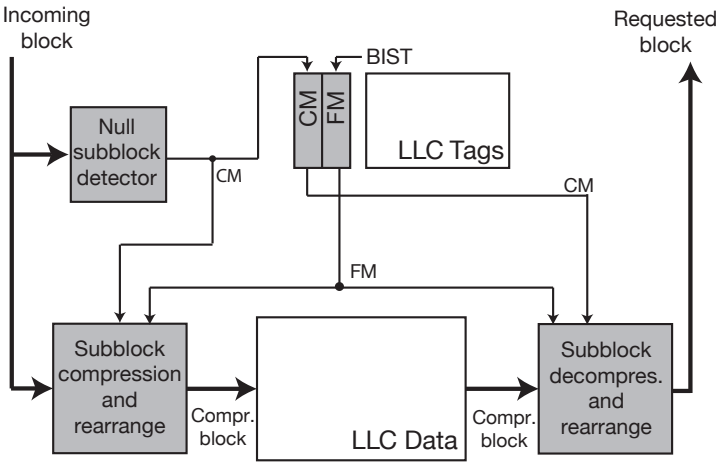


Figure 5.3: Concertina design overview. Shaded boxes indicate added components.

block sizes are the same. Later, in the evaluation section we present a more efficient implementation based on pointers for the two tracking structures.

The fault information, FM, comes from a characterization test (such as a built-in self-test, BIST) that can be executed at boot up time or during the first transition to the low-voltage mode [7, 12, 28, 157]. We consider that the number and location of faulty cells do not change during workload execution. The information about null subblocks, CM, is generated with a null subblock detector, and changes continuously during program execution. Section 5.4.3 discusses the implementation of these two components.

5.3.1.1 *Fault and Compression Metadata Sizing*

In terms of cell failures, finer granularities require lower compression ratios to allocate a block to a faulty cache entry. In terms of compression, smaller subblock sizes offer more compression coverage. However, finer granularities imply larger FM and CM sizes, which impacts area and power overheads.

We analyze this trade-off between performance and overhead varying the subblock size in our experiments. Using 16-byte or larger subblocks would cover less than one third of the cache blocks (see Figure 5.2), and, therefore, we will not consider these sizes.

5.3.1.2 *Subblock Compression and Rearrangement Logic*

In this section, we will use an example (Figure 5.4) to explain the subblock compression and rearrangement logic for a cache entry divided into four subentries (Figure 5.5). The goal of this circuit is to match non-null subblocks with non-faulty subentries. Let us suppose that an incoming block with one null subblock has to be inserted at a cache entry with one faulty subentry. The null subblock detector, implemented with logic that performs an OR operation on each of the subblocks, generates the CM for the incoming block, indicating the location of null subblocks. The FM indicates the location of the faulty subentries of the given cache entry.

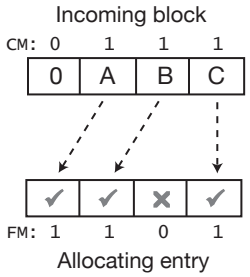


Figure 5.4: Example of compressible block to be stored at a faulty cache entry.

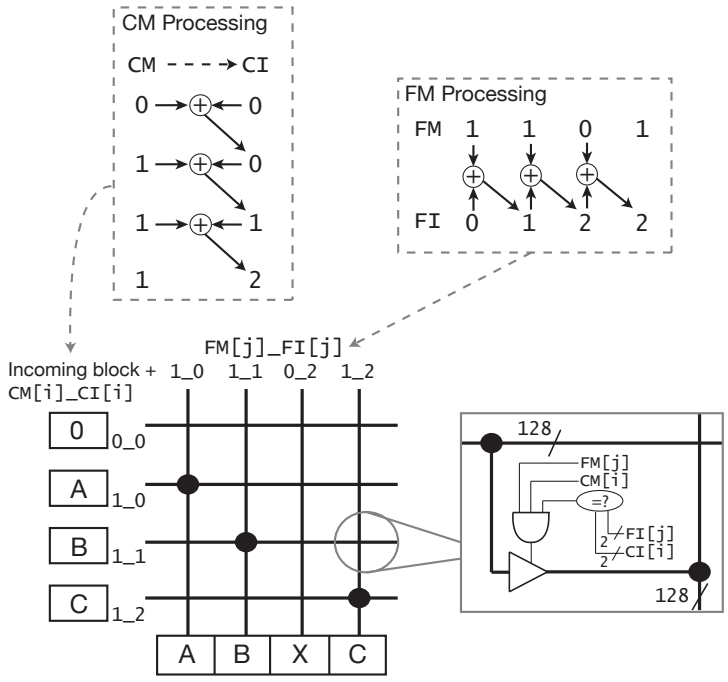


Figure 5.5: Implementation of the compression and rearrangement stage for a 64-byte block (512 bits) and a 16-byte subblock (128 bits).

First, the CM and the FM are processed to assign increasing identifiers to each of the non-null subblocks and for each of the non-faulty subentries, respectively. Thereby, compression and fault vectors of identifiers (CI and FI) are generated. As shown in the dotted boxes in Figure 5.5, this processing consists of adding the corresponding bits of the CM and FM, always using a zero for the first bit. In the general case, the bit width of each CI and FI entry is $\log_2(N)$, N being the number of subentries (subblocks) in a cache entry (block).

Next, each non-null subblock will be assigned to the non-faulty subentry whose generated indexes match. A crossbar-like logic controlled by CM, CI, FM, and FI vectors implements this matching (see Figure 5.5). The incoming block and the CM and CI vectors are input, on the left, and the FM and FI vectors, on the top. A crosspoint is activated only if both of its corresponding CM and FM bits are set; i.e., non-null subblock and non-faulty subentry, and if both CI and FI indexes are equal. This activation logic requires N^2 com-

parators of $\log_2(N)$ bits, and N^2 tri-state buffers. For example, a 4-subentry cache requires 16 comparators of 2 bits. When the subentry size decreases (finer granularity), both the number of the comparators and the size of their inputs increase (cache entries with 8 subentries require 64 comparators of 3 bits, etc.), but overall, the overhead is still low.

A similar logic is used to reconstruct a compressed cache block. Decompression is on the critical path, but Concertina's decompression stage is fully parallel and, due to its simple logic, we estimate that it can be performed in one extra cycle.

5.3.1.3 *Writeback Handling*

A common problem for cache compression mechanisms is handling writebacks. Compression techniques aiming to increase the effective cache capacity have internal fragmentation issues, because several blocks are stored in a single cache entry. Thus, blocks might need to be shifted properly before insertion [8, 119]. This issue can be overcome by using sparse super-block tags and a skewed associative mapping [135]. Nevertheless, Concertina does not have this problem, as only one block is stored per cache entry. On a writeback event, the block is compressed again. If it fits in its cache entry, the entry is updated; if it does not fit, the block is written back to memory. On average, we found that less than 2% of the writeback blocks are evicted because they do not fit in the same cache entry after writeback.

5.3.2 *Replacement Policy for Concertina*

Existing cache management policies rely on the assumption that a block can be stored in any entry of a cache set. However, caches with faulty cells have entries with a variable number of defective subentries, which can store only blocks that are compressed at least to a certain size. To overcome these stricter size requirements, Concertina implements a compression- and fault-aware replacement policy that takes advantage of the various degrees of compression that a block might have, and is able to evenly distribute all the blocks (both compressed and uncompressed) across the available cache capacity (both defective and non-defective entries). In the rest of this section, we will assume a baseline Least-Recently Used (LRU) replacement policy to simplify our explanations. Nevertheless, the algorithms described herein could work with other replacement scheme.

We present two replacement policies: *Best-Fit LRU* and *Fit LRU*. The first tries to find a precise match between cache entries and blocks in terms of the number of faulty subentries and the number of compressible subblocks. That is, once the block is compressed, the replacement algorithm searches for a victim among the entries whose effective size (number of non-faulty subentries) matches the size of the compressed block. The latter builds upon the first, but relaxes the matching condition, searching among all the cache entries where the block fits.

Let X_i be the group of blocks with i null subblocks of a given size, and E_j the group of entries with j faulty subentries of the same size, $0 \leqslant i, j \leqslant n$, where $n = \frac{\text{entry size}}{\text{subentry size}}$ (e.g., $n = 4$ for 64-byte cache entries and 16-byte subentries, as we might find entries with $0, 1, \ldots, 4$ faults). Therefore, *a block* $x \in X_i$ *can be allocated to an entry* $e \in E_j$ *if and only if* $j \leqslant i$*, but it cannot be allocated if* $j > i$. This premise is key for our replacement strategies.

A straightforward fault- and compression-aware allocation policy would try first to pair blocks and entries of the same index group, i.e., allocate blocks of type X_i to entries of type E_i. In the event that there is no entry of type E_i, it searches for entries of type $E_j, j < i$, until it finds a non-empty group. If the selected group has several entries, i.e., several entries have the same number of faulty subentries, the LRU information is used to select a victim among them. We call this the *Best-Fit LRU* policy, because it allocates blocks to the entries that best fit the size of the compressed block.

The Best-Fit LRU policy assumes that the distribution of blocks and entries of a given index group is well-balanced, but as we saw in Table 2.1 and in Figures 5.1 and 5.2, different cells and different workloads have different characteristics. For example, roughly 80% of the blocks in `wrf` contain 64 null bytes (X_n), while the majority of cache entries have between 0 and 4 faults, irrespective of the cell type. To compensate for the uneven distribution between entries and blocks, we can follow a different strategy and allocate a block X_i to any entry E_j where it fits, i.e., $j \leqslant i$. In this case, the victim block is selected using the LRU information about all the entries that belong to the E_j groups. We call this second policy *Fit LRU*, because it allocates blocks to entries where they fit, even if the allocating entry is not the best fit for a given block.

Note that both replacement policies make use of all the cache entries and allocate every block to the LLC. For instance, in the case where the percentage of non-compressible blocks (X_0) is higher than the percentage of non-faulty entries, blocks that cannot be compressed fight for non-faulty entries and, therefore, the pressure on the non-faulty entries is greater than the pressure on the faulty ones. In other words, although all the cache entries are utilized, in this example, the probability of a block of being replaced in a certain temporal interval is higher for blocks allocated to non-faulty entries than for blocks allocated to faulty ones.

It might be the case that a given block cannot be stored in its cache set, because the block size is larger than any of the available entries (e.g., a block cannot be compressed and all the entries have at least one faulty subentry). To overcome this potential problem, we do as follows. When the block arrives to the LLC from main memory, it is sent to the L1 cache as usual, but the coherence state of the block in the LLC specifies that the content of the block in the LLC is not valid. Future requests to that block cannot be satisfied by the LLC, and they will be forwarded to the L1 cache. This functionality already exists in the coherence protocol to forward exclusively owned blocks. After the L1 replacement, if the block still does not fit in the LLC entry, it is written back to memory.

Certainly, more complex policies are feasible. It could be expected that starting from the distribution of blocks and entries observed and adjusting the allocation of blocks according to the frequency of each block type would lead to more promising trade-offs. Unfortunately, after thoroughly evaluating such a strategy, we obtained similar results to those obtained with the Fit LRU policy, the latter being much simpler and with lower storage requirements.

5.3.2.1 *Replacement Policy Implementation*

Both Best-Fit and Fit LRU replacement algorithms must be able to select the LRU element among a subset of the blocks in a cache set. Two hardware modules implement these actions:

- *Definition of the subset of candidate blocks*: the null subblock detector generates the CM of the incoming block. A hardware bit-counter computes the number of zeros in the CM. This value (CM_{count}) is the number of null subblocks in the incoming block, and is used to classify the block into a given index-group X_i. Likewise, a group of hardware bit-counters compute the number of zeros in the FM for all the cache entries in the cache set, and the result is a vector, FM_{count}, whose elements indicate the number of faulty subentries in each cache entry. These values (FM_{count_k}) are used to classify the entries into E_j groups. The comparison between each FM_{count_k} and CM_{count} generates a bit vector that represents the subset of candidate blocks to replace according to the fit criteria (Best-Fit or Fit).
- *LRU selection in the subset*: our mechanism maintains the LRU order among all the elements in a cache set. The replacement algorithm applies the LRU selection logic to the subset of candidate blocks [73].

5.4 EVALUATION

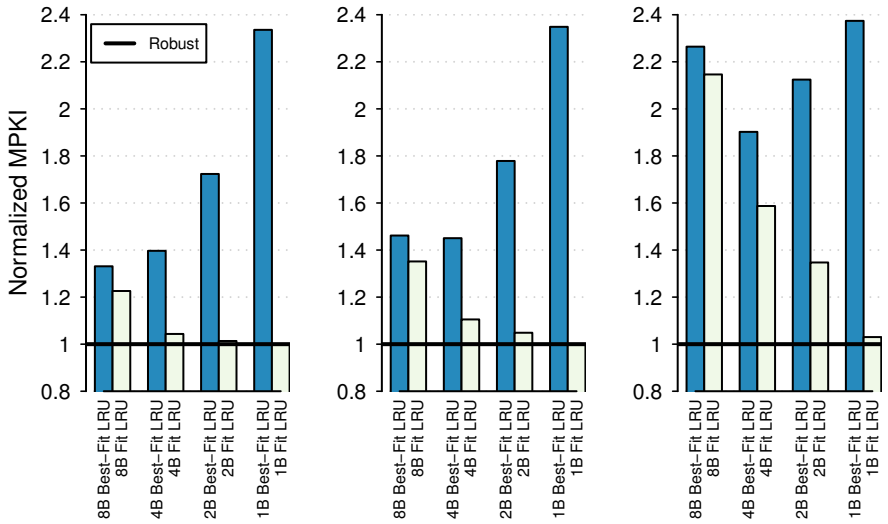
In this section, we present the evaluation results for Concertina. We first analyze the different design options for Concertina in terms of LLC MPKI. Then we explore a new implementation that reduces the storage requirements and evaluate the Concertina overhead. Finally, we compare our proposal with prior work.

5.4.1 *Replacement Policy*

Here we explore the impact on the LLC MPKI of the replacement policies proposed in the previous section: Best-Fit LRU and Fit LRU, with respect to an unrealistically robust cell (*Robust*); i.e., a cell operating at an ultra-low voltage but with no failures and no power or area penalization.

Figure 5.6 shows the increase in LLC MPKI with respect to the unrealistically robust cell for both strategies and different subentry/subblock sizes (8, 4, 2, and 1 byte), for cells C4, C3, and C2. The Best-Fit LRU strategy needs an even balance between the distribution of blocks X_i and entries E_i. However, as the distribution is usually not balanced, this replacement strategy fails to exploit the full potential of Concertina. The observed trend is that the smaller the subentry size, the poorer the performance. This happens because there are more non-empty block groups (X_i) than non-empty entry groups (E_j). For example, considering 1-byte subblocks, there are 65 X_i groups, most of them likely to be non-empty, while due to the small number of faults per cache entry (Figure 5.1), the number of non-empty groups E_j will be around 5 ($E_{0...4}$), which increases the pressure on the E_j groups with the higher number of faulty subentries (E_3 and E_4).

If we focus on the Fit LRU replacement strategy, Figure 5.6 reinforces the intuition that the smaller the subentry/subblock size, the smaller the increase in MPKI. The best configuration is 1-byte subblock size, irrespective



(a) C4 (b) C3 (c) C2

Figure 5.6: LLC MPKI relative to an unrealistically robust cell, for the different replacement alternatives and subblock sizes.

of the cell size, because we recover more capacity and fit a larger number of compressed blocks in faulty entries (compression coverage increases). In comparison with the unrealistically robust cell, there is no MPKI increase when using C4 or C3 cells, and the number of MPKI increases only by 3% when C2 cells are used. However, for the smaller subblock sizes, the overhead is intolerable in terms of area: the bit map for implementing 1-byte subblock size requires 128 bits (64 bits for the CM and 64 bits for the FM), which corresponds to 25% of the 64-byte cache entry. In the next subsection, we explore alternatives to significantly reduce storage overhead with a small impact on performance. From now on, we assume the Fit LRU policy.

5.4.2 *Reducing Concertina Storage Requirements: Implementation with Pointers*

The fault and compression metadata (FM and CM) can be encoded in several ways. As stated before, even for the smallest subentry size and the smallest cell, only 7.6% of the cache entries have more than four faulty subentries, which leads us to propose a new implementation based on pointers. For the sake of brevity, we only detail the implementation for the FM, but the same applies to the CM.

Instead of using a bit map, we could use four pointers to identify faulty subentries. Entries with more faults than pointers are disabled, because there is not enough space available to store the required fault information, reducing fault coverage. However, as the storage requirements are still high, we also consider implementations with three and two pointers.

Let us consider an example with 8-byte subentry size and three pointers, each of three bits. Specifically, the three pointers 000-011-101 are equivalent to the bit map 11010110. Although the maximum number of pointers is fixed per cache entry, not all of them are always valid (an entry may have

Table 5.2: Concertina storage requirements (bit map vs. pointer implementation) and potential fault coverage for cells C4, C3, and C2. The fault coverage for the bit map approach is always 100%.

	Storage Requirements (bits)				Fault Coverage (% entries)								
Subblock size					**C4**			**C3**			**C2**		
(bytes)	**Map**	**4 Ptrs**	**3 Ptrs**	**2 Ptrs**	**4 Ptrs**	**3 Ptrs**	**2 Ptrs**	**4 Ptrs**	**3 Ptrs**	**2 Ptrs**	**4 Ptrs**	**3 Ptrs**	**2 Ptrs**
8	16	26[a]	20[a]	14		99	94		98	90	97	88	68
4	32	34[a]	26	18	≈ 100	98	93	≈ 100	97	88	94	84	63
2	64	42	32	22		98	92		96	87	93	82	61
1	128	50	38	26		98	92		96	87	92	81	60

[a] The storage overhead is greater using pointers than a bit map.

fewer faulty subentries than the three available pointers). To overcome this issue, we can store redundant pointers: the pointers 000-011-011 are equivalent to the bit map 11110110, which represents a cache entry with only two faulty (0) subentries. One extra bit is needed to distinguish between an entry with one fault—000-000-000-(1)—and a non-faulty entry—000-000-000-(0). Finally, another bit indicates whether the entry has more faults than the number of available pointers. Thus, the storage required by this pointer-based implementation (fault and compression metadata) is:
$2 \times (\mathit{number\ of\ pointers}) \times \log_2(\mathit{number\ of\ subblocks}) + 2$ bits per cache entry.

Table 5.2 compares the storage and fault coverage values of both approaches (bit maps and pointers). We can conclude that using three to four pointers per cache entry offers a good trade-off between storage overhead and fault coverage. For C4 and C3, three pointers would suffice to cover more than 96% of the cache entries. Conversely, due to the higher probability of SRAM cell failure, C2 would require four pointers per cache entry for a fault coverage of at least 92%. If we seek a lower overhead implementation, two pointers would cover around 90% of cache entries for C4 and C3, but the coverage would drop to 60% for C2.

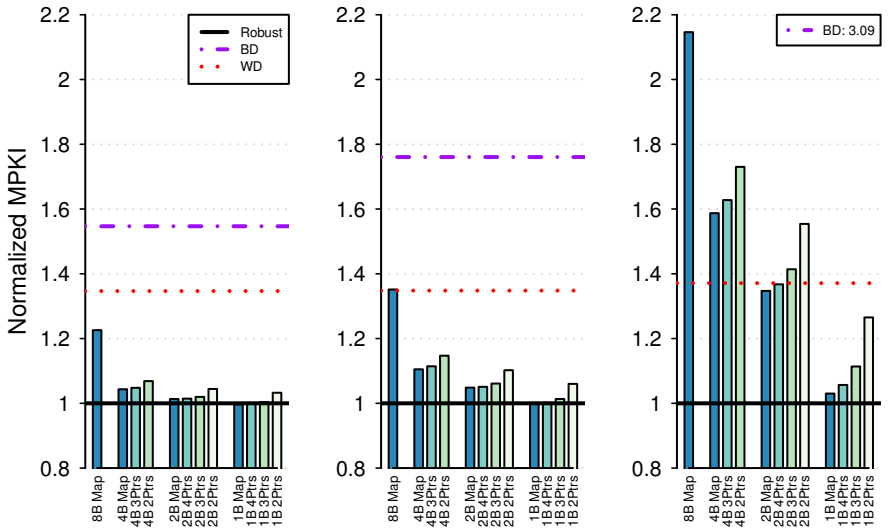
In terms of the design of the compression/decompression and rearrangement mechanism, the pointer proposal only affects the processing of the FM and CM. After recovering the pointer information from the corresponding metadata structure, a small set of decoders can transform these pointers to bit maps, leaving the compression/decompression, rearrangement, and replacement logic unaffected.

The drawback of the pointer-based approach is that all entries with more faults than the number of available pointers have to be disabled. Nevertheless, these entries can store whole null blocks with neither extra complexity nor storage overhead, enabling 100% of the LLC capacity. The bit used to mark that the entry has more faults than the available number of pointers now indicates that the cache entry, if valid, contains a null block (all zeros).

5.4.2.1 *Pointer-based Implementation Evaluation*

Figure 5.7 shows the LLC MPKI increase for cells C4, C3, and C2 . Results are showed for different granularities (8, 4, 2, and 1 byte) and for both implementations: map and pointers (considering 2, 3, and 4 pointers per entry). For simplicity, we will follow a *subentry-size implementation* name convention, e.g., *1B 3Ptrs* refers to subentry size of 1 byte, implemented using 3 pointers. We will only consider design choices that reduce the storage requirements, because in terms of performance, the implementation based on maps will always be preferable.

We find that pointers are consistently a good alternative to bit maps. We obtain results within 1-2% of those obtained with bit maps for cells C4 and C3 and greatly reduce the storage requirements (e.g., from 64 to 32 bits for the *2B 3Ptrs* configuration). In the case of C2, the high percentage of defective cells compels Concertina to use 1-byte subblocks. In this context, the pointer-based approach significantly reduces the implementation overhead, from 128 bits to 50, 38, and 26 bits, at the cost of MPKI increases of 5%, 11%, and 26% for 4, 3, and 2 pointers, respectively.



(a) C4 (b) C3 (c) C2

Figure 5.7: LLC MPKI for C4, C3, and C2 cells, and the different compression alternatives and subblock sizes.

5.4.3 *Concertina Overhead*

Concertina's main overhead is the storage of the FM and CM metadata. We saw that the use of pointers greatly reduces the storage requirements with respect to bit maps. Here we evaluate the impact in terms of latency, area, and energy of the added storage for the pointer-based implementation. We consider the configurations with the best performance results: 1-byte 4-, 3-, and 2-pointer configurations. To quantify these costs we use CACTI [95].

Assuming the cache organization of Table 3.1 and 40 bits of physical address in a 64-bit architecture, a data array entry has 512 bits, while a tag array entry has 39 bits: 21-bit tag identifier, 14 bits for coherence information (5-bit coherence state, 8-bit presence vector, and 1 dirty bit), and 4 bits for the replacement related information (LRU implementation of $\log_2(\mathtt{associativity})$ bits per cache entry). Concertina adds 50, 38, and 26 bits per cache entry for 4, 3, and 2 pointers, respectively. Although the increase in storage requirements with respect to the tag array is significant, the impact is small with respect to the whole LLC structure. Table 5.3 summarizes the overheads for 4, 3, and 2 pointers in terms of area, latency, and leakage with respect to the whole baseline cache structure. We also show the dynamic energy consumption for a cache read access to a non-faulty entry, and a cache read access to a faulty entry. However, static energy dominates the overall consumption of the LLC, even more when operating at voltages near V_{th} [79]. Hence, the observed variations in the dynamic energy have no significant impact on the total consumption of the system.

The FM and CM can be organized in different ways, and these offer different trade-offs. As a first approach, both FM and CM are integrated into the tag array. For 3 pointers, area and leakage increase 5.4% and 5.9%, respectively, with respect to the baseline cache structure. After a tag hit, the logic for decompression can be pre-charged with the metadata. On every

tag access, we are reading extra information that is only useful in case of a cache hit, which translates to an increase in the tag array latency of 14%, 10%, and 7% for 4, 3, and 2 pointers, respectively. If there is not enough slack available, this design would increase the tag array latency, penalizing every cache access. Nevertheless, a one cycle latency increase in the tag array access would have a minor impact in the overall performance (less than 0.2% performance drop).

An alternative design stores the FM and CM in separate SRAM structures. In this case, a bit in the tag array indicates if the entry is faulty and, therefore, the FM and CM must be accessed. The metadata is now only accessed in case of a tag hit to a faulty entry. The decompression logic can still be pre-charged before the data block is available, because the access to the metadata arrays is faster than the access to the data array. For 3 pointers, area and leakage increase 0.8% and 0.3%, respectively, compared to the first approach. At the cost of slight area and static power penalties, the separate design avoids the extra latency to access the tag array. Besides, storing the metadata in separate arrays makes it easier to power gate the structures in high-voltage/high-frequency modes, minimizing the impact on power and energy consumption.

In both designs, the power overhead is negligible compared to the reduction in both static and dynamic power obtained by working at ultra-low or near-threshold voltages (dynamic and static power scale quadratically and linearly with supply voltage, respectively), while the area overhead is negligible compared to the area savings obtained by using smaller SRAM cell transistors.

Table 5.3: FM and CM overheads with respect to the cache structure.

Configuration	**Area**	**Latency**	**Leakage**	**Read Energy** [a] (access/access-faulty)
Baseline	1.000	1.000	1.000	1.000/NA
FM and CM integrated in tag array				
1B 4Ptrs	1.071	1.041	1.078	1.035/1.035
1B 3Ptrs	1.054	1.030	1.059	1.027/1.027
1B 2Ptrs	1.038	1.020	1.041	1.019/1.019
FM and CM separate structures				
1B 4Ptrs	1.082	1.000	1.081	1.000/1.066
1B 3Ptrs	1.063	1.000	1.062	1.000/1.051
1B 2Ptrs	1.044	1.000	1.044	1.000/1.036

[a] With respect to a read access in the baseline configuration.

Regarding the compression/decompression logic, and with the *1B 3Ptrs* configuration, we could take advantage of the restriction in the number of subblock shifts: 3 pointers mean that a given subblock can move a maximum of three positions to the right or three positions to the left.

Thus, the original matrix of comparators becomes a band matrix with 7 elements per row, and a 7:1 multiplexer can substitute each column of tri-

state buffers. The corresponding logic could be implemented with 436 6-bit comparators and 512 7:1 multiplexers, which roughly translates to 8 K logic gates. This overhead is insignificant, in comparison with the total LLC size: it represents approximately 0.01% of the data array of the on-chip LLC (i.e., without accounting for the tag array and the decoder logic). Since cache memories normally have zero slack, we assume an extra cycle in the read operation to decompress the cache blocks. Compression latency is hidden, as it occurs during the cache block allocation.

5.4.4 *Comparison with Prior Work*

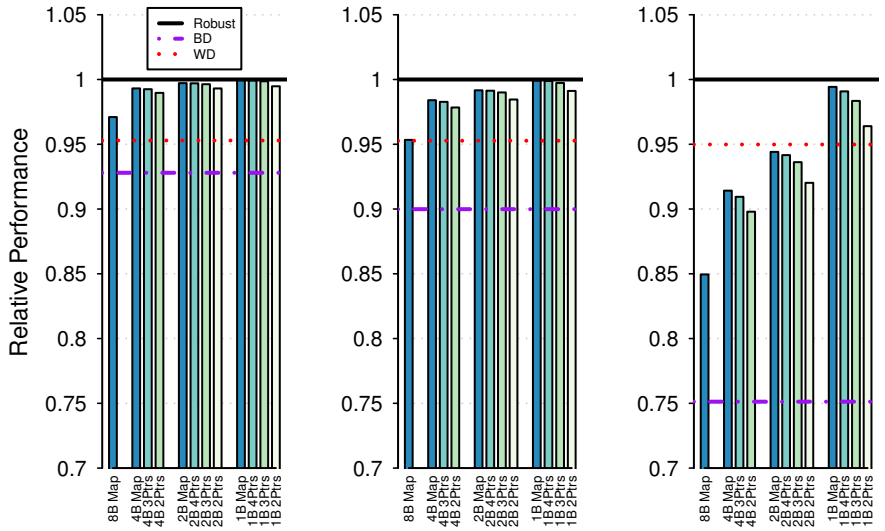
Regarding the compression mechanism, at ultra-low voltage operation, our objective is to maximize compression coverage rather than compression ratio. State-of-the-art techniques aiming to increase the LLC capacity seek to maximize compression ratio, e.g., BΔI has an average compression ratio of 1.5 for the SPEC CPU 2006 benchmarks, but its coverage is around 40% [119]. Any compression technique aiming to maximize coverage could be used, but we opted to use null subblock compression in Concertina given its simplicity and effectiveness.

Regarding the replacement algorithm, recent studies on cache block compression have underlined the importance of taking into account compression information for the replacement decision [14, 120]. These studies are based on the fact that several blocks are allocated to the same cache entry, while in the case of Concertina only one block is stored per cache entry. Thus, evicting a block with a given compression ratio does not directly translate into an increase in the number of blocks stored at the LLC.

ECCs are orthogonal to the Concertina design, and they should be implemented as well to protect against soft errors. Although ECCs have also been proposed to correct and detect hard errors, they require either large storage overhead or complex detection and correction logic. For example, by using orthogonal Latin square codes as in [39], the decoding latency would be comparable to Concertina, but at the cost of storing 138 check bits to correct 3 errors, while the *1B 3Ptrs* implementation has a storage overhead of just 38 bits.

Combining techniques such as [12, 88, 104] require complex algorithms to find collision-free cache entries across all the cache structure, and the storage of all the remapping strategy. Moreover, schemes that distribute blocks into several entries need to access two or more entries to recover a single block, with the corresponding latency and energy penalty.

We focus our detailed comparison on two techniques: block disabling (BD) [142] and word disabling (WD) [157]. Block disabling is implemented in modern processors operating at nominal voltages, and some authors advocate its use at lower voltages [93]; word disabling is similar to ours in complexity. Figure 5.7 shows the increase in MPKI for these techniques with respect to the unrealistically robust cell. We see that, irrespective cell size, Concertina always results in lower LLC MPKI values than both the other techniques. In fact, for C4 and C3, the implementation with least area overhead (*8B Map*) produces fewer MPKI than WD. In the case of C2, the *8B Map* implementation fails to reach the performance of WD, but the 1-byte implementations greatly improve on WD results.



(a) C4 (b) C3 (c) C2

Figure 5.8: Performance relative to Robust for cells C4, C3, and C2, and the different compression alternatives and subentry/subblock sizes.

5.5 SYSTEM IMPACT

This section analyzes the impact of Concertina on the system in terms of performance (instructions per cycle or IPC), area, and power consumption. We compare Concertina with the unrealistically Robust and the fault-tolerant mechanisms block and word disabling.

5.5.1 *Performance*

Figure 5.8 shows the performance relative to Robust for the two Concertina implementation alternatives (bit maps and pointers), and cells C4, C3, and C2. This confirms the *1B Map* as the best design approach in terms of per-

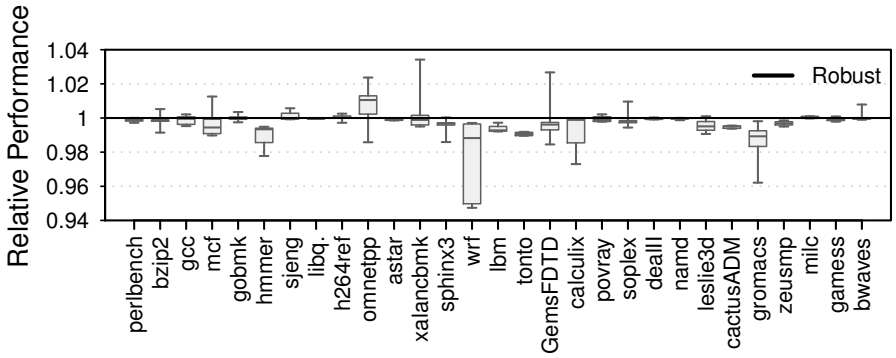


Figure 5.9: Per-application performance analysis for the *1B 3Ptrs* implementation (cell type C3).

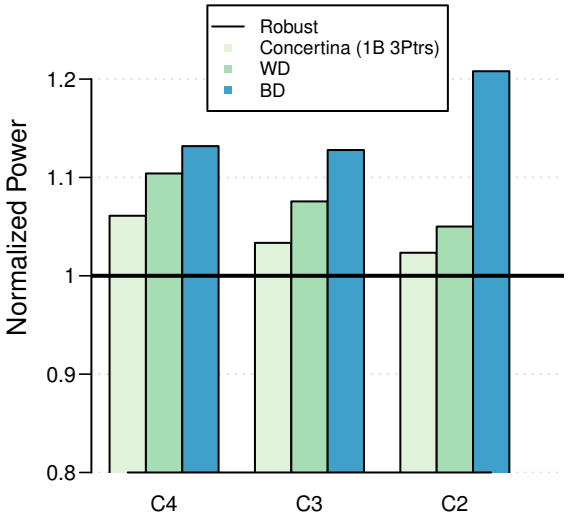


Figure 5.10: System power consumption relative to Robust.

formance, achieving the same performance as an unrealistically robust cell for C4 and C3, and a minimal 0.5% degradation for C2. For C4 and C3, the pointer approaches have a performance within 1% of that of the bit maps, while it degrades up to 1%, 2%, and 3% for C2 when using 4, 3, and 2 pointers, respectively.

Performance follows the same trends as LLC MPKI regarding BD and WD. WD has uniform performance across all cell sizes because there are usually fewer than four faults per cache entry, and hence it succeeds in combining consecutive entries to store a complete cache block. On average, it degrades performance approximately 5% with respect to the robust non-faulty cell. BD impairs performance by 25% for C2 when the available cache capacity is scarce, and most of the LLC accesses become misses.

Concertina performance results are also consistent across all the programs considered. Figure 5.9 shows the distribution of IPC per application for the *1B 3Ptrs* implementation (cell C3), with respect to the robust cell. For each application we show a boxplot with the minimum, first quartile, median, third quartile, and maximum of the distribution. Most applications show minimal performance degradation with respect to the robust cell: in the worst cases, the median represents a 1% performance degradation. In some cases, performance improves slightly, since applications with more compression potential (more blocks that can be compressed), such as `omnetpp` or `mcf`, have higher chances of finding an available entry where a block fits.

5.5.2 *Area and Power*

Larger SRAM cells have lower probability of failure, but at the cost of an increase in area and power consumption. Even with small cells (C2), Concertina provides performance comparable with an ideal cell. Even the largest cell considered by Zhou's study (C6), which increases the area by 41.1% relative to C2, does not accomplish fully functional performance: 40.1% of the cache entries are faulty at 0.5 V (Table 2.1). In comparison to this increase,

the extra area required by Concertina (6.3% for the *1B 3Ptrs* implementation) is a reasonable overhead, largely compensated for by its performance results.

Regarding power, our design focuses on the LLC, a structure where static energy dominates the overall consumption. Therefore, we estimate that the use of larger cells mainly affects the static energy of the LLC. I_{sub} increases with transistor width, and hence, based on the transistor widths of the cells considered [170], we estimate the increase in I_{sub} relative to C2. We assume that the unrealistically robust cell has the same dimensions as C2, but with a null probability of failure. We also add the dynamic and static overhead of the metadata computed in Section 5.4.3, assuming the separate design. Finally, we account for both the on-chip power and the power of the off-chip DRAM module.

Figure 5.10 shows the power consumption for Concertina's *1B 3Ptrs* implementation, WD, and BD, with each of the cell sizes considered (C4, C3, and C2). Concertina is always the best option, very close to the ideal configuration in the case of C2. Concertina and WD follow a similar trend, the power consumption decreasing when downsizing the cell size. For BD, when the LLC is implemented using C2 cells, the large increase in off-chip DRAM traffic translates to a significant power consumption increase in the overall system.

The above results confirm Concertina to be an attractive LLC cache design to operate at ultra-low voltages, as it exhibits performance and power requirements comparable to a robust cell with a null probability of failure, even with a limited fault-free fraction of the LLC (9.9% when using C2 cells).

5.6 CONCLUSIONS

Scaling supply voltage presents a significant challenge to improving processor performance, especially for SRAM cell transistors used in cache memories. Lower voltages reduce cell reliability, which effectively reduces cache capacity and, therefore, performance.

Existing micro-architectural approaches to increasing cache reliability focus on enabling and combining the valid cells, thereby reducing the available cache capacity. For programs with large footprints and/or working sets, the extent of performance degradation is substantial. We depart from these approaches and propose Concertina, an LLC that compresses cache blocks to reduce their size in order that they can fit into entries with non-functional cells. Concertina ensures the use of 100% of the LLC capacity through a low overhead insertion/replacement policy that combines block compressibility and fault awareness to enable smart allocation of blocks to cache entries.

We explore three implementations with cells that trade off area and power for reliability, namely C4, C3, and C2. Concertina's *1B Map* design exhibits a negligible MPKI increase relative to a defect-free LLC for C4 and C3 cells, and just a 3% MPKI degradation for the C2 implementation, but at the cost of large storage overhead. A lower-overhead design based on pointers (the *1B 3Ptrs* configuration) greatly reduces Concertina storage requirements, with a minimal performance degradation of less than 1% for cells C4 and C3, and 2% for C2, with respect to the unrealistic defect-free LLC.

Part III

EXPLOITING REDUNDANCY TO REDUCE THE ISA-COMPLEXITY: APPROXIMATE INSTRUCTION SET COMPUTER

6

EXPLOITING THE INTRINSIC NOISE TOLERANCE OF EMERGING APPLICATIONS: APPROXIMATE COMPUTING

Emerging application domains such as Recognition, Mining, and Synthesis (RMS) rely on iterative, often probabilistic algorithms for processing massive yet noisy input data sets. Due to their nature, these applications can intrinsically tolerate certain error in their computations. Furthermore, these applications do not usually have a unique or golden valid output, and different outputs might be acceptable or precise enough. Approximate computing deliberately introduces "acceptable errors" into the computing process, with the objective of obtaining significant energy gains, within a restricted accuracy loss in the outputs. In this chapter, we provide a general overview of the approximate computing paradigm and some of its main contributions.

6.1 INTRODUCTION

The advances in computation and technology are changing the way we look into the world. We use computations in our daily life, and bring powerful devices in our pockets, wrists, clothes, or even inside our body. We increasingly rely on computers for our daily tasks, and it has been estimated that by 2020 the Internet of Things (IoT) will grow to 26 billion installed units[1].

A large percentage of these emerging applications interact with the physical world, process large amount of data obtained from noisy sources, such as sensors, and provide results which do not need to be 100% accurate. For example, video encoders take advantage of the fact that the human eye will not notice bits of missing information to drop frames and reduce the size of video files. A picture sent via *WhatsApp* is lossy compressed from several MBs to several KBs, trading-off quality (accuracy) for fast communication (and hence, energy). Internet is in a continuous change, with content added and deleted every second, including text, pictures, and video. Google search processes on average 40,000 queries a second (over 3.5 billion searches per day[2]); the answers to these queries are not unique, and many results are admissible for the same question. Applications with iterative refinement algorithms can run several intermediate computations with less precision without losing accuracy on the final result.

In this context, where the amount of information to process keeps growing, it is essential to improve the energy efficiency of these emerging workloads [35]. Fortunately, these applications usually tolerate a certain amount of noise, featuring an intrinsic error-resilience property [37]. Their intrinsic redundancy comes from their inputs, often noisy and redundant (e.g., sensors), and their computation patterns, often stochastic in nature. Furthermore, these applications outputs do not need to be unique or "golden", or they can tolerate a certain amount of error due to, for instance, perceptual limitations.

1 http://www.gartner.com/newsroom/id/2636073; last access: April 2016
2 http://www.internetlivestats.com/google-search-statistics/; last access: April 2016

Approximate computing takes advantage of this property and exploits the gap between the level of accuracy required by the application and the level of accuracy given by the computation, providing that reducing this accuracy translates into an energy gain. Approximate computing has been used in a variety of domains, such as image processing or multimedia [168], machine learning [50], signal processing [11], or scientific computing [131]. Some approximation techniques include precision reduction [164], skipping tasks [132], memory accesses [108], or some iterations of a loop [110], relaxing synchronization [109], performing an operation on inexact hardware [74], or voltage scaling [38], among others.

One of the main limitations of approximate computing is to decide whether a given code region or data structure can be safely approximated. Too aggressive approximation or approximating the wrong piece of code or data might result in worthless results or, even worse, safety issues and abnormal termination. The compiler or the programmer can expose which data structures or regions of code can be approximated [133] or use dedicated approximate data types [21]. The approximation technique, on the other hand, needs to be selected on an application basis.

We take a novel approach and explore a more aggressive approximation strategy: approximate ISA (Instruction Set Architecture), inspired by the fact that for current ISAs, there is a percentage of instructions that are rarely (or never) used [100] and, therefore, can be approximated or removed, and that for the applications we consider, a small subset of instructions correspond to most of the execution time. On top of that, all the work done by most used instructions is not always required. For instance, there are instructions that can be approximated by sequences of other instructions, or, in a floating point operation between a very big and a very small value, the result can be approximated by returning the first operand without performing the addition or by adding the high order bits of the small value. Next section delves into these ideas.

6.2 REDUCING THE GENERAL-PURPOSE PROCESSORS COMPLEXITY

An increasingly amount of resources are dedicated to hardware accelerators. Accelerators are a good choice for dark silicon, because they contribute to the overall performance but only consume power in special situations. A potential limitation is that specialized accelerators are conceived to tackle very specific tasks. Chung *et al.* studied the performance potential of GPU, field-programmable gate array (FPGA), and application-specific integrated circuit (ASIC), regarding area, power, and memory bandwidth [42]. Their study, corroborated by [155], shows that reconfigurable accelerators (e.g., FPGA) are more competitive than dedicated ASICs.

Hence, a hardware accelerator with the potential of executing any application without the complexity of a general purpose core, offers a middle ground between a dedicated ASIC and a complex general purpose core, with less programmability effort than reconfigurable hardware (i.e., FPGA).

We envision a hardware accelerator capable of executing an approximate ISA containing the minimal corpus of instructions used in the main computations of RMS applications, and approximate some of those computations to obtain energy gains. In Chapter 8, we explore the potential of an Approximate Instruction Set Computer (AISC) and implement a proof-of-concept

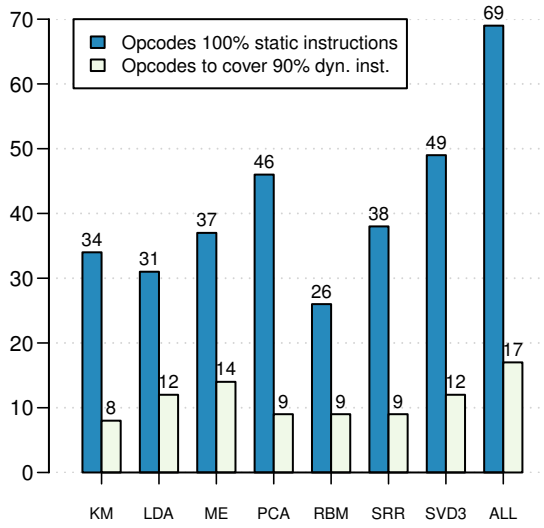


Figure 6.1: Opcodes distribution (native execution).

that approximates the ISA in two directions: *Depth* (e.g., dropping instructions) and *Breadth* (e.g., simplifying instructions).

6.2.1 *ISA Redundancy*

The ISA bridges the gap between software and hardware layers of the system stack. ISA grows with the addition of new extensions in depth (addition of new instructions) and breadth (wider instructions, in the case of CISC machines) directions. This growth increases the complexity of the fetch and decode hardware (front-end), an already power hungry part of the pipeline, and critical for performance, as it feeds the back-end with instructions. Besides, execution units (back-end) and their control policies also become more complex.

Lopes *et al.* perform a chronological analysis of several x86 applications and operating systems and show that 30% of the instructions were rarely used or become unused over time [100].

Following the same philosophy, we performed the following experiment. We executed a Pintool [101] that instruments our applications binaries running on x86 (see Chapter 7) and extracts the static and dynamic instruction mix and opcode distribution. Figure 6.1 collects in a per application basis (observing only the selected kernels), the total number of different static opcodes, and the number of different opcodes that would cover 90% of the dynamic instructions of the applications (Chapter 7 describes all the applications in detail). The last group of bars "ALL" considers all the applications. The number of different opcodes used in our application's kernels is low, and most of the instructions executed correspond to a very small set of different instructions, which reinforces our intuition that these applications could be executed on an AISC, obtaining large energy gains with a limited accuracy loss.

ISA extensions have been the proposed in the context of approximation [54, 75]. In [54], the authors describe an ISA extension to provide approximate

operations and storage. Kamal *et al.* extend the ISA with custom instructions for embedded devices, and allow approximation on those custom instructions by not meeting timing requirements, as the results might be still good enough [75]. Instead of extending the ISA, we explore the possibilities of reducing the ISA complexity, by reducing the set of instructions and/or the size of the operands. Next, we provide an overview of prior work on approximate computing.

6.3 RELATED WORK

Approximate computing is not a novel idea, but it has gained traction in the last years because of the increasing power constraints of devices. Our perceptual limitations provide scope for approximate computing in, for example, visual applications: video and audio encoders reduce the size of audio and movie files without a perceptive quality degradation. In other scenarios, no unique answer exists. For instance, many results may be acceptable as answer of a search query. In fact, an increasingly large amount of RMS applications are intrinsically error-tolerant, and dropping or approximating some computations yields to acceptable results, reducing the energy consumption [35, 96, 148, 158].

Approximate computing is based on the observation that in many scenarios, performing exact computation requires large amount of resources. However, allowing a certain degree of approximation can provide gains in performance and energy, while still achieving acceptable accuracy [54, 110, 133].

The first step of any approximation framework/technique is to find the *approximable* regions or instructions of a given kernel or program (i.e., those that do not compromise safety). This can be done automatically, for example via statistical methods [130], automatic resilience characterization [37], or with a genetic algorithm (compiler) [13]; ensuring the quality of approximable computations through output monitoring [60, 84, 103, 127]; with programming language support for approximate computing (i.e., the programmer selects approximable and non-approximable data), e.g., EnerJ [133], Axilog [163], Rely [25]; or using Open-MP style directives for marking approximable portions [125, 151].

On the *program* side, we find approximation-aware programming languages [25, 133, 143] and libraries [21], most based on probabilistic programming languages [89].

On the *compiler* side, compilers can use approximations inferred or automatically expressed, in addition to approximation-aware analysis; i.e., search for optimizations and/or transformations that maximize the energy savings subject to the accuracy constraints, either by dynamic testing [15, 110] or by reducing the problem to an optimization one and use linear or integer mathematical programmer solvers [111].

On the *architecture* side, we find approximate techniques applied to processor [78], memory [98], and storage [134]. Regarding approximate processor architectures, we could distinguish two implementations: traditional code running on enhanced general purpose processors and accelerators. In the first case, the energy savings are limited by running in a general purpose system. The approximation can be fine-grained [54] or coarse-grained [78]. In the second case, neural-inspired accelerators have been proposed [55].

6.3.1 *Approximate Computing Techniques*

One of the most widely used approximation technique is precision scaling. Many proposals include it as part of their approximation strategy [38, 64, 133, 164, 169]. Rubio-Gonzalez *et al.* point out that for high-performance computing, mixed precision programs might have the same accuracy but compute faster [131]. For example, one operation that requires high accuracy is done with *double* (64 bit), while the rest of the program uses *float* (32 bit). Tong *et al.* also explore precision scaling, and they show that a digit serial multiplier has a linear decrease in energy and latency with operand bit width [150]. In [144], a compiler pass tool also profiles the program to obtain the minimum width needed for different integers. More recent works also point out the potential gains of precision reduction and the need of architectures able to translate lower-precision requirements in high energy gains [66, 115].

Some techniques use code (or loop) perforation, which consists on dropping some iterations of a loop [110]. The rationale behind loop perforation is to drop redundant computations, often manifested as extra loop iterations. Computational patterns such as Monte Carlo simulation, iterative refinement, and search space enumeration are well suited for these techniques [140]. We explore random dropping of computations as a more aggressive code perforation technique.

Finally, it is worth referring to the work from Schkufza *et al.* [136]. Their objective is the automatic generation of different floating point kernels, applying different optimization in an stochastic way. In detail, for each computation kernel, they apply a series of transformations, including changes in the opcodes, the operands, swapping instructions, and even dropping computations. Their goal is to obtain aggressive optimization of high performance applications running on conventional hardware, whilst we focus on emerging RMS applications executing on an AISC-like accelerator.

7 EXPERIMENTAL FRAMEWORK

This chapter presents the designed environment for testing the proposals of Part III of the dissertation, including the experimental set-up, programs under test, and the metrics to quantify the impact of the proposed techniques.

7.1 WORKLOADS

We consider a set of workloads from emerging domains such as Recognition, Mining, and Synthesis (RMS) from Cortex Suite [149]. In some cases, we extend the provided applications with alternative inputs or input combinations that do not fully match the *small*, *medium*, or *large* input suggestions of the benchmark suite, but that are provided by the authors as alternative input sets. For completeness, we add the k-means workload from MineBench [116]. In the rest of this section, we will explain in detail the applications we have considered, as well as their inputs and outputs. Table 7.1 summarizes the workloads and their input parameters.

k-MEANS (KM): KM clustering is a method of vector quantization; it aims to partition n observations into k clusters, in which each observation belongs to the cluster with the nearest mean, using an iterative refinement technique. We use as input 100 entries from the edge features of the Corel Coorporation real image database [116]. The outputs of the workload are the cluster centers (a vector of coordinates) and the cluster assignments (a cluster assignment for each input point). The k-means algorithm has many applications, and it has been successfully used in various topics, including market segmentation, computer vision, geostatistics, astronomy, and agriculture. It is often used as a preprocessing step for other algorithms, for example to find a starting configuration.

LATENT DIRICHLET ALLOCATION (LDA): LDA is a topic modeling algorithm commonly found in natural language processing. A topic model is a type of statistical model for discovering the abstract "topics" that occur in a collection of documents. In LDA, each document may be viewed as a mixture of various topics. Documents are represented as random mixtures over latent topics, where each topic is characterized by a distribution over the words. We run the estimation phase, obtaining as outputs the final estimation model and the Dirichlet parameter α. We estimate α in each iteration and initialize the model randomly. We explore 500 documents with 6907 terms. The stop criteria for the variational inference is 10^{-6} or 20 iterations; the stop criteria for the variational expectation-maximization (EM) is 10^{-4} or 100 iterations.

MOTION ESTIMATION (ME): ME is the process of determining motion vectors that describe the transformation form one 2D image to another, usually adjacent frames in a video sequence. Motion estimation is a key part of video compression as a way to exploit temporal redundancy. The ap-

plication we run combines a block matching algorithm for the super-pixel motion estimation and optical flow for the sub-pixel motion estimation. It takes as input the "Alpaca" dataset from the Multi-Dimensional Signal Processing Research Group (MDSP) from the University of California, Santa Cruz (UCSC)[1]. This input set has 16 frames of 96x128 pixels. The outputs of ME are the motion vectors (1 vector per frame).

PRINCIPAL COMPONENT ANALYSIS (PCA): PCA is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. PCA is one of the most versatile and widely used statistical techniques for feature extraction in multivariate datasets. PCA is mostly used as a tool in exploratory data analysis and for making predictive models. We use correlation analysis and, therefore, the application computes PCA by eigenvalue decomposition of a data correlation matrix. We use as input the "Handwritten Digit Data Set" from the University of California, Irvine (UCI)[2]. This input set contains 1593 instances with 256 attributes. The outputs are the correlation matrix, the eigenvalues, the eigenvectors, and the projections of the row- and column-points on the first three principal components.

RESTRICTED BOLTZMANN MACHINE (RBM): RBM is a generative stochastic artificial neural network that can learn a probability distribution over its set of inputs. RBMs are found in areas such as dimensionality reduction, classification, collaborative filtering, feature learning, and topic modeling. The application we run uses RBM to implement movie suggestions (collaborative filtering) on variants of the Netflix database. We train the network with 100 users and 100 movies for 20 iterations, and then obtain movie suggestions for 100 users. The output of the application is the movie suggestions.

SUPER-RESOLUTION RECONSTRUCTION (SRR): SRR is a technique that enhances the resolution of images. It uses slight sub-pixel variations in the information encoded in a series of low resolution images, to recover a higher resolution image. We use as input the "EIA" dataset from the MDSP Group from UCSC[3]. This input set has 16 frames of 64x64 pixels. The output is the reconstructed image, which has a size of 256x256.

SINGULAR VALUE DECOMPOSITION (SVD3): SVD3 is a factorization of a real or complex matrix, and it is used in many artificial intelligence, signal processing, and computer vision applications. The singular value decomposition of a mxn matrix M is a factorization of the form USV^*, where U (mxm) and V (nxn) are unitary matrices, and S (mxn) is a rectangular diagonal matrix, whose values are the singular values of M. The input is a 500x500 M matrix from KOS Press; the outputs are the U, V, and S matrices.

1 https://users.soe.ucsc.edu/~milanfar/software/sr-datasets.html; last access: February 2016

2 http://archive.ics.uci.edu/ml/datasets.html; last access: February 2016

3 See Footnote 1

Table 7.1: Benchmarks deployed: classification and input parameters.

Application	Domain	Input
K-means (KM)	Clustering	Edge features Corel Corporation DB (100 entries, 18 features)
Latent Dirichlet Allocation (LDA)	Topic Modeling	500 documents, 6097 terms Variational inference: 20 itr./10^{-6} error Variational EM: 100 itr./10^{-5} error Estimate alpha, random initialization
Motion-Estimation (ME)	Computer Vision	"Alpaca" Dataset (16 frames, 96 x 128)
Principal Component Analysis (PCA)	Feature Selection	"Handwritten Digit Dataset" (1593 instances, 256 attributes)
Restricted Boltzmann Machine (RBM)	Deep Learning	Netflix DB (100 train users, 100 test users, 100 movies, 20 training iterations)
Super-Resolution Reconstruction (SRR)	Computer Vision	"EIA" Dataset (16 frames, 64 x 64)
Single Value Decomposition (SVD3)	Feature Selection	KOS Press (500x500 matrix)

7.2 METHODOLOGY

We compiled the applications with GCC 4.8.4 on Ubuntu 14.04 with `-O1` optimization. As we perform manual transformations on the code, high optimization levels hinder the task; we resort to `-O1` for our proof-of-concept and leave for future work more exploration on compiler optimizations. We run our experiments on an Intel Core i5 3210M machine running at 2.5 GHz.

First, we selected the main kernels of each application, focusing on the main computations and avoiding I/O routines. We used Callgrind, a profiling tool that runs on Valgrind [117], and which records the call history among functions in a program's run as a call-graph, obtaining information about the runtime behavior of an application.

Then, to experiment with the approximate ISA, we run the different experiments with a custom Pintool that runs on Pin 2.14 [101]. Pin is a dynamic instrumentation framework for the IA-32 and the x86-64[4] instruction set architectures that enables the creation of dynamic program analysis tools or Pintools. Our Pintool instruments the compiled binary at run time, observing only the selected kernels, and captures the instructions being executed on the fly; then, for each instruction captured, it emulates it (e.g., replacing it by another instruction or instructions), removes it, or executes it natively, depending on the desired behavior of the approximate ISA.

7.2.1 *Evaluation Metrics*

To evaluate the behavior of the selected applications, we consider three different metrics: performance, energy, and accuracy.

4 Our ideas are, in general, ISA-independent, but we provide results for the x86-64 architecture as a use case.

7.2.1.1 *Performance*

As performance metric we use the number of executed instructions and the instruction mix, provided by our Pintool.

We classify the instructions in three categories: *critical*, *integer*, and *floating point*. In the critical instructions group, we include control-flow instructions (e.g., jumps, call/return from functions, comparison), access to the stack (e.g., push/pop), conversion instructions (e.g., from integer to double), etc. We consider these instructions "critical", as approximating or removing them might alter the control flow of the program and prevent successful program termination. For this work, we consider these instructions not suitable for approximation, and we will always execute them natively (no emulation). Integer instructions include all the arithmetic integer instructions; note that in this group, we find also instructions that compute memory addresses. Finally, floating point instructions are those instructions that operate with floating point data. We further divide this group in sizes: 64-bit, 32-bit, and 16-bit[5] operands.

7.2.1.2 *Energy*

We build a first-order approach to quantify the energy savings based on bare-metal measurements for x86 processors [43]. Energy efficiency is often measured by calculating the effective energy per instruction (EPI), but EPI makes the most sense as a metric when it remains constant, independent of IPC. However, real machines expend energy at a certain minimum rate even when idle. Hence, actual EPI becomes a function of IPC, and the effective cost of an instruction decreases with higher IPC, because the fixed costs are shared. This relationship is the result of high static power and complex control logic that is always on, independently of core utilization. The percentage of fixed and variable power depends on the architecture, technology, micro-architecture enhancements, etc. For example, on the Intel Penryl micro-architecture, 56% of the energy consumption is due to fix costs, while on the Haswell micro-architecture the percentage of energy consumption due to fix costs increases to 75%. Power can be modeled as a function of IPC to approximate this overhead:

$$P_t = \alpha * IPC + \beta \tag{7.1}$$

where P_t is the total power, and α and β are constants that model the variable and the fixed power, respectively, and depend on the architecture, technology, frequency, voltage, etc. From this equation, it is clear that the α parameter represents EPI/T_c for a given configuration, being T_c the cycle time. To model the energy consumption, we can multiply each of the terms of the previous equation by execution time ($T_{exec} = T_c * Cycles$, where $Cycles$ represent the total execution cycles):

$$E_t = \alpha' * N_{inst} + \beta * T_{exec} \tag{7.2}$$

where $\alpha' = EPI$.

5 The micro-architecture lacks support for 16-bit floating point values, so we emulate these operations. Please refer to the next chapter for more details.

In the next chapter, we discuss in more detail how we apply this model to our experiments.

7.2.1.3 *Accuracy*

Regarding the applications output accuracy loss, we follow the guidelines from [5] and run the ACCURAX tool with the proper application-specific accuracy loss metric. Table 7.2 collects the different outputs we consider for each of the applications described in the previous section, together with the accuracy loss metric for each of the outputs. Table 7.2 also includes *output randomization* [5] results: for each application output, we compute the error we would obtain with a complete random output with respect to the native outputs. The idea of output randomization is, for a given application and its accuracy metric, to compare the accuracy loss under approximation with the average accuracy loss by a completely random output. A total random output, as it is input-independent, will be likely the close-to-worst-case accuracy loss we could obtain. We generate 10^4 random outputs and collect the average of the distribution of the calculated accuracy loss for each application output. These results will guide the accuracy loss impact discussion of the following chapter.

Table 7.2: Benchmarks deployed: output results and accuracy metrics.

Application	**Output**	**Accuracy Metric**	**Random Output**
K-means (KM)	• Cluster centers • Cluster assignments	• Coordinate displacement • Rel. number differences	• 51.91 • 92.43
Latent Dirichlet Allocation (LDA)	• Estimation model	• ANPS [a]	• 41.83
Motion Estimation (ME)	• Motion vectors (1 per frame)	• Coordinate displacement	• 39.60
Principal Componenet Analysis (PCA)	• Correlation matrix • Row projections • Column projections	• ANPS	• 39.40 • 43.95 • 60.06
Restricted Boltzmann Machine (RBM)	• Suggestions for users (100 x 100 matrix)	• ANPS	• 28.27
Super-resolution Reconstruction (SRR)	• Reconstructed image (256 x 256)	• SSIM [b]	• 49.99
Single Value Decomposition (SVD3)	• Decomposition matrices (U, V, S)	• ANPS	• 83.32 • 55.11 • 46.10

[a] ANPS: Average Noise to Peak Signal

[b] SSIM: Structural Similarity Index

8

AISC: APPROXIMATE INSTRUCTION SET COMPUTER

Three emerging classes of applications, Recognition, Mining, and Synthesis (RMS), rely on iterative, often probabilistic, algorithms in processing massive yet noisy (inaccurate) input data sets. Usually there is no single correct output and valid outputs span a range. Therefore, RMS applications are intrinsically tolerant to errors in computation. In this chapter, we explore how such intrinsic algorithmic error tolerance can help reduce the complexity of the instruction set architecture and thereby improve the energy efficiency of computing without compromising functional completeness.

8.1 INTRODUCTION

The Instruction Set Architecture (ISA) specifies semantic and syntactic characteristics of a *practically* functionally complete set of machine instructions. Modern ISAs are not necessarily *mathematically* functionally complete, but provide sufficient expressiveness for practical algorithms. For software layers, the ISA defines the underlying machine—as capable as the variety of algorithmic tasks the composition of its building blocks, *instructions*, can express. For hardware layers, the ISA rather acts as a behavioral design specification for the machine organization. Accordingly, the ISA governs both the *functional completeness* and *complexity* of a machine design.

Based on the observation that the emerging Recognition, Mining, and Synthesis (RMS) classes of applications [35] can effectively tolerate errors in computation due to iterative (usually probabilistic) algorithms processing massive yet noisy (i.e., inaccurate) input data [96, 148, 158], in this chapter, we explore *how the intrinsic algorithmic error tolerance can help reduce ISA (hence, hardware design) complexity without compromising functional completeness*. Such reduction in complexity often directly translates into higher energy efficiency, i.e., performance gain per unit power consumed.

The end result is an *Approximate Instruction Set Computer (AISC)* that trades computation accuracy for complexity, and thereby energy efficiency. In this context, the complexity reduction in ISA spans two dimensions:

- *Depth*: elimination of relatively more complex and less frequently used instructions.
- *Breadth*: simplification on a per instruction basis.

The combination of these two dimensions, *Breadth + Depth*, replaces complex and less frequently used instructions by a sequence of simpler instructions.

Depending on the instruction mix of the workload, aggressive complexity reduction along any of these dimensions can affect the computation accuracy, and may potentially lead to invalid or unacceptable results. AISC implementations should prevent the latter cases by limiting the complexity reduction such that the anticipated loss in accuracy remains bounded. This represents the key difference between AISC and other ISAs optimized for complexity such as RISC (Reduced Instruction Set Computer) [124], which

do not compromise computation accuracy. When compared to RISC variants, AISC can unlock more opportunities for complexity reduction at the cost of accuracy loss, as long as this accuracy loss remains bounded and acceptable.

AISC facilitates ISA complexity reduction along two dimensions: Depth, Breadth, and its combination, Breadth + Depth. Various techniques apply to realize complexity reduction along each dimension. This study covers a representative, but not necessarily exhaustive, set. Different techniques give rise to different AISC implementations in a rich design space.

Depth encompasses all techniques that orchestrate elimination, i.e., drop, of complex and less frequently used instructions. Such techniques can be of static or dynamic nature. Static variants of instruction dropping techniques in Depth selectively eliminate (static) instructions from the binary—these instructions can never get executed. A collection of such instructions can be permanently eliminated from the ISA. Dynamic variants of instruction dropping techniques in Depth, on the other hand, selectively eliminate dynamic instances of static instructions within the course of execution.

Under Breadth fall all techniques that can reduce the complexity of an instruction. The stellar representatives are techniques which modulate arithmetic precision by reducing operand widths. Well-studied precision scaling variants [38, 64, 66, 115, 131, 133, 144, 150, 164, 169] are directly applicable, so are techniques to simply discard a sequence of the least significant operand bits. Reducing the operand width often enables simplification in the corresponding arithmetic operation, in addition to a more efficient utilization of the available communication bandwidth for data (i.e., operand) transfer.

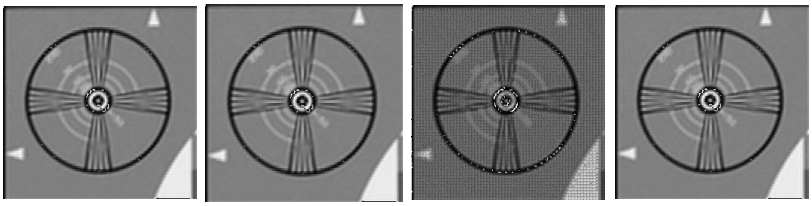
Breadth + Depth embeds all techniques that emulate a complex instruction by using a sequence of simpler instructions. The key difference from Depth comes from the emulation of relatively more complex instructions: Depth techniques drop such instructions without any replacement.

AISC-induced accuracy loss may not be (or not always be) acceptable for many workloads, including the operating system code. At the same time, the tolerance to accuracy loss of an application may change within the course of execution. We envision a heterogeneous many-core platform with a subset of the cores featuring AISC, and the rest, any ISA not compromising accuracy. This platform can then map only applications (or application phases) that can tolerate accuracy loss to AISC cores.

In the rest of this chapter, Section 8.2 details a proof-of-concept AISC implementation and practical limitations. In Section 8.3, we quantify the complexity versus energy efficiency trade-off as induced by AISC. Section 8.4 concludes and summarizes our findings.

8.2 AISC: PROOF-OF-CONCEPT IMPLEMENTATION

We start with a motivating example. Figure 8.1 shows how the (graphic) output of a typical RMS application, SRR (see Table 7.1), changes for representative Depth, Breadth, and Breadth + Depth techniques under AISC. Figure 8.1a captures the output for the baseline for comparison, *Native* execution, which excludes AISC. We observe that the accuracy loss remains barely visible and varies across different techniques. We will next analyze the sources of this diversity.



(a) Native. (b) Depth. (c) Breadth. (d) Breadth + Depth.

Figure 8.1: Graphic output of SRR under representative AISC techniques (b)-(d).

8.2.1 *Depth Techniques*

Depth encompasses all techniques that orchestrate dropping of complex and less frequently used instructions. The key question is how to pick the instructions to drop. A more general version of this question, *which instructions to expose to AISC techniques*, already applies to AISC techniques across all dimensions, but the question becomes more critical for Depth. As the most aggressive in our bag of tricks, Depth can incur significant loss in accuracy. RMS applications can tolerate errors in data-centric phases as opposed to control [40]. Therefore, confining instruction dropping to data-flow can help limit the incurred accuracy loss. A profiler backed compiler pass, possibly exploiting algorithmic information via programmer annotations [133], can determine candidate instructions for dropping.

Depth techniques from the proof-of-concept AISC implementation come in two flavors: *Static-Drop* and *Dynamic-Drop*. Static-Drop directly eliminates static instructions from the binary; Dynamic-Drop eliminates dynamic instances of static instructions within the course of execution on top. For Dynamic-Drop, runtime or hardware [54] support is necessary to drop instructions as a function of evolving runtime conditions. Figure 8.1b captures an example execution outcome under Static-Drop.

8.2.2 *Breadth Techniques*

The proof-of-concept AISC implementation features various Breadth techniques: *DPtoSP*, *DP(SP)toHP*, and *DP(SP)toINT*.

Under the IEEE 754 standard, 32 (64) bits express a single (double) precision floating point number: one bit specifies the *sign*; 8 (11) bits, the *exponent*; and 23 (52) bits the *mantissa*, i.e., the fraction. For example, $(-1)^{sign} \times 2^{exponent-127} \times 1.mantissa$ represents a single-precision floating number.

DPtoSP is a *bit discarding* variant, which omits 32 least-significant bits of the mantissa of each double-precision operand of an instruction, and keeps the exponent intact. DP(SP)toHP comes in two flavors. DPtoHP omits 48 least-significant bits of the mantissa of each double-precision operand of an instruction, and keeps the exponent intact; SPtoHP omits 16 least-significant bits of the mantissa of each single-precision operand of an instruction. Figure 8.1c captures an example execution outcome under DPtoHP. DP(SP)toINT also comes in two flavors. DPtoINT (SPtoINT) replaces double (single) precision instructions with their integer counterparts, by rounding the floating point operand values to the closest integer.

8.2.3 *Breadth + Depth Techniques*

The proof-of-concept AISC design features two representative Breadth + Depth techniques: *MULtoADD* and *DIVtoMUL*.

MULtoADD converts multiplication instructions to a sequence of additions. AISC picks the smaller of the factors as the multiplier, which determines the number of additions. AISC rounds floating point multipliers to the closest integer number.

DIVtoMUL converts division instructions to multiplications. AISC first calculates the reciprocal of the divisor, which next gets multiplied by the dividend to render the end result. In our proof-of-concept implementation based on the x86 ISA, the reciprocal instruction has 12-bit precision. *DIVtoMUL12* uses this instruction. *DIVtoMUL.NR*, on the other hand, relies on one iteration of the Newton-Raphson method [52] to increase the precision of the reciprocal to 23 bits. DIVtoMUL12 can be regarded as an approximate version of DIVtoMUL.NR, eliminating the Newton-Raphson iteration, and hence enforcing a less accurate estimate of the reciprocal (of only 12 bit precision). Figure 8.1d captures an example execution outcome under DIVtoMUL.NR. Algorithm 1 provides an example sequence of instructions to emulate division according to DIVtoMUL.NR, where one reciprocal (`RCPSS`), 3 multiplication (`MULSS`), one addition (`ADDSS`), and one subtraction (`SUBSS`) instruction substitute a division. DIVtoMUL12 omits the iteration of the Newton-Raphson method (lines 3-6) from Algorithm 1. As opposed to DIVtoMUL.NR, DIVtoMUL12 keeps the 12-bit accuracy of the `RCPSS` instruction. Hence, it becomes mathematically equivalent to omitting 11 bits of the mantissa.

Algorithm 1 DIVtoMUL.NR

; Take reciprocal of the divisor: $x_0 = \mathrm{RCP}(\mathrm{divisor})$; 12-bit precision
; Newton-Raphson iteration to increase reciprocal precision to 23 bits:
; $x_1 = x_0 \times (2 - \mathrm{divisor} \times x_0) = 2 \times x_0 - \mathrm{divisor} \times x_0^2$
; Multiply dividend with reciprocal: $\mathrm{result} = \mathrm{dividend} \times x_1$

```
1 MOVSS xmm1, divisor
2 RCPSS xmm0, xmm1        ; x0
3 MULSS xmm1, xmm0        ; divisor × x0
4 MULSS xmm1, xmm0        ; (divisor × x0) × x0
5 ADDSS xmm0, xmm0        ; 2 × x0
6 SUBSS xmm0, xmm1        ; x1 = 2 × x0 − divisor × x0^2
7 MULSS xmm0, dividend    ; result = dividend × x1
```

8.3 EVALUATION

In this section, we analyze the impact of the different proposed AISC techniques in terms of instruction mix and count, energy, and accuracy of the results, based on the methodology depicted in Chapter 7. We first refine the energy model discussed in Chapter 7. Then we explore the proof-of-concept AISC results.

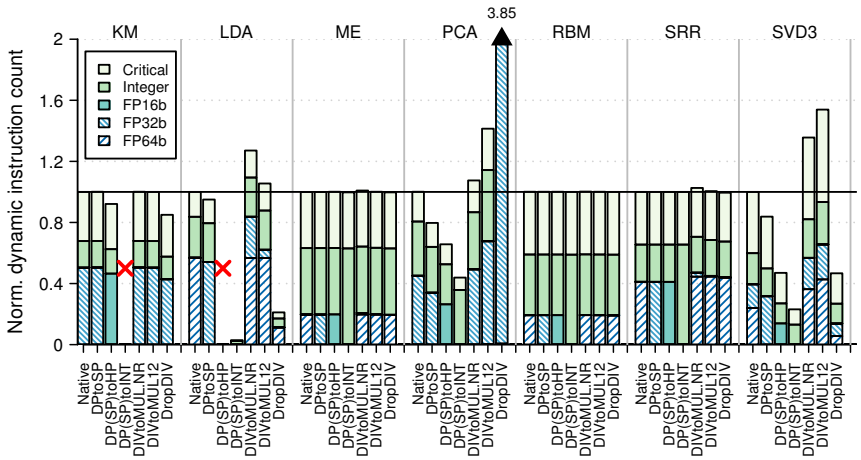


Figure 8.2: Impact on instruction mix and count.

8.3.1 *Evaluation Setup: Energy Modeling*

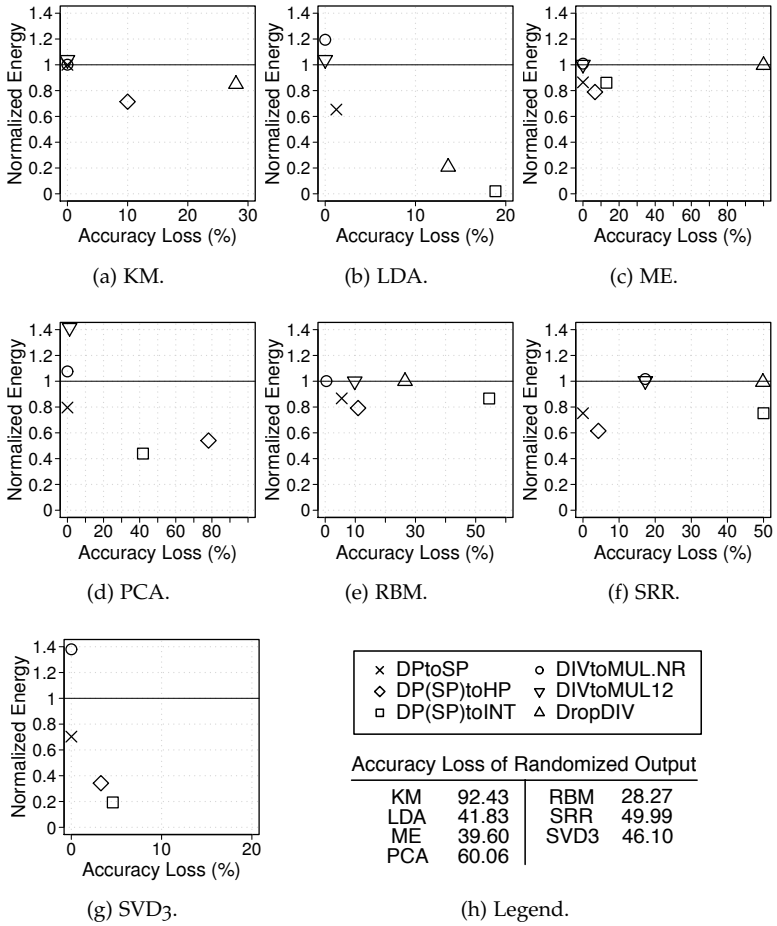
In Chapter 7, we discussed energy modeling as the addition of a fixed and a variable component. The fixed component gathers the fixed costs associated to the execution of an instruction (e.g., clock, front-end, ROB, etc.), independently of the instruction type and IPC, while the variable component captures changes in activity. AISC affects both, the fixed and the variable component: savings in the variable component come from the operand width reduction under Breadth, and instruction dropping under Depth. The fixed component can also decrease if the micro-architecture exploits AISC for a less complex pipeline front-end (fetch + decode) design.

For this work, we do not assume or model any concrete micro-architecture, so we cannot estimate execution time trustingly. Therefore, we model the impact of our proposals on energy in a conservative way, and report the anticipated energy savings in the variable component. The total savings will depend on the percentage of variable energy with respect to the total. Note also that some of the techniques we explored aimed to reduce the complexity of the processor front-end would directly impact in the fixed costs, likely reducing the ratio of fixed/variable energy.

We conservatively assume that the EPI of an integer instruction equals the EPI of a 32-bit floating point arithmetic instruction, and scale the EPI values for 64-bit and 16-bit operations according to [41].

8.3.2 *AISC Proof-of-Concept Evaluation*

Figure 8.2 shows the impact on instruction mix and count, as characterized by a group of bars for each benchmark. The first bar corresponds to the *Native* execution outcome, excluding any AISC technique. The rest of the bars capture execution under Breadth, Breadth + Depth, and Depth. The height of each bar captures the relative change in the instruction count with respect to the native outcome. The stacks in each bar depict the instruction mix, considering three categories: *Critical* indicates control-flow instructions such as accesses to the stack; *Integer*, integer data transfer and arithmetic; *FP*



(a) KM. (b) LDA. (c) ME.

(d) PCA. (e) RBM. (f) SRR.

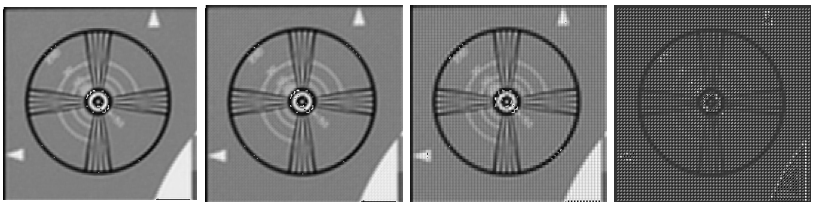
(g) SVD3. (h) Legend.

Figure 8.3: Energy vs. accuracy trade-off.

(Floating Point), floating point data transfer and arithmetic. AISC excludes *Critical* instructions, as approximating or removing them might alter the control flow and prevent successful program termination. The proof-of-concept AISC implementation does not approximate *Integer* instructions to avoid corrupting pointer arithmetic, which may in turn prevent successful termination. We further categorize *FP* instructions by the size of the operands: 64b(it), 32b, and 16b.

To understand the corresponding implications for accuracy, Figure 8.3 shows, on a per benchmark basis, the trade-off space of energy versus accuracy loss. Each point corresponds to the outcome under one AISC technique. We report both energy and accuracy loss relative to the native execution outcome, which excludes AISC techniques. Trade-off spaces do not include non-terminating executions and executions that render more than 40% increase in energy consumption. On the normalized energy axis, any point above 1 is unfeasible. Figure 8.3h shows the output randomization results for each benchmark [5], which serve as a proxy for (close to) worst case accuracy loss.

Next, we examine the proof-of-concept AISC techniques from Section 8.2, which span the dimensions in more detail.



(a) Native. (b) 1 instr. (c) 3 instr. (d) 5 instr.

Figure 8.4: Accuracy of SRR output under Static-Drop.

8.3.2.1 *Breadth Techniques*

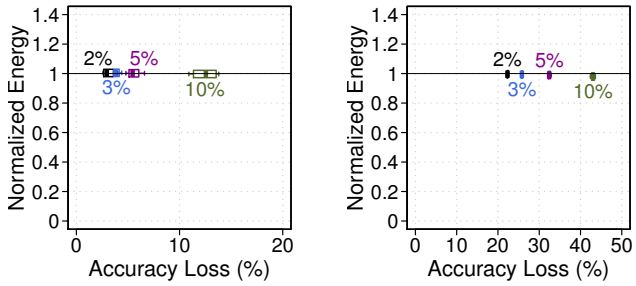
We observe that Breadth—DPtoSP, DP(SP)toHP, DP(SP)toINT—can reduce the instruction count significantly for benchmarks PCA and SVD3 (Figure 8.2). These benchmarks adapt iterative refinement; under bit discarding due to Breadth, they reach the convergence criteria earlier. In this case, only two experiments fail to terminate, marked by a cross in Figure 8.2: DP(SP)toINT in KM and DP(SP)toHP in LDA. Significant changes in instruction count mainly stem from early or late convergence.

Breadth, in terms of DPtoSP and DP(SP)toHP, provides large energy reductions (20% and 37% on average) accompanied by a modest accuracy loss (less than 10%) for most of the applications—except PCA and SRR, where accuracy loss reaches 78.1% and 34.6%, respectively; and LDA where the experiments failed to terminate due to bit discarding. Even the very aggressive DP(SP)toINT works for LDA, ME, and SVD3, where the accuracy loss becomes 18.9%, 13.0%, and 4.6%, respectively. The energy reduction for LDA (98%) and SVD3 (81%) is significant, as the number of iterations to convergence gets drastically reduced: LDA reduces the number of iterations by 97.3%; SVD3, by 97.9%. KM is the only application that does not survive DP(SP)toINT. For the rest of the benchmarks, the accuracy loss becomes comparable to the accuracy of a randomly generated output (which captures close-to-worst-case accuracy loss [5]), therefore, likely not acceptable (Figure 8.3h).

8.3.2.2 *Depth Techniques*

Under Depth, we study two different techniques: Static-Drop and Dynamic-Drop (Section 8.2).

Static-Drop comes in two flavors. First, we selectively remove all the floating point division instructions of the binary (*DropDIV*). This would be equivalent to removing the floating point division instruction from the ISA, without providing any substitute (as oppose to Breadth + Depth techniques). Dropping division instructions does not affect the termination of the experiments, although PCA and SVD3 do not reach convergence. As shown in Figure 8.2, KM and LDA show a significant reduction in the executed instructions under DropDIV, which translates into an energy reduction of 15% and 90%, respectively, with an accuracy loss of 28% and 13.62%. For ME, RBM, and SRR, the instruction count remains practically the same, while the accuracy loss of the outputs reaches the loss of completely random outputs (as indicated Figure 8.3h).



(a) ME. (b) RBM.

Figure 8.5: Energy vs. accuracy trade-off for Dynamic-Drop.

As a more aggressive Static-Drop approach, we randomly delete static (arithmetic) FP instructions. For each static instruction, we base the dropping decision on a pre-defined threshold t. We generate a random number r in the range $[0, 1]$, and drop the static instruction if r remains below t. We experiment with threshold values between 1% and 10%. We observe three distinctive behavior:

1. SVD3 and PCA do not tolerate Static-Drop; experiments either fail to terminate, or render an invalid output/excessive accuracy loss.
2. ME, RBM, and SRR can tolerate dropping, but the outcome highly depends on the instructions dropped. Figure 8.4 illustrates three different SRR outcomes for t=3%: dropping 1 static instruction (Figure 8.4b); 3 static instructions (Figure 8.4c); 5 static instructions (Figure 8.4d); the native output is also shown for comparison (Figure 8.4a). SRR has 477 static instructions, out of which around 80 are FP. The dropped static instructions translate into dropping 16 million dynamic instructions in Figure 8.4b; 245 million in Figure. 8.4c; and 255 million in Figure 8.4d, respectively. The numeric accuracy metric, SSIM [5] becomes 17.1% (Figure 8.4b), 30.2% (Figure 8.4c), and 48.2% (Figure 8.4d). To compensate for the missing instructions, SRR executes additional iterations, increasing the dynamic instruction count.
3. For KM and LDA some experiments fail and some survive with varying accuracy loss.

We next analyze Dynamic-Drop. This time, for each *dynamic* instruction, we base the dropping decision on a pre-defined threshold t. We generate a random number r in the range $[0, 1]$, and drop the dynamic instruction if r remains below t. We experiment with threshold values between 1% and 10%, similar to Static-Drop. In this case, LDA, SRR, and SVD3 fail to terminate, while KM and PCA reach the maximum number of iterations without convergence in most of the experiments. Only ME and RBM survive: Figure 8.5 shows the energy vs. accuracy trade-off for both applications with different dropping thresholds: t = 2%, 3%, 5%, and 10%, respectively. In both cases, the accuracy loss increases with the dropping threshold. ME compensates the dropped instructions with more iterations, which practically translates into no energy gain. RBM, on the other hand, shows a slight (and practically invisible) energy decrease with the number of instructions dropped, but the accuracy loss increases and reaches the output randomization values which capture the close-to-worst-case accuracy loss (Figure 8.3h).

8.3.2.3 *Breadth + Depth Techniques*

Under Breadth + Depth, we observe that MULtoADD (Section 8.2) significantly increases the instruction count and/or prevents successful termination (LDA and SVD3)—except for RBM, where most of the multiplications involve very small operands between 0 and 1. In any case, we did not observe any improvement on the energy vs. accuracy trade-off space, and, therefore, we excluded MULtoADD from Figures 8.2 and 8.3.

DIVtoMUL variants also increase instruction count, although this increase is only significant for LDA, PCA, and SVD3, where more iterations are run to compensate for the precision reduction, as we did not alter the convergence criteria. Breadth + Depth, in terms of DIVtoMUL.NR and DIVtoMUL12, does not show an energy advantage, mostly due to our conservative energy modeling (we assume that all *FP* instructions of a given precision have the same EPI). KM, ME, and SRR have a very small percentage of division operations, accordingly, DIVtoMUL variants have minimal impact. Eliminating the Newton-Raphson iteration under DIVtoMUL.NR only increases the accuracy loss in RBM. PCA and SVD3 experience a significant energy increase under DIVtoMUL12 (when compared to DIVtoMUL.NR) due to the increasing number of iterations until convergence to meet the convergence criteria.

8.4 CONCLUSION AND DISCUSSION

The ISA bridges the gap between software and hardware layers of the system stack. ISA grows with the addition of new extensions in Depth (addition of new instructions) and Breadth (wider instructions, in the case of CISC machines) directions. This growth increases the complexity of the fetch and decode hardware (front-end), an already power hungry part of the pipeline, and critical for performance, as it feeds the back-end with instructions. Besides, execution units (back-end) and their control policies also become more complex.

Conventional ISAs are unaware of the intrinsic error tolerance of many emerging algorithms. AISC turns around this assumption to reduce hardware complexity, and, thereby, energy consumption. To reduce hardware complexity, AISC shrinks the Breadth and Depth of an ISA design, where algorithmic error tolerance helps masking the corresponding loss in accuracy without compromising correctness or functional completeness of the ISA. In RMS workloads, AISC can cut energy 20.6% on average, at around 14.9% accuracy loss.

We find that the energy vs. accuracy trade-off is very sensitive to application convergence characteristics. Therefore, an efficient AISC implementation should carefully examine convergence criteria. For example, under Depth, applications that tolerate instruction dropping either compensate for the missing instructions by executing more to meet the convergence criteria, or exhibit a very large accuracy loss.

The presented proof-of-concept implementation does not track data dependencies beyond instruction boundaries. Operand width reduction under Breadth or Breadth + Depth is confined to the input and output operands of the respective instructions. Transitively adjusting the precision of the variables which depend on or determine such input and output operands may substantially increase energy savings.

Part IV

CONCLUSION

9 CONCLUSIONS AND FUTURE WORK

This chapter highlights the main conclusions of this dissertation and points out future research directions based on the main findings herein presented.

This dissertation explores new approaches to improve the energy efficiency of future systems in the dark silicon era. We explore sources of natural on-chip redundancy and take advantage of them in the context of computing at voltages near the threshold and approximate computing.

9.1 FAULTY LAST-LEVEL SRAM CACHES AT NEAR-THRESHOLD VOLTAGES

Insufficient voltage reduction in ultra-deep technologies jeopardizes the benefits that scaling has been providing to the processor industry in the last decades. Within the processor, SRAM cells limit the minimum voltage because at lower voltages, manufacturing induced parameter variations limit the correct functionality of SRAM structures.

The stochastic nature of parameter variations makes some cells more prone to errors than others when voltage is lowered. To cope with this problem, block disabling techniques enable voltage reductions by disconnecting cells that do not fulfill the stability requirements, improving energy efficiency. However, inclusive large last-level caches, LLCs, implementing block disabling experiment a substantial reduction in associativity and capacity, which manifest in two specific drawbacks: the number of inclusion victims in private L1 caches increases and the MPKI figures grow, increasing main memory energy consumption.

In Part II of this dissertation, we propose a technique to improve block disabling in CMPs leveraging current coherence mechanisms: we call this technique Block Disabling with Operational Tags (BDOT). BDOT relies on the use of robust cells to implement the LLC tag array, enabling some cache blocks to be only in private levels by simply tracking their tags. With regard to inclusion victims, the LLC associativity is fully restored, but to unlock the benefits of BDOT, we need to implement the proper cache management policy to take into account the different nature of LLC entries. Hence, we implement on top of BDOT a cache management policy with two goals: protect blocks that are going to be used in the future and maximize the on-chip content. Our BDOT-based cache management policy is able to reduce MPKI up to 37.3% and 54.2% for multiprogrammed and parallel workloads, respectively, compared to block disabling. Those improvements translate into performance improvements of 13% and 34.6%, respectively.

This approach, as well as other existing micro-architectural approaches to increasing cache reliability, focus on enabling and combining the valid cells, thereby reducing the available cache capacity. Although our fault-aware cache management policy obtains great improvements over block disabling with a very low overhead, for programs with large footprints and/or working sets, the extent of performance degradation is still substantial. Depart-

ing from these approaches, we propose Concertina, an LLC that compresses cache blocks to reduce their size in order that they can fit into entries with non-functional cells. Concertina ensures the use of 100% of the LLC capacity through a low overhead insertion/replacement policy that combines block compressibility and fault awareness to enable smart allocation of blocks to cache entries. We explore several LLC implementations with cells that trade off area and power for reliability, as well as several Concertina designs. A lower-overhead design with a modest 5.4% and 5.9% increase in area and leakage, respectively, achieves a minimal performance degradation within 2%, with respect to an unrealistic defect-free LLC.

9.2 APPROXIMATE COMPUTING AT THE ISA LAYER

Approximate computing has gained much traction in the last years due to the emergence of intrinsically error-tolerant applications, which rely on iterative algorithms to process noisy inputs, and whose outputs are not required to be unique ("acceptable" rather than precise outputs). Conventional ISAs are unaware of the intrinsic noise tolerance of these emerging algorithms. In Part III of this dissertation, we propose AISC (Approximate Instruction Set Computer), a set of techniques that turn around this assumption to reduce hardware complexity, and, hence, energy consumption, shrinking the ISA in several dimensions: Breadth (e.g., operand bit discarding), Depth (e.g., dropping instructions), and the combination of both Breadth + Depth (e.g., substituting complex instructions with simpler ones). In RMS (Recognition, Mining, and Synthesis) workloads, AISC can cut energy 20.6% on average, at around 14.9% accuracy loss. We find that the energy/accuracy trade-off is very sensitive to converge characteristics, therefore, an efficient AISC implementation should carefully examine convergence criteria.

The presented proof-of-concept implementation does not track data dependencies beyond instruction boundaries. Operand width reduction under Breadth or Breadth + Depth is confined to the inputs and outputs of the respective instructions. Transitively adjusting the precision of the variables which depend on or determine the inputs and outputs may substantially increase energy savings.

9.3 FUTURE WORK

As technology advances, power becomes a first-order design constraint that imposes severe restrictions on future chips. This dissertation tackles power consumption from two perspectives: near-threshold voltage computing and approximate computing. This work opens doors to many more research ideas, and we anticipate some:

- BDOT is implemented on top of an inclusive cache hierarchy; exclusive and non-inclusive/non-exclusive hierarchies expose different trade-offs, which could be as well exploited for efficient near-threshold voltage execution. For example, we could explore decouple tag/data LLC designs to obtain higher flexibility in the allocation of cache blocks.
- Concertina implements a simple yet efficient compression technique (null subblock compression). Other compression techniques with the

same design objectives (high coverage, low compression ratio) might be implemented.

- BDOT and Concertina lend themselves well to mutual integration: adding a reuse- and fault-aware management policy would likely improve the performance and, therefore, reduce the energy consumption of Concertina.
- So far, the computer architecture community has collected low-hanging fruits on the approximate computing paradigm, and we anticipate a shift towards more approximate applications and designs in the near future. With respect to our concrete proposals, we made a preliminary analysis of the potential of an approximate ISA. Other techniques can be integrated in our framework. For example, on the Depth shrinkage techniques side, only random dropping has been examined. A smarter dropping mechanism might unlock the energy potential we envision.

Finally, approximate computing and near-threshold voltage computing philosophies get along very well; low-voltage SRAM caches might be used for approximate storage, and intrinsically fault-tolerant applications might run in hardware prone to errors. However, the main limitation of approximate computing, *which data/code is suitable for approximation and which is not*, posses a major challenge to the widespread of this paradigm in commercial systems.

BIBLIOGRAPHY

[1] J. Abella, J. Carretero, P. Chaparro, X. Vera, and A. González. "Low Vccmin Fault-tolerant Cache with Highly Predictable Performance." In: *42nd Annual IEEE/ACM Int. Symp. on Microarchitecture*. 2009, pp. 111–121. DOI: 10.1145/1669112.1669128 (cit. on p. 17).

[2] A. Agarwal, B. Paul, H. Mahmoodi, A. Datta, and K. Roy. "A Process-tolerant Cache Architecture for Improved Yield in Nanoscale Technologies." In: *IEEE Tran. on Very Large Scale Integration (VLSI) Systems* 13.1 (Jan. 2005), pp. 27–38. DOI: 10.1109/TVLSI.2004.840407 (cit. on pp. 15, 24).

[3] A. Agarwal, B. Paul, S. Mukhopadhyay, and K. Roy. "Process Variation in Embedded Memories: Failure Analysis and Variation Aware Architecture." In: *IEEE Journal of Solid-State Circuits* 40.9 (Sept. 2005), pp. 1804–1814. DOI: 10.1109/JSSC.2005.852159 (cit. on p. 14).

[4] M. Agostinelli et al. "Erratic Fluctuations of SRAM Cache Vmin at the 90nm Process Technology Node." In: *IEEE Int. Electron Devices Meeting Technical Digest*. 2005, pp. 655–658. DOI: 10.1109/IEDM.2005.1609436 (cit. on p. 14).

[5] I. Akturk, K. Khatamifard, and U. R. Karpuzcu. "On Quantification of Accuracy Loss in Approximate Computing." In: *12th Annual Workshop on Duplicating, Deconstructing and Debunking*. 2015 (cit. on pp. 77, 84–86).

[6] A. Alameldeen, Z. Chishti, C. Wilkerson, W. Wu, and S.-L. Lu. "Adaptive Cache Design to Enable Reliable Low-Voltage Operation." In: *IEEE Trans. on Computers* 60.1 (Jan. 2011), pp. 50–63. DOI: 10.1109/TC.2010.207 (cit. on p. 16).

[7] A. Alameldeen, I. Wagner, Z. Chishti, W. Wu, C. Wilkerson, and S.-L. Lu. "Energy-efficient Cache Design Using Variable-strength Error-correcting Codes." In: *38th Annual Int. Symp. on Computer Architecture*. 2011, pp. 461–471. DOI: 10.1145/2000064.2000118 (cit. on pp. 45, 50).

[8] A. R. Alameldeen and D. A. Wood. "Adaptive Cache Compression for High-performance Processors." In: *31st Annual Int. Symp. on Computer Architecture*. 2004, pp. 212–223. DOI: 10.1145/1028176.1006719 (cit. on pp. 18, 45, 52).

[9] J. Albericio, P. Ibáñez, V. Viñals, and J. M. Llabería. "Exploiting Reuse Locality on Inclusive Shared Last-level Caches." In: *ACM Trans. on Architecture and Code Optimization* 9.4 (Jan. 2013), 38:1–38:19. DOI: 10.1145/2400682.2400697 (cit. on pp. 18, 31).

[10] J. Albericio, P. Ibáñez, V. Viñals, and J. M. Llabería. "The Reuse Cache: Downsizing the Shared Last-level Cache." In: *46th Annual IEEE/ACM Int. Symp. on Microarchitecture*. 2013, pp. 310–321. DOI: 10.1145/2540708.2540735 (cit. on p. 24).

[11] R. S. Amant, A. Yazdanbakhsh, J. Park, B. Thwaites, H. Esmaeilzadeh, A. Hassibi, L. Ceze, and D. Burger. "General-purpose Code Acceleration with Limited-precision Analog Computation." In: *41st Annual Int. Symp. on Computer Architecture*. 2014, pp. 505–516. DOI: 10.1109/ISCA.2014.6853213 (cit. on p. 68).

[12] A. Ansari, S. Feng, S. Gupta, and S. Mahlke. "Archipelago: A Polymorphic Cache Design for Enabling Robust Near-threshold Operation." In: *IEEE 17th Int. Symp. on High Performance Computer Architecture*. 2011, pp. 539–550. DOI: 10.1109/HPCA.2011.5749758 (cit. on pp. 17, 21, 30, 39, 50, 60).

[13] J. Ansel, Y. L. Wong, C. Chan, M. Olszewski, A. Edelman, and S. Amarasinghe. "Language and Compiler Support for Auto-tuning Variable-accuracy Algorithms." In: *9th Annual IEEE/ACM Int. Symp. on Code Generation and Optimization*. 2011, pp. 85–96. DOI: http://dl.acm.org/citation.cfm?id=2190025.2190056 (cit. on p. 70).

[14] S. Baek, H. Lee, C. Nicopoulos, J. Lee, and J. Kim. "Size-aware Cache Management for Compressed Cache Architectures." In: *IEEE Trans. on Computers* 64.8 (Aug. 2015), pp. 2337–2352. DOI: 10.1109/TC.2014.2360518 (cit. on pp. 18, 60).

[15] W. Baek and T. M. Chilimbi. "Green: A Framework for Supporting Energy-conscious Programming Using Controlled Approximation." In: *31st ACM SIGPLAN Conf. on Programming Language Design and Implementation*. 2010, pp. 198–209. DOI: 10.1145/1806596.1806620 (cit. on p. 70).

[16] J. L. Baer and W. Wang. "On the Inclusion Properties for Multi-level Cache Hierarchies." In: *15th Annual Int. Symp. on Computer Architecture*. 1988, pp. 73–80. DOI: 10.1145/633625.52409 (cit. on pp. 28, 29).

[17] A. Banaiyanmofrad, H. Homayoun, and N. Dutt. "Using a Flexible Fault-Tolerant Cache to Improve Reliability for Ultra Low Voltage Operation." In: *ACM Trans. on Embedded Computing Systems* 14.2 (Feb. 2015), 32:1–32:24. DOI: 10.1145/2629566 (cit. on p. 17).

[18] A. J. Bhavnagarwala, X. Tang, and J. D. Meindl. "The Impact of Intrinsic Device Fluctuations on CMOS SRAM Cell Stability." In: *IEEE Journal of Solid-State Circuits* 36.4 (Apr. 2001), pp. 658–665. DOI: 10.1109/4.913744 (cit. on pp. 13, 14, 27).

[19] C. Bienia, S. Kumar, J. P. Singh, and K. Li. "The PARSEC Benchmark Suite: Characterization and Architectural Implications." In: *17th Int. Conf. on Parallel Architectures and Compilation Techniques*. 2008, pp. 72–81. DOI: http://doi.acm.org/10.1145/1454115.1454128 (cit. on p. 23).

[20] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De. "Parameter Variations and Impact on Circuits and Microarchitecture." In: *40th ACM/EDAC/IEEE Design Automation Conf.* 2003, pp. 338–342. DOI: 10.1109/DAC.2003.1219020 (cit. on p. 12).

[21] J. Bornholt, T. Mytkowicz, and K. S. McKinley. "Uncertain<T>: A First-order Type for Uncertain Data." In: *19th Int. Conf. on Architectural Support for Programming Languages and Operating Systems*. Salt Lake City, Utah, USA, 2014, pp. 51–66. DOI: 10.1145/2541940.2541958 (cit. on pp. 68, 70).

[22] D. Bouvier, B. Cohen, W. Fry, S. Godey, and M. Mantor. "Kabini: An AMD Accelerated Processing Unit System on a Chip." In: *IEEE Micro* 34.2 (Mar. 2014), pp. 22–33. DOI: 10.1109/MM.2014.3 (cit. on p. 5).

[23] B. H. Calhoun, A. Wang, and A. Chandrakasan. "Modeling and Sizing for Minimum Energy Operation in Subthreshold Circuits." In: *IEEE Journal of Solid-State Circuits* 40.9 (Sept. 2005), pp. 1778–1786. DOI: 10.1109/JSSC.2005.852162 (cit. on p. 11).

[24] B. Calhoun and A. Chandrakasan. "A 256-kb 65nm Sub-threshold SRAM Design for Ultra-low-voltage Operation." In: *IEEE Journal of Solid-State Circuits* 42.3 (Mar. 2007), pp. 680–688. DOI: 10.1109/JSSC.2006.891726 (cit. on pp. 14, 15).

[25] M. Carbin, S. Misailovic, and M. C. Rinard. "Verifying Quantitative Reliability for Programs That Execute on Unreliable Hardware." In: *ACM SIGPLAN Int. Conf. on Object Oriented Programming Systems Languages & Applications*. 2013, pp. 33–52. DOI: 10.1145/2509136.2509546 (cit. on p. 70).

[26] J. Chang, M. Huang, J. Shoemaker, J. Benoit, S.-L. Chen, W. Chen, S. Chiu, R. Ganesan, G. Leong, V. Lukka, S. Rusu, and D. Srivastava. "The 65-nm 16-MB Shared On-Die L3 Cache for the Dual-Core Intel Xeon Processor 7100 Series." In: *IEEE Journal of Solid-State Circuits* 42.4 (Apr. 2007), pp. 846–852. DOI: 10.1109/JSSC.2007.892185 (cit. on pp. 13, 15, 16, 27, 28).

[27] L. Chang, D. Fried, J. Hergenrother, J. Sleight, R. Dennard, R. Montoye, L. Sekaric, S. McNab, A. Topol, C. Adams, K. Guarini, and W. Haensch. "Stable SRAM Cell Design for the 32 nm Node and Beyond." In: *Symp. on VLSI Technology Digest of Technical Papers*. 2005, pp. 128–129. DOI: 10.1109/.2005.1469239 (cit. on p. 14).

[28] L. Chang, Y. Nakamura, R. Montoye, J. Sawada, A. Martin, K. Kinoshita, F. Gebara, K. Agarwal, D. Acharyya, W. Haensch, K. Hosokawa, and D. Jamsek. "A 5.3GHz 8T-SRAM with Operation Down to 0.41V in 65nm CMOS." In: *IEEE Symp. on VLSI Circuits*. 2007, pp. 252–253. DOI: 10.1109/VLSIC.2007.4342739 (cit. on p. 50).

[29] L. Chang, R. Montoye, Y. Nakamura, K. Batson, R. Eickemeyer, R. Dennard, W. Haensch, and D. Jamsek. "An 8T-SRAM for Variability Tolerance and Low-Voltage Operation in High-performance Caches." In: *IEEE Journal of Solid-State Circuits* 43.4 (Apr. 2008), pp. 956–963. DOI: 10.1109/JSSC.2007.917509 (cit. on pp. 14, 15, 30).

[30] L. Chang, D. J. Frank, R. K. Montoye, S. J. Koester, B. L. Ji, P. W. Coteus, R. H. Dennard, and W. Haensch. "Practical Strategies for Power-Efficient Computing Technologies." In: *Proc. of the IEEE* 98.2 (Feb. 2010), pp. 215–236. DOI: 10.1109/JPROC.2009.2035451 (cit. on pp. 5, 12).

[31] M. Chaudhuri, J. Gaur, N. Bashyam, S. Subramoney, and J. Nuzman. "Introducing Hierarchy-awareness in Replacement and Bypass Algorithms for Last-level Caches." In: *21st Int. Conf. on Parallel Architectures and Compilation Techniques*. 2012, pp. 293–304. DOI: 10.1145/2370816.2370860 (cit. on p. 31).

[32] G. Chen, D Sylvester, D. Blaauw, and T. Mudge. "Yield-Driven Near-threshold SRAM Design." In: *IEEE Trans. on Very Large Scale Integration (VLSI) Systems* 18.11 (Nov. 2010), pp. 1590–1598. DOI: 10.1109/TVLSI.2009.2025766 (cit. on pp. 14, 46).

[33] L. Chen, Y. Cao, and Z. Zhang. "Free ECC: An Efficient Error Protection for Compressed Last-level Caches." In: *IEEE 31st Int. Conf. on Computer Design*. 2013, pp. 278–285. DOI: 10.1109/ICCD.2013.6657054 (cit. on pp. 16, 45).

[34] X. Chen, L. Yang, R. P. Dick, L. Shang, and H. Lekatsas. "C-Pack: A High-Performance Microprocessor Cache Compression Algorithm." In: *IEEE Trans. on Very Large Scale Integration (VLSI) Systems* 18.8 (Aug. 2010), pp. 1196–1208. DOI: 10.1109/TVLSI.2009.2020989 (cit. on pp. 18, 45).

[35] Y. K. Chen, J. Chhugani, P. Dubey, C. J. Hughes, D. Kim, S. Kumar, V. W. Lee, A. D. Nguyen, and M. Smelyanskiy. "Convergence of Recognition, Mining, and Synthesis Workloads and Its Implications." In: *Proc. of the IEEE* 96.5 (May 2008), pp. 790–807. DOI: 10.1109/JPROC.2008.917729 (cit. on pp. 5, 67, 70, 79).

[36] L. Cheng, P. Gupta, C. J. Spanos, K. Qian, and L. He. "Physically Justifiable Die-level Modeling of Spatial Variation in View of Systematic Across Wafer Variability." In: *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* 30.3 (Mar. 2011), pp. 388–401. DOI: 10.1109/TCAD.2010.2089568 (cit. on pp. 15, 24).

[37] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan. "Analysis and Characterization of Inherent Application Resilience for Approximate Computing." In: *50th ACM/EDAC/IEEE Design Automation Conf.* 2013, pp. 1–9. DOI: 10.1145/2463209.2488873 (cit. on pp. 5, 67, 70).

[38] V. K. Chippa, D. Mohapatra, K. Roy, S. T. Chakradhar, and A. Raghunathan. "Scalable Effort Hardware Design." In: *IEEE Trans. on Very Large Scale Integration (VLSI) Systems* 22.9 (Sept. 2014), pp. 2004–2016. DOI: 10.1109/TVLSI.2013.2276759 (cit. on pp. 68, 71, 80).

[39] Z. Chishti, A. R. Alameldeen, C. Wilkerson, W. Wu, and S.-L. Lu. "Improving Cache Lifetime Reliability at Ultra-low Voltages." In: *42nd Annual IEEE/ACM Int. Symp. on Microarchitecture*. 2009, pp. 89–99. DOI: 10.1145/1669112.1669126 (cit. on pp. 16, 60).

[40] H. K. Cho and S. Mahlke. "Dynamic Acceleration of Multithreaded Program Critical Paths in Near-threshold Systems." In: *45th Annual IEEE/ACM Int. Symp. on Microarchitecture Workshops*. 2012, pp. 63–67. DOI: 10.1109/MICROW.2012.18 (cit. on p. 81).

[41] J. W. Choi, D. Bedard, R. Fowler, and R. Vuduc. "A Roofline Model of Energy." In: *IEEE 27th Int. Symp. on Parallel Distributed Processing*. 2013, pp. 661–672. DOI: 10.1109/IPDPS.2013.77 (cit. on p. 83).

[42] E. Chung, P. Milder, J. Hoe, and K. Mai. "Single-chip Heterogeneous Computing: Does the Future Include Custom Logic, FPGAs, and GPGPUs?" In: *43rd Annual IEEE/ACM Int. Symp. on Microarchitecture*. Dec. 2010, pp. 225–236. DOI: 10.1109/MICRO.2010.36 (cit. on p. 68).

[43] K. Czechowski, V. W. Lee, E. Grochowski, R. Ronen, R. Singhal, R. Vuduc, and P. Dubey. "Improving the Energy Efficiency of Big Cores." In: *41st Annual Int. Symp. on Computer Architecture*. 2014, pp. 493–504. DOI: 10.1145/2678373.2665743 (cit. on p. 76).

[44] S. Damaraju, V. George, S. Jahagirdar, T. Khondker, R. Milstrey, S. Sarkar, S. Siers, I. Stolero, and A. Subbiah. "A 22nm IA multi-CPU and GPU System-on-Chip." In: *IEEE Int. Solid-State Circuits Conf. Digest of Technical Papers*. 2012, pp. 56–57. DOI: 10.1109/ISSCC.2012.6176876 (cit. on p. 16).

[45] M. Demler. "Mali-G71 Enables Coherent Computing." In: *Mobile Chip Report* (May 2016) (cit. on p. 5).

[46] R. Dennard, V. Rideout, E. Bassous, and A. LeBlanc. "Design of Ion-implanted MOSFET's with Very Small Physical Dimensions." In: *IEEE Journal of Solid-State Circuits* 9.5 (Oct. 1974), pp. 256–268. DOI: 10.1109/JSSC.1974.1050511 (cit. on p. 3).

[47] B. S. Doyle, S. Datta, M. Doczy, S. Hareland, B. Jin, J. Kavalieros, T. Linton, A. Murthy, R. Rios, and R. Chau. "High Performance Fully-depleted Tri-gate CMOS Transistors." In: *IEEE Electron Device Letters* 24.4 (Apr. 2003), pp. 263–265. DOI: 10.1109/LED.2003.810888 (cit. on p. 3).

[48] R. G. Dreslinski et al. "Centip3De: A 64-Core, 3D Stacked Near-threshold System." In: *IEEE Micro* 33.2 (Mar. 2013), pp. 8–16. DOI: 10.1109/MM.2013.4 (cit. on p. 11).

[49] R. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge. "Near-threshold Computing: Reclaiming Moore's Law Through Energy Efficient Integrated Circuits." In: *Proc. of the IEEE* 98.2 (Feb. 2010), pp. 253–266. DOI: 10.1109/JPROC.2009.2034764 (cit. on pp. 5, 11, 12, 21).

[50] Z. Du, K. Palem, A. Lingamneni, O. Temam, Y. Chen, and C. Wu. "Leveraging the Error Resilience of Machine-learning Applications for Designing Highly Energy Efficient Accelerators." In: *19th Asia and South Pacific Design Automation Conf.* 2014, pp. 201–206. DOI: 10.1109/ASPDAC.2014.6742890 (cit. on p. 68).

[51] J. Dusser, T. Piquet, and A. Seznec. "Zero-content Augmented Caches." In: *23rd Int. Conf. on Supercomputing*. 2009, pp. 46–55. DOI: 10.1145/1542275.1542288 (cit. on pp. 18, 48).

[52] M. Ercegovac and T. Lang. *Digital Arithmetic*. Morgan Kaufmann Series in Computer Architecture and Design. Morgan Kaufmann, 2004 (cit. on p. 82).

[53] H. Esmaeilzadeh, E. Blem, R. Amant, K. Sankaralingam, and D. Burger. "Dark Silicon and the End of Multicore Scaling." In: *38th Annual Int. Symp. on Computer Architecture*. 2011, pp. 365–376. DOI: 10.1145/2024723.2000108 (cit. on p. 4).

[54] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger. "Architecture Support for Disciplined Approximate Programming." In: *17th Int. Conf. on Architectural Support for Programming Languages and Operating Systems*. 2012, pp. 301–312. DOI: 10.1145/2150976.2151008 (cit. on pp. 69, 70, 81).

[55] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger. "Neural Acceleration for General-Purpose Approximate Programs." In: *45th Annual IEEE/ACM Int. Symp. on Microarchitecture*. 2012, pp. 449–460. DOI: 10.1109/MICRO.2012.48 (cit. on p. 70).

[56] H. Ghasemi, S. Draper, and N. S. Kim. "Low-voltage On-chip Cache Architecture Using Heterogeneous Cell Sizes for High-performance Processors." In: *IEEE 17th Int. Symp. on High Performance Computer Architecture*. 2011, pp. 38–49. DOI: 10.1109/HPCA.2011.5749715 (cit. on pp. 16, 38).

[57] R. González-Alberquilla. "Memory Access Management in High Performance Computers." PhD thesis. Universidad Complutense de Madrid, 2013 (cit. on p. 18).

[58] M. Gottscho, A. BanaiyanMofrad, N. Dutt, A. Nicolau, and P. Gupta. "DPCS: Dynamic Power/Capacity Scaling for SRAM Caches in the Nanoscale Era." In: *ACM Trans. on Architecture and Code Optimization* 12.3 (Aug. 2015), 27:1–27:26. DOI: 10.1145/2792982 (cit. on p. 42).

[59] P. Greenhalgh. "Big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7." In: *ARM White paper* (2011), pp. 1–8 (cit. on p. 5).

[60] B. Grigorian and G. Reinman. "Dynamically Adaptive and Reliable Approximate Computing Using Light-weight Error Analysis." In: *NASA/ESA Conf. on Adaptive Hardware and Systems*. 2014, pp. 248–255. DOI: 10.1109/AHS.2014.6880184 (cit. on p. 70).

[61] J. L. Henning. "SPEC CPU2006 Benchmark Descriptions." In: *SIGARCH Comput. Archit. News* 34.4 (Sept. 2006), pp. 1–17. DOI: 10.1145/1186736.1186737 (cit. on p. 22).

[62] S. Herbert and D. Marculescu. "Analysis of Dynamic Voltage/Frequency Scaling in Chip-multiprocessors." In: *ACM/IEEE Int. Symp. on Low Power Electronics and Design*. 2007, pp. 38–43. DOI: 10.1145/1283780.1283790 (cit. on p. 4).

[63] D. Hisamoto. "FD/DG-SOI MOSFET-a Viable Approach to Overcoming the Device Scaling Limit." In: *IEEE Int. Electron Devices Meeting Technical Digest*. 2001, pp. 19.3.1–19.3.4. DOI: 10.1109/IEDM.2001.979528 (cit. on p. 3).

[64] C.-C. Hsiao, S.-L. Chu, and C.-Y. Chen. "Energy-aware Hybrid Precision Selection Framework for Mobile GPUs." In: *Computures and Graphics* 37.5 (Aug. 2013), pp. 431–444. DOI: 10.1016/j.cag.2013.03.003 (cit. on pp. 71, 80).

[65] Intel. *Intel Turbo Boost Technology in Intel Core Microarchitecture (Nehalem) Based Processors*. White paper. Intel Corporation, 2008 (cit. on p. 11).

[66] A. Jain, P. Hill, M. Laurenzano, M. Haque, M. Khan, S. Mahlke, L. Tang, and J. Mars. "CPSA: Compute Precisely Store Approximately." In: *Workshop on Approximate Computing Across the Stack*. 2016 (cit. on pp. 71, 80).

[67] S. Jain et al. "A 280mV-to-1.2V Wide-operating-range IA-32 Processor in 32nm CMOS." In: *IEEE Int. Solid-State Circuits Conf. Digest of Technical Papers*. 2012, pp. 66–68. DOI: 10.1109/ISSCC.2012.6176932 (cit. on p. 11).

[68] A. Jaleel, E. Borch, M. Bhandaru, S. C. Steely Jr., and J. Emer. "Achieving Non-inclusive Cache Performance with Inclusive Caches: Temporal Locality Aware (TLA) Cache Management Policies." In: *43rd Annual IEEE/ACM Int. Symp. on Microarchitecture*. 2010, pp. 151–162. DOI: 10.1109/MICRO.2010.52 (cit. on pp. 18, 22, 28, 32).

[69] A. Jaleel, K. B. Theobald, S. C. Steely Jr., and J. Emer. "High Performance Cache Replacement Using Re-reference Interval Prediction (RRIP)." In: *37th Annual Int. Symp. on Computer Architecture*. 2010, pp. 60–71. DOI: 10.1145/1816038.1815971 (cit. on p. 18).

[70] N. James, P. Restle, J. Friedrich, B. Huott, and B. McCredie. "Comparison of Split-Versus Connected-Core Supplies in the POWER6 Microprocessor." In: *IEEE Int. Solid-State Circuits Conf. Digest of Technical Papers*. 2007, pp. 298–604. DOI: 10.1109/ISSCC.2007.373412 (cit. on p. 12).

[71] J. Jeddeloh and B. Keeth. "Hybrid Memory Cube New DRAM Architecture Increases Density and Performance." In: *Symp. on VLSI Technology*. 2012, pp. 87–88. DOI: 10.1109/VLSIT.2012.6242474 (cit. on p. 5).

[72] L. K. John and L. Eeckhout. *Performance Evaluation and Benchmarking*. CRC Press, 2005 (cit. on p. 24).

[73] C. Johns, J. Kahle, and P. Liue. "Implementation of an LRU and MRU Algorithm in a Partitioned Cache." Pat. US 6,931,493 B2. 2005 (cit. on p. 54).

[74] A. B. Kahng and S. Kang. "Accuracy-configurable Adder for Approximate Arithmetic Designs." In: *49th ACM/EDAC/IEEE Design Automation Conf.* 2012, pp. 820–825. DOI: 10.1145/2228360.2228509 (cit. on p. 68).

[75] M. Kamal, A. Ghasemazar, A. Afzali-Kusha, and M. Pedram. "Improving Efficiency of Extensible Processors by Using Approximate Custom Instructions." In: *Design, Automation & Test in Europe Conf. Exhibition*. 2014, pp. 1–4. DOI: 10.7873/DATE.2014.238 (cit. on pp. 69, 70).

[76] D. Kanter. "Phytium Aims for Chinese Servers." In: *Microprocessor Report* (2015) (cit. on p. 21).

[77] E. Karl, Y. Wang, Y.-G. Ng, Z. Guo, F. Hamzaoglu, U. Bhattacharya, K. Zhang, K. Mistry, and M. Bohr. "A 4.6GHz 162Mb SRAM Design in 22nm Tri-gate CMOS Technology with Integrated Active Vmin-enhancing Assist Circuitry." In: *IEEE Int. Solid-State Circuits Conf. Digest of Technical Papers*. 2012, pp. 230–232. DOI: 10.1109/ISSCC.2012.6176988 (cit. on p. 14).

[78] U. R. Karpuzcu, I. Akturk, and N. S. Kim. "Accordion: Toward Soft Near-threshold Voltage Computing." In: *IEEE 20th Int. Symp. on High Performance Computer Architecture*. 2014, pp. 72–83. DOI: 10.1109/HPCA.2014.6835977 (cit. on pp. 12, 70).

[79] U. R. Karpuzcu, A. Sinkar, N. S. Kim, and J. Torrellas. "EnergySmart: Toward Energy-efficient Manycores for Near-threshold Computing." In: *IEEE 19th Int. Symp. on High Performance Computer Architecture*. 2013, pp. 542–553. DOI: 10.1109/HPCA.2013.6522348 (cit. on pp. 12, 58).

[80] G. Keramidas, M. Mavropoulos, A. Karvouniari, and D. Nikolos. "Spatial Pattern Prediction Based Management of Faulty Data Caches." In: *Design, Automation & Test in Europe Conf. Exhibition*. 2014, 60:1–60:6. DOI: http://dl.acm.org/citation.cfm?id=2616606.2616680 (cit. on pp. 18, 39).

[81] S. M. Khan, A. R. Alameldeen, C. Wilkerson, J. Kulkarni, and D. A. Jimenez. "Improving Multi-core Performance Using Mixed-cell Cache Architecture." In: *IEEE 19th Int. Symp. on High Performance Computer Architecture*. 2013, pp. 119–130. DOI: 10.1109/HPCA.2013.6522312 (cit. on pp. 16, 30, 38).

[82] S. M. Khan, Y. Tian, and D. A. Jimenez. "Sampling Dead Block Prediction for Last-level Caches." In: *43rd Annual IEEE/ACM Int. Symp. on Microarchitecture*. 2010, pp. 175–186. DOI: 10.1109/MICRO.2010.24 (cit. on p. 36).

[83] M. Khellah, A Keshavarzi, D. Somasekhar, T. Karnik, and V. De. "Read and Write Circuit Assist Techniques for Improving Vccmin of Dense 6T SRAM Cell." In: *IEEE Int. Conf. on Integrated Circuit Design and Technology and Tutorial*. 2008, pp. 185–188. DOI: 10.1109/ICICDT.2008.4567275 (cit. on p. 14).

[84] D. S. Khudia, B. Zamirai, M. Samadi, and S. Mahlke. "Rumba: An Online Quality Management System for Approximate Computing." In: *42nd Annual Int. Symp. on Computer Architecture*. 2015, pp. 554–566. DOI: 10.1145/2749469.2750371 (cit. on p. 70).

[85] C. H. I. Kim, H. Soeleman, and K. Roy. "Ultra-low-power DLMS Adaptive Filter for Hearing Aid Applications." In: *IEEE Trans. on Very Large Scale Integration (VLSI) Systems* 11.6 (Dec. 2003), pp. 1058–1067. DOI: 10.1109/TVLSI.2003.819573 (cit. on p. 11).

[86] N. S. Kim, T. Austin, D. Blaauw, T. Mudge, K. Flautner, J. S. Hu, M. J. Irwin, M. Kandemir, and V. Narayanan. "Leakage Current: Moore's Law Meets Static Power." In: *Computer* 36.12 (Dec. 2003), pp. 68–75. DOI: 10.1109/MC.2003.1250885 (cit. on p. 4).

[87] S. Kim, J. Lee, J. Kim, and S. Hong. "Residue Cache: A Low-energy Low-area L2 Cache Architecture via Compression and Partial Hits." In: *44th Annual IEEE/ACM Int. Symp. on Microarchitecture*. 2011, pp. 420–429. DOI: 10.1145/2155620.2155670 (cit. on pp. 18, 45).

[88] C.-K. Koh, W.-F. Wong, Y. Chen, and H. Li. "Tolerating Process Variations in Large, Set-associative Caches: The Buddy Cache." In: *ACM Trans. on Architecture and Code Optimization* 6.2 (July 2009), 8:1–8:34. DOI: 10.1145/1543753.1543757 (cit. on pp. 17, 39, 60).

[89] D. Kozen. "Semantics of Probabilistic Programs." In: *Journal of Computer and System Sciences* 22.3 (June 1981), pp. 328–350. DOI: http://dx.doi.org/10.1016/0022-0000(81)90036-2 (cit. on p. 70).

[90] J. Kulkarni, K. Kim, and K. Roy. "A 160mV Robust Schmitt Trigger Based Subthreshold SRAM." In: *IEEE Journal of Solid-State Circuits* 42.10 (Oct. 2007), pp. 2303–2313. DOI: 10.1109/JSSC.2007.897148 (cit. on p. 15).

[91] R. Kumar and G. Hinton. "A Family of 45nm IA Processors." In: *IEEE Int. Solid-State Circuits Conf. Digest of Technical Papers*. 2009, pp. 58–59. DOI: 10.1109/ISSCC.2009.4977306 (cit. on pp. 14, 21, 27).

[92] D. E. Lackey, P. S. Zuchowski, T. R. Bednar, D. W. Stout, S. W. Gould, and J. M. Cohn. "Managing Power and Performance for System-on-chip Designs Using Voltage Islands." In: *IEEE/ACM Int. Conf. on Computer Aided Design*. 2002, pp. 195–202. DOI: 10.1109/ICCAD.2002.1167534 (cit. on p. 16).

[93] N. Ladas, Y. Sazeides, and V. Desmet. "Performance-effective Operation Below Vccmin." In: *IEEE Int. Symp. on Performance Analysis of Systems Software*. 2010, pp. 223–234. DOI: 10.1109/ISPASS.2010.5452017 (cit. on pp. 16, 28, 38, 60).

[94] H. Lee, S. Cho, and B. Childers. "Performance of Graceful Degradation for Cache Faults." In: *IEEE Computer Society Annual Symp. on VLSI*. 2007, pp. 409–415. DOI: 10.1109/ISVLSI.2007.81 (cit. on p. 16).

[95] 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: *42nd Annual IEEE/ACM Int. Symp. on Microarchitecture*. 2009, pp. 469–480. DOI: 10.1145/1669112.1669172 (cit. on pp. 22, 58).

[96] X. Li and D. Yeung. "Application-Level Correctness and its Impact on Fault Tolerance." In: *IEEE 13th Int. Symp. on High Performance Computer Architecture*. 2007, pp. 181–192. DOI: 10.1109/HPCA.2007.346196 (cit. on pp. 70, 79).

[97] D. Liu, T. Chen, S. Liu, J. Zhou, S. Zhou, O. Teman, X. Feng, X. Zhou, and Y. Chen. "PuDianNao: A Polyvalent Machine Learning Accelerator." In: *20th Int. Conf. on Architectural Support for Programming Languages and Operating Systems*. 2015, pp. 369–381. DOI: 10.1145/2694344.2694358 (cit. on p. 5).

[98] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn. "Flikker: Saving DRAM Refresh-power Through Critical Data Partitioning." In: *16th Int. Conf. on Architectural Support for Programming Languages and Operating Systems*. 2011, pp. 213–224. DOI: 10.1145/1950365.1950391 (cit. on p. 70).

[99] G. H. Loh, N. Jayasena, M Oskin, M. Nutter, D. Roberts, M. Meswani, D. P. Zhang, and M. Ignatowski. "A Processing in Memory Taxonomy and a Case for Studying Fixed-function PIM." In: *1st Workshop on Near-Data Processing*. 2013 (cit. on p. 5).

[100] B. C. Lopes, R. Auler, L. Ramos, E. Borin, and R. Azevedo. "SHRINK: Reducing the ISA Complexity via Instruction Recycling." In: *42nd Annual Int. Symp. on Computer Architecture*. 2015, pp. 311–322. DOI: 10.1145/2749469.2750391 (cit. on pp. 68, 69).

[101] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood. "Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation." In: *ACM SIGPLAN Conf. on Programming Language Design and Implementation*. 2005, pp. 190–200. DOI: 10.1145/1065010.1065034 (cit. on pp. 69, 75).

[102] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. "Simics: A Full System Simulation Platform." In: *Computer* 35.2 (Feb. 2002), pp. 50–58. DOI: 10.1109/2.982916 (cit. on p. 22).

[103] D. Mahajan, A. Yazdanbakhsh, J. Park, B. Thwaites, and H. Esmaeilzadeh. "Prediction-based Quality Control for Approximate Accelerators." In: *Workshop on Approximate Computing Across the System Stack*. 2015 (cit. on p. 70).

[104] T. Mahmood, S. Kim, and S. Hong. "Macho: A Failure Model-oriented Adaptive Cache Architecture to Enable Near-threshold Voltage Scaling." In: *IEEE 19th Int. Symp. on High Performance Computer Architecture*. 2013, pp. 532–541. DOI: 10.1109/HPCA.2013.6522347 (cit. on pp. 17, 39, 60).

[105] W.-K. Mak and J.-W. Chen. "Voltage Island Generation Under Performance Requirement for SoC Designs." In: *Asia and South Pacific Design Automation Conf.* 2007, pp. 798–803. DOI: 10.1109/ASPDAC.2007.358087 (cit. on pp. 14, 16).

[106] D. Markovic, C. Wang, L. Alarcon, T.-T. Liu, and J. Rabaey. "Ultralow-Power Design in Near-Threshold Region." In: *Proc. of the IEEE* 98.2 (Feb. 2010), pp. 237–252. DOI: 10.1109/JPROC.2009.2035453 (cit. on p. 12).

[107] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood. "Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset." In: *SIGARCH Comput. Archit. News* 33.4 (Nov. 2005), pp. 92–99. DOI: 10.1145/1105734.1105747 (cit. on p. 22).

[108] J. S. Miguel, M. Badr, and N. E. Jerger. "Load Value Approximation." In: *47th Annual IEEE/ACM Int. Symp. on Microarchitecture*. 2014, pp. 127–139. DOI: 10.1109/MICRO.2014.22 (cit. on p. 68).

[109] S. Misailovic, S. Sidiroglou, and M. C. Rinard. "Dancing with Uncertainty." In: *ACM Workshop on Relaxing Synchronization for Multicore and Manycore Scalability*. 2012, pp. 51–60. DOI: 10.1145/2414729.2414738 (cit. on p. 68).

[110] S. Misailovic, S. Sidiroglou, H. Hoffmann, and M. Rinard. "Quality of Service Profiling." In: *32nd ACM/IEEE Int. Conf. on Software Engineering*. 2010, pp. 25–34. DOI: 10.1145/1806799.1806808 (cit. on pp. 68, 70, 71).

[111] S. Misailovic, M. Carbin, S. Achour, Z. Qi, and M. C. Rinard. "Chisel: Reliability- and Accuracy-aware Optimization of Approximate Computational Kernels." In: *ACM Int. Conf. on Object Oriented Programming Systems Languages & Applications*. Portland, Oregon, USA, 2014, pp. 309–328. DOI: 10.1145/2660193.2660231 (cit. on p. 70).

[112] S. Mittal and J. S. Vetter. "A Survey of Software Techniques for Using Non-Volatile Memories for Storage and Main Memory Systems." In: *IEEE Trans. on Parallel and Distributed Systems* 27.5 (May 2016), pp. 1537–1550. DOI: 10.1109/TPDS.2015.2442980 (cit. on p. 5).

[113] S. Mittal. "A Survey of Techniques for Approximate Computing." In: *ACM Computing Surveys* 48.4 (Mar. 2016), 62:1–62:33. DOI: 10.1145/2893356 (cit. on p. 5).

[114] G. Moore. "Cramming More Components onto Integrated Circuits." In: *Electronics* 38.8 (Apr. 1965), pp. 114–117. DOI: 10.1109/N-SSC.2006.4785860 (cit. on p. 3).

[115] T. Moreau, A. Sampson, L. Ceze, and M. Oskin. "Approximating to the Last Bit." In: *Workshop on Approximate Computing Across the Stack*. 2016 (cit. on pp. 71, 80).

[116] R. Narayanan, B. Ozisikyilmaz, J. Zambreno, G. Memik, and A. Choudhary. "MineBench: A Benchmark Suite for Data Mining Workloads." In: *IEEE Int. Symp. on Workload Characterization*. 2006, pp. 182–188. DOI: 10.1109/IISWC.2006.302743 (cit. on p. 73).

[117] N. Nethercote and J. Seward. "Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation." In: *28th ACM SIGPLAN Conf. on Programming Language Design and Implementation*. 2007, pp. 89–100. DOI: 10.1145/1250734.1250746 (cit. on p. 75).

[118] D. J. Palframan, N. S. Kim, and M. H. Lipasti. "iPatch: Intelligent Fault Patching to Improve Energy Efficiency." In: *IEEE 21st Int. Symp. on High Performance Computer Architecture*. 2015, pp. 428–438. DOI: 10.1109/HPCA.2015.7056052 (cit. on p. 17).

[119] G. Pekhimenko, V. Seshadri, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry. "Base-delta-immediate Compression: Practical Data Compression for On-chip Caches." In: *21st Int. Conf. on Parallel Architectures and Compilation Techniques*. 2012, pp. 377–388. DOI: 10.1145/2370816.2370870 (cit. on pp. 18, 45, 46, 48, 52, 60).

[120] G. Pekhimenko, T. Huberty, R. Cai, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry. "Exploiting Compressed Block Size as an Indicator of Future Reuse." In: *IEEE 21st Int. Symp. on High Performance Computer Architecture*. 2015, pp. 51–63. DOI: 10.1109/HPCA.2015.7056021 (cit. on pp. 18, 60).

[121] M. J. M. Pelgrom, A. C. J. Duinmaijer, and A. P. G. Welbers. "Matching Properties of MOS Transistors." In: *IEEE Journal of Solid-State Circuits* 24.5 (Oct. 1989), pp. 1433–1439. DOI: 10.1109/JSSC.1989.572629 (cit. on p. 4).

[122] N. Pinckney, R. Dreslinski, K. Sewell, D. Fick, T. Mudge, D. Sylvester, and D. Blaauw. "Limits of Parallelism and Boosting in Dim Silicon." In: *IEEE Micro* 33.5 (Sept. 2013), pp. 30–37. DOI: 10.1109/MM.2013.73 (cit. on p. 12).

[123] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar. "Gated-Vdd: A Circuit Technique to Reduce Leakage in Deep-submicron Cache Memories." In: *Int. Symp. on Low Power Electronics and Design*. 2000, pp. 90–95. DOI: 10.1109/LPE.2000.155259 (cit. on p. 42).

[124] *RISC-V: The Free and Open RISC Instruction Set Architecture*. URL: https://riscv.org (cit. on p. 79).

[125] A. Rahimi, A. Marongiu, R. K. Gupta, and L. Benini. "A Variability-aware OpenMP Environment for Efficient Execution of Accuracy-configurable Computation on Shared-FPU Processor Clusters." In: *9th IEEE/ACM/IFIP Int. Conf. on Hardware/Software Codesign and System Synthesis*. 2013, 35:1–35:10 (cit. on p. 70).

[126] V. J. Reddi, S. Kanev, W. Kim, S. Campanoni, M. D. Smith, G. Y. Wei, and D. Brooks. "Voltage Smoothing: Characterizing and Mitigating Voltage Noise in Production Processors via Software-guided Thread Scheduling." In: *43rd Annual IEEE/ACM Int. Symp. on Microarchitecture*. 2010, pp. 77–88. DOI: 10.1109/MICRO.2010.35 (cit. on p. 12).

[127] M. Ringenburg, A. Sampson, I. Ackerman, L. Ceze, and D. Grossman. "Monitoring and Debugging the Quality of Results in Approximate Programs." In: *20th Int. Conf. on Architectural Support for Programming Languages and Operating Systems*. 2015, pp. 399–411. DOI: 10.1145/2694344.2694365 (cit. on p. 70).

[128] P. Rosenfeld, E. Cooper-Balis, and B. Jacob. "DRAMSim2: A Cycle Accurate Memory System Simulator." In: *Computer Architecture Letters* 10.1 (Jan. 2011), pp. 16–19. DOI: 10.1109/L-CA.2011.4 (cit. on p. 22).

[129] E. Rotem, A. Mendelson, R. Ginosar, and U. Weiser. "Multiple Clock and Voltage Domains for Chip Multiprocessors." In: *42nd Annual IEEE/ACM Int. Symp. on Microarchitecture*. 2009, pp. 459–468. DOI: doi={10.1145/1669112.1669170} (cit. on p. 4).

[130] P. Roy, R. Ray, C. Wang, and W. F. Wong. "ASAC: Automatic Sensitivity Analysis for Approximate Computing." In: *SIGPLAN/SIGBED Conf. on Languages, Compilers and Tools for Embedded Systems*. 2014, pp. 95–104. DOI: 10.1145/2597809.2597812 (cit. on p. 70).

[131] C. Rubio-González, C. Nguyen, H. D. Nguyen, J. Demmel, W. Kahan, K. Sen, D. H. Bailey, C. Iancu, and D. Hough. "Precimonious: Tuning Assistant for Floating-point Precision." In: *Int. Conf. on High Performance Computing, Networking, Storage and Analysis*. 2013, 27:1–27:12. DOI: 10.1145/2503210.2503296 (cit. on pp. 68, 71, 80).

[132] M. Samadi, J. Lee, D. A. Jamshidi, A. Hormati, and S. Mahlke. "SAGE: Self-tuning Approximation for Graphics Engines." In: *46th Annual IEEE/ACM Int. Symp. on Microarchitecture*. 2013, pp. 13–24. DOI: 10.1145/2540708.2540711 (cit. on p. 68).

[133] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman. "EnerJ: Approximate Data Types for Safe and General Low-power Computation." In: *32nd ACM SIGPLAN Conf. on Programming Language Design and Implementation*. 2011, pp. 164–174. DOI: 10.1145/1993498.1993518 (cit. on pp. 5, 68, 70, 71, 80, 81).

[134] A. Sampson, J. Nelson, K. Strauss, and L. Ceze. "Approximate Storage in Solid-state Memories." In: *46th Annual IEEE/ACM Int. Symp. on Microarchitecture*. 2013, pp. 25–36. DOI: 10.1145/2540708.2540712 (cit. on p. 70).

[135] S. Sardashti, A. Seznec, and D. A. Wood. "Skewed Compressed Caches." In: *47th Annual IEEE/ACM Int. Symp. on Microarchitecture*. 2014, pp. 331–342. DOI: 10.1109/MICRO.2014.41 (cit. on pp. 18, 52).

[136] E. Schkufza, R. Sharma, and A. Aiken. "Stochastic Optimization of Floating-point Programs with Tunable Precision." In: *35th ACM SIGPLAN Conf. on Programming Language Design and Implementation*. 2014, pp. 53–64. DOI: 10.1145/2594291.2594302 (cit. on p. 71).

[137] S. Schuster. "Multiple Word/Bit Line Redundancy for Semiconductor Memories." In: *IEEE Journal of Solid-State Circuits* 13.5 (Oct. 1978), pp. 698–703. DOI: 10.1109/JSSC.1978.1051122 (cit. on p. 15).

[138] E. v. Setten, F. Wittebrood, E. Psara, D. Oorschot, and V. Philipsen. "Patterning Options for N7 Logic: Prospects and Challenges for EUV." In: *SPIE 9661, 31st European Mask and Lithography Conf., 96610G*. 2015. DOI: 10.1117/12.2196426 (cit. on p. 3).

[139] A. Shan. "Heterogeneous Processing: A Strategy for Augmenting Moore's Law." In: *Linux J.* 2006.142 (Feb. 2006), pp. 7– (cit. on p. 4).

[140] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard. "Managing Performance vs. Accuracy Trade-offs with Loop Perforation." In: *19th ACM SIGSOFT Symp. and the 13th European Conf. on Foundations of Software Engineering*. 2011, pp. 124–134. DOI: 10.1145/2025113.2025133 (cit. on p. 71).

[141] H. Soeleman and K. Roy. "Ultra-low Power Digital Subthreshold Logic Circuits." In: *Int. Symp on Low Power Electronics and Design*. 1999, pp. 94–96. DOI: 10.1145/313817.313874 (cit. on p. 11).

[142] G. Sohi. "Cache Memory Organization to Enhance the Yield of High Performance VLSI Processors." In: *IEEE Trans. on Computers* 38.4 (Apr. 1989), pp. 484–492. DOI: 10.1109/12.21141 (cit. on pp. 6, 16, 27, 28, 60).

[143] J. Sorber, A. Kostadinov, M. Garber, M. Brennan, M. D. Corner, and E. D. Berger. "Eon: A Language and Runtime System for Perpetual Systems." In: *5th Int. Conf. on Embedded Networked Sensor Systems*. 2007, pp. 161–174. DOI: 10.1145/1322263.1322279 (cit. on p. 70).

[144] M. Stephenson, J. Babb, and S. Amarasinghe. "Bidwidth Analysis with Application to Silicon Compilation." In: *ACM SIGPLAN Conf. on Programming Language Design and Implementation*. 2000, pp. 108–120. DOI: 10.1145/349299.349317 (cit. on pp. 71, 80).

[145] Sun Microsystems. *UltraSPARC T2 Supplement to the UltraSPARC Architecture*. Draft D1.4.3. Sun Microsystems Inc., 2007 (cit. on p. 22).

[146] X. Tang, V. K. De, and J. D. Meindl. "Intrinsic MOSFET Parameter Fluctuations due to Random Dopant Placement." In: *IEEE Trans. on Very Large Scale Integration (VLSI) Systems* 5.4 (Dec. 1997), pp. 369–376. DOI: 10.1109/92.645063 (cit. on p. 13).

[147] M. Taylor. "A Landscape of the New Dark Silicon Design Regime." In: *IEEE Micro* 33.5 (Sept. 2013), pp. 8–19. DOI: 10.1109/MM.2013.90 (cit. on p. 4).

[148] D. D. Thaker, D. Franklin, J. Oliver, S. Biswas, D. Lockhart, T. Metodi, and F. T. Chong. "Characterization of Error-tolerant Applications when Protecting Control Data." In: *IEEE Int. Symp. on Workload Characterization*. 2006, pp. 142–149. DOI: 10.1109/IISWC.2006.302738 (cit. on pp. 70, 79).

[149] S. Thomas, C. Gohkale, E. Tanuwidjaja, T. Chong, D. Lau, S. Garcia, and M. B. Taylor. "CortexSuite: A Synthetic Brain Benchmark Suite." In: *IEEE Int. Symp. on Workload Characterization*. 2014, pp. 76–79. DOI: 10.1109/IISWC.2014.6983043 (cit. on p. 73).

[150] Y. F. Tong, R. A. Rutenbar, and D. F. Nagle. "Minimizing Floating-point Power Dissipation via Bit-width Reduction." In: *Power-Driven Microarchitecture Workshop*. 1998 (cit. on pp. 71, 80).

[151] V. Vassiliadis, K. Parasyris, C. Chalios, C. D. Antonopoulos, S. Lalis, N. Bellas, H. Vandierendonck, and D. S. Nikolopoulos. "A Programming Model and Runtime System for Significance-aware Energy-efficient Computing." In: *20th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming*. 2015, pp. 275–276. DOI: 10.1145/2688500.2688546 (cit. on p. 70).

[152] L. Villa, M. Zhang, and K. Asanović. "Dynamic Zero Compression For Cache Energy Reduction." In: *33rd Annual ACM/IEEE Int. Symp. on Microarchitecture*. 2000, pp. 214–220. DOI: 10.1145/360128.360150 (cit. on p. 48).

[153] D. Wackerly, W. Mendenhall, and R. L. Scheaffer. *Mathematical Statistics With Applications*. Nelson Education, 2007 (cit. on p. 23).

[154] A. Wang and A. Chandrakasan. "A 180mV FFT Processor Using Subthreshold Circuit Techniques." In: *IEEE Int. Solid-State Circuits Conf. Digest of Technical Papers*. 2004, pp. 292–529. DOI: 10.1109/ISSCC.2004.1332709 (cit. on p. 11).

[155] L. Wang and K. Skadron. "Implications of the Power Wall: Dim Cores and Reconfigurable Logic." In: *IEEE Micro* 33.5 (Sept. 2013), pp. 40–48. DOI: 10.1109/MM.2013.74 (cit. on p. 68).

[156] N. Weste and D. Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*. 4th. Addison-Wesley Publishing Company, 2010 (cit. on p. 4).

[157] C. Wilkerson, H. Gao, A. R. Alameldeen, Z. Chishti, M. Khellah, and S.-L. Lu. "Trading off Cache Capacity for Reliability to Enable Low Voltage Operation." In: *35th Annual Int. Symp. on Computer Architecture*. 2008, pp. 203–214. DOI: 10.1109/ISCA.2008.22 (cit. on pp. 17, 21, 30, 36, 39, 50, 60).

[158] V. Wong and M. Horowitz. "Soft Error Resilience of Probabilistic Inference Applications." In: *Workshop on Silicon Errors in Logic-System Effects*. 2006 (cit. on pp. 70, 79).

[159] C.-J. Wu, A. Jaleel, W. Hasenplaugh, M. Martonosi, S. C. Steely Jr., and J. Emer. "SHiP: Signature-based Hit Predictor for High Performance Caching." In: *44th Annual IEEE/ACM Int. Symp. on Microarchitecture*. 2011, pp. 430–441. DOI: 10.1145/2155620.2155671 (cit. on p. 36).

[160] C. Xu, D. Niu, N. Muralimanohar, R. Balasubramonian, T. Zhang, S. Yu, and Y. Xie. "Overcoming the Challenges of Crossbar Resistive Memory Architectures." In: *IEEE 21st Int. Symp. on High Performance Computer Architecture*. 2015, pp. 476–488. DOI: 10.1109/HPCA.2015.7056056 (cit. on p. 18).

[161] Q. Xu, T. Mytkowicz, and N. S. Kim. "Approximate Computing: A Survey." In: *IEEE Design & Test* 33.1 (Feb. 2016), pp. 8–22. DOI: 10.1109/MDAT.2015.2505723 (cit. on p. 5).

[162] J. Yang, Y. Zhang, and R. Gupta. "Frequent Value Compression in Data Caches." In: *33rd Annual IEEE/ACM Int. Symp. on Microarchitecture*. 2000, pp. 258–265. DOI: 10.1109/MICRO.2000.898076 (cit. on pp. 18, 45, 46, 48).

[163] A. Yazdanbakhsh, D. Mahajan, B. Thwaites, J. Park, A. Nagendrakumar, S. Sethuraman, K. Ramkrishnan, N. Ravindran, R. Jariwala, A. Rahimi, H. Esmaeilzadeh, and K. Bazargan. "Axilog: Language Support for Approximate Hardware Design." In: *Design, Automation & Test in Europe Conf. Exhibition*. 2015, pp. 812–817 (cit. on p. 70).

[164] T. Y. Yeh, G. Reinman, S. J. Patel, and P. Faloutsos. "Fool Me Twice: Exploring and Exploiting Error Tolerance in Physics-based Animation." In: *ACM Trans. on Graphics* 29.1 (Dec. 2009), 5:1–5:11. DOI: 10.1145/1640443.1640448 (cit. on pp. 68, 71, 80).

[165] G. Yeric. "Moore's Law at 50: Are we Planning for Retirement?" In: *IEEE Int. Electron Devices Meeting*. 2015, pp. 1.1.1–1.1.8. DOI: 10.1109/IEDM.2015.7409607 (cit. on p. 4).

[166] R. Zahir, M. Ewert, and H. Seshadri. "The Medfield Smartphone: Intel Architecture in a Handheld Form Factor." In: *IEEE Micro* 33.6 (Nov. 2013), pp. 38–46. DOI: 10.1109/MM.2013.22 (cit. on pp. 14, 16).

[167] B. Zhai, D. Blaauw, D. Sylvester, and S. Hanson. "A Sub-200mV 6T SRAM in 0.13um CMOS." In: *IEEE Int. Solid-State Circuits Conf. Digest of Technical Papers*. 2007, pp. 332–334 (cit. on p. 14).

[168] H. Zhang, M. Putic, and J. Lach. "Low Power GPGPU Computation with Imprecise Hardware." In: *51st ACM/EDAC/IEEE Design Automation Conf.* 2014, pp. 1–6 (cit. on p. 68).

[169] Q. Zhang, T. Wang, Y. Tian, F. Yuan, and Q. Xu. "ApproxANN: An Approximate Computing Framework for Artificial Neural Network." In: *Design, Automation & Test in Europe Conf. Exhibition*. 2015, pp. 701–706 (cit. on pp. 71, 80).

[170] S.-T. Zhou, S. Katariya, H. Ghasemi, S. Draper, and N. S. Kim. "Minimizing Total Area of Low-voltage SRAM Arrays Through Joint Optimization of Cell Size, Redundancy, and ECC." In: *IEEE Int. Conf. on Computer Design*. 2010, pp. 112–117. DOI: 10.1109/ICCD.2010.5647605 (cit. on pp. 14–16, 24, 41, 63).