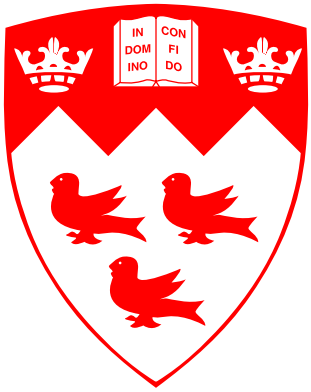
Redundancy for Cost, Performance, and Lifetime Improvement in Application Specific SIMT Processors

Seyyed Hasan Mozafari



Department of Electrical and Computer Engineering
McGill University
Montréal, Québec, Canada

December 2018

A thesis submitted to McGill University in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Dedicated to...

Abstract

Redundancy is now routinely allocated in circuits, micro-architectural structures, or at the system level, to mitigate mounting manufacturing yield losses and reducing the cost of fabricating ICs. Also, the structure of single-instruction multiple-thread (SIMT) systems makes them particularly suitable for applying redundancy at multiple levels of granularity: coarse-grained (core) and fine-grained (lane).

To address yield loss in SIMT processors, this thesis presents cold and hot core-, lane-, and shared-lane-sparing. Cold sparing is a kind of redundancy that addresses cost only, while hot-sparing is available to increase yield (and reduce costs) when the components are defective; otherwise, it can be used to improve performance in the field. Adding cold redundancy, specifically fine-grained one, would have timing overheads to the system's critical path. Also, hot sparing can improve the performance of a SIMT system for some applications. Therefore this thesis characterizes performance and cost together for SIMT systems with redundancy. In this regard, it introduces a metric that captures cost and performance together, *expected performance per cost* ($E[P]/C$). Then, it shows that for a case study system hot-sparing improves $E[P]/C$ significantly, while it shows that cold sparing has a negligible performance overhead in SIMT systems.

Next, it introduces two $E[P]$ estimation techniques, $\hat{E}_m[P]$ and $\hat{E}_s[P]$. Hot-sparing, complicates system evaluation: the presence of defects affects what resources are available in the field. Accurate performance evaluation thus requires the simulation of the entire population of resulting dice in order to

determine the *expected performance* $E[P]$ of the system. While simply expensive for single system evaluation, it is intractable for design space exploration. Therefore, it deploys $\hat{E}_m[P]$ to reduce the $E[P]$ calculation complexity. $\hat{E}_m[P]$ reduces simulation time by 93%. This remains computationally expensive for design space exploration when individual, detailed, simulations require hours. Therefore, it introduces $\hat{E}_s[P]$. $\hat{E}_s[P]$ reduces simulation by 98% with no more than 2.6% error in $E[P]$ sufficient for design space exploration. Consequently, designers may add redundancy, and evaluate system performance and cost, with no greater design effort than performance evaluation alone.

Lastly, this thesis investigates performance-per-watt (PPW) and lifetime-chip-performance (LCP) implications in SIMT systems with hot-sparing. Hot-sparing improves the system performance for some applications. However it adds power consumption, and consequently, hot-sparing may increase the temperature of a system, which can lead to lower lifetime and LCP. It shows that hot-sparing is effective for specific types of SIMT processor configurations (small and medium sized) in terms of PPW. On these configurations, hot-sparing can improve PPW more than 16%, on average, for some applications. It shows that on the contrary embedding hot-sparing into SIMT processors not only does not damage LCP, rather it increases LCP outstandingly for some specific types of SIMT processor configurations (small and medium systems) and applications (FFT and FILTER), while hot-sparing improves cost and LCP over other configurations and applications as well. For example, hot-sparing can improve LCP more than 75% compared with conventional methods (i.e., cold sparing), on average, for FFT and FILTER applications.

Abrégé

La redondance est maintenant systématiquement attribuée dans les circuits, les structures micro-architecturales ou au niveau du système, afin dátténuer láugmentation des pertes de rendement de fabrication et la réduction des coûts de fabrication des circuits intégrés. En outre, la structure des systèmes á instructions multiples á threads unique (SIMT) les rend particulièrement adapté pour appliquer une redondance á plusieurs niveaux de granularité: á grain grossier (noyau) et á grain fin (piste). Pour remédier aux pertes de rendement dans les processeurs SIMT, cette thèse présente une économie de froid, de chaud et de froid pour les noyaux chauds et partagés. Le froid épargne est une sorte de redondance qui ne traite que des coûts, alors que le chaud épargne est disponible pour augmenter le rendement (et réduire les coûts) lorsque les composants sont défectueux; sinon, il peut tre utilisé pour améliorer performance sur le terrain.

Lájout de redondance á froid, en particulier á grain fin, aurait des coûts de synchronisation sur le chemin critique du système. En outre, le remplacement á chaud peut améliorer les performances d'un système SIMT pour certaines applications. Par conséquent, cette thèse caractérise la performance et le coût ensemble pour les systèmes SIMT avec redondance. á cet égard, il introduit une métrique qui rend compte des coûts et des performances ensemble, *performances attendues par coût* ($E[P]/C$). Ensuite, il montre que, pour une étude de cas, l'économie de stockage á chaud améliore considérablement $E[P]/C$, alors quélle montre que l'épargne de froid a un surcoût de perfor-

mance négligeable dans les systèmes SIMT. Ensuite, il introduit deux techniques déstimation $E[P]$, $\hat{E}_m[P]$ et $\hat{E}_s[P]$.

Hot-Sparing, complique l'évaluation du système: la présence de défauts affecte les ressources disponibles sur le terrain Une évaluation précise des performances nécessite donc la simulation de la population entière des dés résultants afin de déterminer la *performance attendue* $E[P]$ du système. Bien que tout simplement coûteux pour une évaluation de système unique, il est insoluble pour l'exploration d'espace de conception. Par conséquent, il déploie firstEvalTech pour réduire la complexité du calcul $E[P]$. $\hat{E}_m[P]$ réduit le temps de simulation de 93%. Cela reste coûteux en termes de calcul pour l'exploration de l'espace de conception lorsque des simulations individuelles et détaillées nécessitent des heures. Par conséquent, il introduit $\hat{E}_s[P]$. $\hat{E}_s[P]$ réduit la simulation de 98% sans qu'une erreur de plus de 2.6% dans $E[P]$ soit suffisante pour l'exploration de l'espace de conception. Par conséquent, les concepteurs peuvent ajouter de la redondance et évaluer les performances et les coûts du système, sans autre effort de conception que la seule évaluation des performances.

Enfin, cette thèse étudie les implications en termes de performances par watt (PPW) et de performances á la puce sur la durée de vie (LCP) dans les systèmes SIMT avec hot-sparing. Le remplacement á chaud améliore les performances du système pour certaines applications. Cependant, cela augmente la consommation d'énergie et, par conséquent, la gestion á chaud peut augmenter la température d'un système, ce qui peut réduire la durée de vie du système et la durée de vie du système. Cela montre que le hot-sparing est

efficace pour certains types de configurations de processeurs SIMT (petites et moyennes) en termes de PPW. Sur ces configurations, la sauvegarde á chaud peut améliorer PPW de plus de 16% en moyenne pour certaines applications. Cela montre quáu contraire l'intégration de la sauvegarde á chaud dans les processeurs SIMT non seulement n'endommage pas le LCP, mais au contraire, elle augmente considérablement le LCP pour certains types de processeurs SIMT spécifiques configurations (petits et moyens systèmes) et applications (FFT et FILTER), tandis que la gestion á chaud améliore les coûts et le LCP par rapport aux autres configurations et applications aussi bien. Par exemple, le maintien á chaud peut améliorer le LCP plus moins de 75% par rapport aux méthodes conventionnelles (épargne á froid), en moyenne, pour les applications FFT et FILTER.

Acknowledgements

I would like to express my special appreciation and thanks to my advisor Dr. Brett H. Meyer, you have been a tremendous mentor for me. I would like to thank you for encouraging my research and for allowing me to grow as a research scientist. Your advice on both research has been invaluable. I would also like to thank my committee members, professor Z. Zilic, professor W. Gross, and D. Nowrouzezahrai for serving as my committee members even at hardship. I also want to thank you for your brilliant comments and suggestions, thanks to you. A special thanks to my family. Words can not express how grateful I am to my my mother, and father for all of the sacrifices that youve made on my behalf. Your prayer for me was what sustained me thus far. Finally I thank my God, my good Father, for letting me through all the difficulties. I have experienced Your guidance day by day. You are the one who let me finish my degree. I will keep on trusting You for my future. Thank you, Lord.

I am also grateful to the Fonds de Recherche du Québec - Nature et Technologies (FRQNT) and McGill University for financially supporting this work via McGill Engineering Doctoral Awards (MEDA) scholarship.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter, we show that yield loss and reliability problems become more important in multi-core processors as technology size shrinks. In this regard, we study a widely used type of multi-core processors, single-instruction, multiple-thread (SIMT). Then, we introduce our solutions to address yield and reliability in these systems, architectural and micro-architectural cold and hot redundancies. Also, we introduce the implications that these solutions may have on other important design factors such as performance-per-watt and lifetime-chip-performance. In the end, we mention the research articles that are extracted from this work.

1.1 Motivations

Devices are more likely to experience early performance degradation, and even failure in the field due to systematic and random defects [3, 4]. Defect rates have been increased by 1-2 orders of magnitude from 65nm to 22nm tech-

nologies, and this trend continues for newer technology sizes [5, 6]. Such observations support the need for yield-enhancing design techniques to ensure viable future manufacturing yield. Yield can be improved by adding redundancy in architectural and micro-architectural levels in processors [7]. However, some performance and/or die area may need to be sacrificed to benefit yield and reliability [8]. Therefore, a question arises: under what circumstances does a redundancy regime can *optimize* yield? Note that redundancy improves yield, but has area overhead which reduces the number of fabricated chips per wafer. Therefore, in this work, instead of yield maximization, we aim at optimizing manufacturing cost.

Single-instruction, multiple-thread (SIMT) systems [9], which are the consequent generation of single-instruction, multiple-data (SIMD) processors [10] are especially well-suited to incorporating redundancy at multiple granularities (cores and lanes) [2]. SIMT architectures consist of multiple levels of design abstraction: (a) the system, which consists of multiple thread-parallel cores (which are referred to streaming multi-processor (SM) in NVIDIA G80 terminology [10]); and, (b) the cores, each of which consists of a single front-end unit and multiple, data-parallel processing elements (lanes). In NVIDIA G80 terminology, a lane consists of a streaming processor (SP), and a bank of D$ or shared-memory [10]. Therefore, SIMT processors are well-suited to incorporating redundancy at architectural and micro-architectural levels.

Designers need to investigate the opportunity to reduce manufacturing cost in the presence of random defects by employing multi-granularity redundancy in SIMT architecture (cold sparing). However, when designers add

redundancy, specifically, fine-grained (e.g., a spare lane), it adds performance overhead (redundancy can add timing overhead on the critical path of a system) [11]. Therefore, designers need to consider design alternatives in the space of cost-performance trade-offs. This is significant: tools are therefore needed to assist designers with determining when and what sort of redundancy to apply to optimize such systems.

Designers should investigate the cost and performance implications of employing hot spares in SIMT processors. Hot spares are available to increase yield (and reduce cost) when the components are defective; otherwise, they can be used to improve performance in the field [12]. Integrating such components presents additional challenges compared with cold spares: 1) hot spare components must be allocated in such a way that the programming model is not disrupted. 2) Expected performance ($E[P]$) should be calculated in the systems with hot redundancy. $E[P]$ captures the performance of a multi-core SIMT system by considering the resulting population of operational dice (hereafter denoted derivative designs). 3) Designers need to develop a probability of occurrence model for SIMT processors with hot sparing to calculate their $E[P]$.

$E[P]$ must be estimated. Unfortunately, measuring the performance of all configurations derived from a single SIMT system is computationally intractable [2]. Therefore, to reduce the complexity, designers group derivative designs suffering from symmetric failures. However, even by grouping similar derivative configurations, calculating $E[P]$ accurately would be computationally expensive (it takes 792 hours on a Corei7 with 8GB RAM for a single

SIMT system [2]). While this may be tractable for a single design point, such high computational requirements make design space exploration impossible (a mid-size SIMT processor design space consists of 10K systems [13]). Therefore, designers need to estimate $E[P]$. However, finding an estimation method to be fast, but accurate enough to distinguish designs is not trivial.

Designers need to consider hot-sparing implications on other important design factors; performance-per-watt (PPW) and lifetime-chip-performance (LCP) in SIMT processors. Hot spares can increase the power consumption of the processors to the point that damages PPW. Also, this power consumption increase may lead to temperature increase and consequently damages lifetime and LCP. However, on the contrary, the relative performance gain of hot-sparing, for some applications, can be better than its associate relative power and lifetime overhead. Therefore, it is very important for designers to know for which applications and multi-core processor configurations they should (not) utilize hot-sparing to gain the potential benefit in improving cost-PPW and cost-LCP.

1.2 Objectives

The objectives of this work are: (a) to address cost problem in SIMT systems by adding cold redundancy, (b) to introduce hot-sparing as a method to improve performance and cost together in SIMT processors, and (c) to show that hot-sparing can even improve other design factors such as PPW and LCP as well. The main objective of this thesis is to show that redundancy is the way to improve SIMT architectures in different design factors.

1.3 Thesis Contributions

To address the implications that utilizing redundancy have on application-specific SIMT processors in terms of cost, performance, PPW, and LCP, we make the following contributions in this thesis:

Redundancy to Improve Performance and Cost in SIMT Systems

Premkishore et al., [14] and Schuchman and Vijaykumar [15] have investigated the effect of core sparing and micro-architecture redundancy techniques on yield improvement in general purpose multi-core processors (i.e. cold sparing). However, they show that micro-architectural redundancy could have a considerable performance overhead and it becomes more effective as technology scales. We show that the performance overhead of utilizing micro-architectural redundancy in SIMT processors is negligible since its architecture is well-suited to incorporating redundancy at multiple granularities (cores and lanes). In this regard, we utilize core- and lane-sparing. Also, we introduce a a new type of redundancy, shared-spare-lane for SIMT processors. We show that the performance overhead of fine-grained redundancies is negligible in the SIMT architecture.

Gao et al., [12] use hot micro-architectural redundancy in a general purpose processor to increase yield (and reduce cost) when the components are defective; otherwise, they try to use the redundancy to improve performance in the field. However, they show that for fine-grained redundancies, the performance overhead of micro-architectural redundancy usually is bigger than its performance gain, therefore, they utilize these redundancies to extend system's

lifetime instead. We take into consideration this performance overhead and we show that this is not the case for SIMT systems (performance of some systems with hot-sparing could be improved over some applications). Also, we introduce two new types of micro-architectural redundancies in SIMT processors, hot spare-lane (Hot-SL) and hot shared-spare-lane (Hot-SSL). We evaluate cost and performance overhead of utilizing Hot-SL and Hot-SSL, and we show that they significantly improve SIMT systems in terms of expected performance per cost, $E[P]/C$.

Estimating Performance for Systems with Hot Redundancy

Prior work has explored how to quantify performance in the context of defects [16, 15, 12]. They calculate system performance by simulating all possible systems with defects and weighting their contribution to expected performance by the likelihood of each configuration occurring. However it is almost computationally intractable in our case since we need detailed multi-core performance simulation. Therefore, we introduce an approximation for expected performance, $\hat{E}_m[P]$ which evaluate systems at least 11× faster and have 7% error (suitable for distinguishing SIMT systems in terms of performance). Moreover, we extend upon this estimation technique ($\hat{E}_m[P]$) and introduce an alternative method of calculating, $\hat{E}_s[P]$. This approach goes further: it only evaluates the most likely derivative design, reducing the computational cost by an additional 3×, while it's estimating error is less than 2.6%.

Hot-Sparing for Performance-per-Watt and Lifetime-Chip-Performance Improvement in SIMT Processors

Ghasemazar et al., [17] and Hanumaiah et al., [18] showed that by using system-level techniques (dynamic voltage and frequency scaling, task allocation, and task migration) we can significantly increase PPW for homogeneous multi-core processors. Also Das et al., [19] shows that at the system level, task scheduling and dynamic thermal and reliability management techniques improve system lifetime, and consequently LCP. However, these techniques only target PPW and LCP, respectively. We show that hot-sparing, which is orthogonal to these techniques, can improve cost- and PPW and LCP for some applications and systems. To identify which systems and applications benefit from hot-sparing, we develop a framework. In this regard, we evaluate cost-PPW and cost-LCP design spaces for SIMT processors and we categorize applications and SIMT systems with respect to the type of redundancy that they utilize the best. Likewise what we did for performance, we develop expected PPW, $E[PPW]$, and expected LCP, $E[LCP]$, metrics to evaluate the PPW and LCP of designs with hot spares, respectively. Also, to make design space exploration tractable for our framework, we introduce $\hat{E}_s[PPW]$ and $\hat{E}_s[LCP]$ as estimation methods to estimate $E[PPW]$, and $E[LCP]$, which lead to 6.3% and 15.3% average relative error, respectively. To explore cost-PPW and cost-LCP design spaces efficiently, we adopt a guided design space exploration (DSE) algorithm. Our DSE algorithm uses an artificial neural network (ANN) regression model, adopted for our cost-PPW optimization prob-

lem. We use this DSE algorithm in our framework to find optimal solutions. Our DSE algorithm reduces the design exploration time to one-sixth, while finding Pareto-Optimal fronts (POF) three times closer to the real POF than conventional methods such as ReSPIR [20]. Lastly, we show that by utilizing hot-sparing some applications and systems experience more than 16% and 75% improvement in terms of PPW and LCP, while their cost decreases as well.

1.4 Related Publications

This dissertation has resulted in several publications, a partial list of which and how they relate to the chapters of this thesis is provided here.

1. [11] S. H. Mozafari, K. Skadron, B. H. Meyer, "Yield-aware Performance-Cost Characterization for Multi-Core SIMT," Proceedings of the 25th edition on Great Lakes Symposium on VLSI, pp. 237-240.
   In this paper, we propose spare lane sharing, which reduces the cost of multi-core SIMT systems by allowing one of two neighboring cores to make use of a redundant lane if necessary. We have evaluated the cost-performance trade-offs of core-, lane-, and shared-lane-sparing under a variety of benchmarks, and found that for nearly all applications shared-lane-sparing outperforms lane-sparing, reducing cost by up to 20%. This paper is discussed in Chapter 3.

2. [2] S. H. Mozafari, and B. H. Meyer, "Hot spare components for cost-performance improvement in multi-core SIMT," 2015 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology

Systems (DFTS), pp. 53-59.

In this paper, we extend our prior investigation by exploring the performance and cost implications of allocating hot spare components in SIMT systems. As before([11]), we allocate spare cores, lanes, and shared-lanes, but this time enable these components for performance improvement when possible. To evaluate designs with hot spares, we introduce a new metric, expected performance per cost, $E[P]/C$. $E[P]/C$ captures the performance and cost of a multi-core SIMT system by considering the population of dice that results from a given defect density. For a case study system, we observe that expected performance per cost improved more than 2.5 and 1.7 times relative to systems integrating no redundancy and cold spares, respectively. This paper is discussed in Chapter 4.

3. [13] S. H. Mozafari, and B. H. Meyer, "Efficient Performance Evaluation of Multi-core SIMT Processors with Hot Redundancy," IEEE Transactions on Emerging Topics in Computing, 2018, pp. 498-510.

   In this paper, we introduce two $E[P]$ estimation techniques, $\hat{E}_m[P]$ and $\hat{E}_s[P]$. $\hat{E}_m[P]$ evaluates the m most likely configurations, and assumes the performance of all others is zero, reducing simulation by 93%. This remains computationally expensive for design space exploration when individual, detailed, simulations require hours. $\hat{E}_s[P]$ evaluates only the most likely configuration, and assumes its performance for all other configurations, reducing simulation by 98%, with no more than 2.6% error in $E[P]$, sufficient for differentiating designs along the Pareto-optimal front during design space exploration. Consequently, designers may add

redundancy, and evaluate system performance and cost, with no greater design effort than performance evaluation alone. This paper is discussed in Chapter 5.

4. [21] S. H. Mozafari, and B. H. Meyer, "Characterizing the Effectiveness of Hot Sparing on Cost and Performance-per-Watt in Application Specific SIMT," Submitted to INTEGRATION, the VLSI Journal.

   In this paper, we investigate the performance improvement of hot spares to see if it can be used to improve PPW in SIMT processors over different applications. Also, we investigate the cost and PPW implications of employing different types of hot spares in SIMT processors. Then, we study optimal solutions in the cost-PPW design space to see what kind of redundancy improves cost and PPW the most. However, since evaluating individual design points (different SIMT processor configurations with redundancy) is time consuming, we adopt a design space exploration algorithm to find near-optimal solutions without evaluating the design space exhaustively, which finds approximated optimal solutions three times better than conventional methods. We observe that hot sparing is effective for specific types of SIMT processor configurations (small and medium sized). On these configurations, it can improve PPW more than 16%, on average, for applications that experience significant performance improvement by adding hot spares (e.g., FFT and FILTER). Furthermore, we show that hot sparing's PPW improvement on these applications is comparable with the results of conventional techniques (e.g., voltage scaling) and can be utilized together with them

to more effectively improve PPW in the systems. Also, we observed that micro-architectural hot redundant resources (e.g., hot shared-spare lanes) achieve better PPW improvement than conventional architectural redundancies (e.g., hot spare cores). This paper is discussed in Chapter 6.

5. [22] S. H. Mozafari, and B. H. Meyer, "Hot-sparing for Lifetime-Chip-Performance and Cost Improvement in Application Specific SIMT Processors," Submitted to INTEGRATION, the VLSI Journal.
   In this paper, we investigate the effect of hot spares on lifetime-chip-performance (LCP) in multi-core single-instruction, multiple-thread (SIMT) processors. We observe that hot-sparing is outstandingly effective for specific types of SIMT processor configurations (small and medium systems) and applications (FFT and FILTER), while improving cost and LCP over other configurations and applications as well. For example, hot-sparing can improve LCP more than 75% compared with conventional methods (i.e., cold sparing), on average, for applications that experience significant performance improvement when adding hot spares (e.g., FFT and FILTER). In particular, micro-architectural hot redundant resources (e.g., hot spare lanes) achieve better LCP improvement than conventional architectural redundancies (e.g., hot spare cores). This paper is discussed in Chapter 7.

1.5 Thesis Outline

Chapter 2 presents related work.

Chapter 3 presents cold sparing in SIMT processors. Also, this chapter introduces shared-spare lane technique in SIMT processors and investigates the effect of adding micro-architectural redundancy on the performance of SIMT processors. Moreover, this chapter models the yield and cost of SIMT processors with redundancy.

Chapter 4 presents hot-sparing in SIMT processors. Also, this chapter introduces a new metric, $E[P]/C$ to evaluate designs with hot-sparing in terms of cost and performance. Moreover, this chapter introduces an estimation method for calculating $E[P]$, $\hat{E}_m[P]$.

Chapter 5 presents another novel estimation method for estimating expected performance in the presence of defects in SIMT processors, $\hat{E}_s[P]$. Also, it measures the accuracy of $\hat{E}_s[P]$ over a wide range of SIMT systems to see if it is accurate enough to be used to distinguish the designs in the design space of cost and performance. Chapter 6 presents a framework to utilize hot-sparing in SIMT processors to optimize cost and PPW. This chapter introduces an adopted design space exploration algorithm that finds near optimal solutions in cost and PPW design space efficiently.

Chapter 7 presents the utilization of hot-sparing to address cost and LCP in SIMT processors. This chapter introduces a framework to find optimal designs in terms of cost and LCP.

Chapter 8 concludes the thesis highlighting the main findings, and outlining areas for future improvement.

Chapter 2

Related Work

In this chapter, we compare our solutions to address the already mentioned research problems (refer to Chapter 1) with past work. In this regard, we show the importance of SIMT architecture as a multi-core processor by showing the range of its usage in these days applications. Afterward, we show how others utilized redundancy (cold and hot) to address cost, performance, performance-per-watt, and lifetime-chip-performance in multi-core processors, and what their differences are from our solutions. Then, we show that how related work addresses the complication of calculating the performance of a system with hot-sparing and how our solution differentiates from them. Also, we compare our introduced design space exploration method with existing ones in the research criteria.

2.1 Application Specific SIMT Processors

Unlike conventional high-performance computing systems, multi-core single-instruction, multiple-thread (SIMT) systems, popularized in general-purpose graphics processing units (GPGPUs) as well as embedded processors [10]. SIMT is another form of single-instruction, multiple-data (SIMD) architecture executing multiple scalar threads instead of multiple data. Application specific SIMT processors have emerged in a wide range of practical systems: 1) embedded processors into mobile devices, such as smart-phones to accelerate compute-intensive multimedia applications including image processing [23, 24], video processing [25], and 3-D graphics [26], 2) application specific high-performance systems. For example, Amazon Web Service provides computing resources for machine learning algorithms [27], and 3) application specific edge computing workloads that a majority of them include machine learning or signal processing algorithms [28].

SIMT processors are especially well-suited to incorporating redundancy at multiple granularities: their architecture already utilizes component replication at multiple levels of design abstraction for the purpose of performance improvement. At the system level, SIMT cores consist of multiple thread-parallel cores. Each core further consists of a single front-end unit and multiple, data-parallel processing elements (lanes) [1]; because hardware and software support already exists, this replication can be leveraged in a straightforward manner to improve system performance and cost.

2.2 Cold Sparing

Cold redundancy is a well-studied technique for improving yield and reliability, and has been previously employed at a variety of levels of design abstraction [4]. Cold sparing is a kind of redundancy that cannot contribute to improve a system's performance in the field since they only ever replace faulty components, and are otherwise disables. In the context of circuits, repetitive structures have been leveraged for improving yield in the context of cold sparing in such as memory cells and programmable logic arrays (PLAs) [3], and field programmable gate arrays (FPGAs) [29]. Prior work has attempted to reduce the cost of redundancy by designing a single circuit to be able to replace any of a number of different circuits [30].

At the micro-architecture and system level, Premkishore et al. [14] and Schuchman and Vijaykumar [15] have investigated the effect of core sparing and micro-architecture redundancy techniques on yield improvement in general purpose multi-core processors (i.e. cold sparing). In both cases, the authors observed that as the number of cores grows, the relative benefit of core-level redundancy also increases: core-level redundancy becomes both more effective (as one spare can cover for an increasing number of components) and more efficient (lower relative overhead is required). However, their developed yield model is not applicable for SIMT processors (the architecture and the components of general purpose multi-core processors are different from SIMT processors).

Meyer et al. [31] investigated yield improvement for network-on-chip-based multiprocessor systems-on-chips. Under the assumption that software tasks

can be re-assigned and re-scheduled when components are defective, spare computational and storage capacity can significantly improve yield, especially during manufacturing process ramp-up, which is characterized by particularly high yield losses. In our work (which is discussed in Chapter 3), 1) we specifically target multi-core SIMT systems, which have different constraints, and considers multiple granularities of redundancy, rather than the single approach, slack allocation. 2) Also, we consider performance overheads due to adding redundancies to take into account an important practical implication (performance degradation by adding redundancies).

As a consequence of the growing interest in system-level redundancy, a variety of authors proposed modeling or analysis frameworks for evaluating the effectiveness of various redundancy strategies in the multi-core era. Markovsky and Wawrzynek [7] approach the problem analytically with comparison of over-engineering, circuit-level redundancy, and system-level redundancy, and confirm that core-level redundancy is the most cost-effective as cores proliferate. Lower-level techniques incur too great an overhead to cover too few possible defects. Shamshiri et al. [32] develop a yield model for core sparing in multi-core systems, and observe that spare cores make burn-in unnecessary. Our work develops yield models specifically for multi-core SIMT processors, considering their unique opportunity for yield improvement via adding cold redundancies in architectural as well as micro-architectural granularity levels. Also, we introduce a new kind of micro-architectural redundancy in these systems, shared-spare-lane.

2.3 Hot Sparing

Prior work has explored the allocation of cold spare micro-architectural units [16, 15]. The cost of micro-architectural redundancy, however, is high, since the redundancy can only cover defects in a limited number of components. For systems with many cores, core-replication generally performs better [12]. Core sparing, however, is not a panacea: recent work has also shown that when multi- and many-core systems share redundant components, cost is reduced relative to conventional resource sparing methods [12, 33, 11]. We expand upon this work by allocating hot-spare cores as well as hot-spare-, and hot-shared-spare-subcomponents. Hot sparing is a kind of redundnacy that is used to address yield in manufacturing process, otherwise it can be turned on in the field to improve a system's performance.

Prior work has explored how to quantify performance in the context of defects [16, 15, 12]. Like our work, these all calculated system performance by simulating all possible systems with defects and weighting their contribution to expected performance by the likelihood of each configuration occurring. In our work, which requires detailed multi-core performance simulation, such an approach is intractable: Therefore, we introduce an approximation for expected performance in order to reduce evaluation complexity with an acceptable loss in accuracy.

Recent work has also investigated the utilization of redundancy to facilitate graceful performance degradation and extend processor lifetime [34, 15, 35, 36]. These approaches substitute defective components with redundant ones, or scavenge for functional subcomponents within failed components, extending

the useful lifetime of the system while reducing the negative performance impact of defects and failures. Such an approach to redundancy in general purpose architecture, however, is very costly in terms of performance. We show that this is not the case for SIMT architecture; to the best of our knowledge, ours is the first research to evaluate the opportunity to improve both the performance and yield of SIMT systems using hot redundant resources.

2.4 Efficient Performance Evaluation for SIMT with Hot-Sparing

Prior work has quantified performance in the context of defects [16, 12, 15, 8]. These all consider all possible systems with defects and weight their contribution to expected performance by the likelihood of each configuration occurring; however, unlike our work, they employ performance estimates for each system. Our work requires detailed multi-core performance simulation, and evaluating each possible system is intractable: in one case, evaluating a single configuration and all derivative designs for each of eight benchmarks requires 792 hours of simulation, making design space exploration impossible.

We have previously proposed to address this problem by limiting simulations to the first three most likely configurations [2]. We observe, in practice, however, that achieving sufficient accuracy to differentiate Pareto-optimal designs still requires a significant amount of simulation, and a different number of evaluations, benchmark to benchmark. In this work, we therefore detail a new approach which requires just a single evaluation per design: in this case, calculating the expected performance of a system is not any more expensive

than traditional performance evaluation, while it achieves sufficient accuracy for design space exploration and identifying the true POF designs.

2.5 Hot-Sparing for Performance-per-Watt Improvement

Great attention has been paid to energy-efficient computing in multi-core processors since, in new technologies, performance and power do not scale similarly [37]. Ghasemazar et al., [17] and Hanumaiah et al., [18] showed that by using system-level techniques (DVFS, task allocation, and task migration) we can significantly increase PPW for homogeneous multi-core processors. They utilized the existing redundancy (timing slack) in a system to reduce its power without changing the system's performance (throughput). Also, Ejlali et al., [38] used an online energy-management technique for lock-step processors. They exploited dynamic timing slack at run time and utilized DVFS and dynamic power management (DPM) techniques to more effectively reduce the system's energy consumption. Moreover, Leng et al., [39] applied fine grained DVFS and clock gating techniques to GPGPUs to more effectively reduce their power consumption of GPGPU processors without reducing its performance significantly (increasing PPW).

However, none of these works addresses cost, and on the other hand, they rely on the presence of timing slack to optimize energy consumption. Hot-sparing even can reduce energy consumption of applications with short-balanced phases since it does not rely on timing slack to improve PPW. Moreover, note that hot-sparing can be utilized with other energy saving techniques

(such as DVFS) since they are applied at different abstraction levels.

2.6 Adopted Design Space Exploration Algorithms

Design space exploration (DSE) algorithms are, generally, utilized to optimize multiprocessor systems by efficiently exploring a multiprocessor design space and finding optimal solutions. These algorithms can be totally based on prediction models. If so, they decrease the complexity of design space exploration by predicting the design space (e.g., response surface modeling), rather than investigating it exhaustively [40]. However, response surface model (RSM) based methods suffer from prediction error. The prediction error could easily lead them to detect optimal solutions with a considerable distance to the real optimal solutions in a design space. On the other hand, some DSE algorithms, like ReSPIR [20], use a combination of design of experiments (DoEs) and response surface modeling (RSM) techniques to find better optimized design points by performing a less number of simulations. ReSPIR simulates a part of a design space to perform RSM, then it uses the RSM to find optimal solutions. After that, it iteratively simulates new design points and adds them to the training set of the RSM to make it more accurate. Meanwhile the algorithm tracks the optimal solutions improvements. Our DSE approach is similar to this work: we use RSM to predict where the optimal solutions are located in a design space. However, in our algorithm we do not just rely on the optimal solution set that is being detected by RSM, instead we investigate the design points near to the predicted optimal solution set to find better optimal solutions, while we use a different approach to select RSM model as well.

2.7 Hot-Sparing for Lifetime-Chip-Performance Improvement

Many techniques, at different abstraction levels, have also been developed to improve LCP in multi-core processors. Several techniques have taken advantage of existing micro-architectural redundancy to improve lifetime or yield [41]. At the system level, task scheduling and dynamic thermal and reliability management (DTM, and DRM) techniques improve system lifetime, and consequently LCP, [19]. Software techniques exploit parallelism (thread- and/or instruction-level) to improve performance, and consequently, LCP [42]. These techniques only target lifetime or LCP; hot-sparing, which is orthogonal to dynamic thermal or lifetime management, can improve cost as well.

Chapter 3

Yield-Aware Performance-Cost Optimization in SIMT

In this chapter, we investigate the cost and performance of SIMT systems in the presence of cold-sparing. In this regard, we introduce a new type of cold-sparing (shared spare lane), and its yield and cost models. Then, we investigate the design space of cost and performance for systems with cold redundancy to understand which type of redundancy optimal solutions utilize (cold- core, lane, or shared-spare-lane).

CMOS technology is aggressively scaling. Consequently, designers face higher defect rates and detrimental effects of process variation [4]. Therefore, devices are more likely to experience early performance degradation, and even failure in the field due to systematic and random defects [3, 4]. Redundancy is a well-studied technique for improving yield and reliability, and has been previously employed at a variety of levels of design abstraction [4]. At the

micro-architecture and system level, Premkishore et al. [14] and Schuchman and Vijaykumar [15] have investigated the effect of core sparing and micro-architecture redundancy techniques on yield improvement in general purpose multi-core processors (i.e. cold sparing). However, their yield and cost models are not applicable for single-instruction multiple-thread (SIMT) processors which their architecture is completely different [43]. To address yield loss in SIMT processors, we investigate the opportunity to reduce manufacturing costs in the presence of random defects by employing multi-granularity redundancy in SIMT architecture, while we consider their performance overhead. In this regard, we introduce yield and cost models in SIMT processors in the presence of redundancy.

Application specific SIMT architectures present a unique opportunity for yield improvement. As different applications exhibit differences in parallelism, different configurations (e.g., data cache size, number of cores, number of lanes per core) are expected to strike the best performance-cost trade-offs. Systems with many narrow cores may be performance-cost optimal for predominately thread-parallel applications; data-parallel applications demand wider cores with many lanes. As the relative mix of components (cores vs. lanes) changes, the most cost-effective strategy for improving yield is also expected to change (redundant cores vs. redundant lanes). While systems with many narrow cores clearly benefit the most from core sparing, and systems with a few wide cores benefit the most from lane sparing, a number of applications and systems fall in between: not enough cores are present to amortize the cost of a spare, and application-specific performance-optimal cores are not wide

enough to absorb the cost of an extra lane. In this context, we propose the allocation of a shared spare lane (SSL) which can be used by either of two neighboring cores to replace a defective lane. Moreover, tools are needed to assist designers with determining when and what sort of redundancy to apply to optimize such systems.

However, identifying the best trade-off is computationally expensive, due primarily to the size of the design space and the complexity of performance estimation. Consequently, we employ a prediction model, genetically programmed response surfaces (GPRS) [44] to estimate performance and facilitate the efficient identification of cost-performance Pareto-optimal front systems. GPRS uses genetic programming to fit a surface to samples given a set of configuration parameters. Moreover, we employ average distance from reference set (ADRS) [45] as a metric to estimate GPRS accuracy in predicting the Pareto-optimal set in the performance-cost design space. We observed that by simulating 15% of the design space and using GPRS, on average, we can correctly predict about 23% of Pareto-optimal points and reach to 0.02% ADRS-distance of design space Pareto-optimal front, which is a very small distance. Note that utilizing GPRS tool to explore performance-cost design space, and its corresponding results have not been published in our work ([11]) as contributions.

3.1 Multi-Granularity SIMT Redundancy

A multi-core SIMT processor is depicted in Figure 3.1. Each core, which is referred as a streaming multi-processor (SM) in NVIDIA G80 terminology [10],

consists of a unified front-end unit, including L1 instruction cache (I$), hardware thread contexts, and instruction decoder, processing elements (lanes), and a back-end write-back unit. Each lane, consists of a bank of register file (RF), arithmetic and logic unit (ALU), and a bank of L1 data cache (D$). Note that RF and D$ are banked in SIMT architecture [43, 10]. In NVIDIA G80 terminology, a lane is consist of a streaming processor (SP), and a bank of D$ or shared-memory [10]. At the system level, cores communicate with L2 caches through a crossbar [43], which is connected to RAM via a bus.

SIMT processors are designed to simultaneously take advantage of thread-level and data-level parallelism (TLP and DLP, respectively). Not only can different cores can be used to execute different threads of the same application, or even different applications (in this work, we focus on single application workloads), but each core can execute the same sequence of instructions on multiple data elements at the same time [1, 43].

Different parallel applications have different working set sizes and mixes of TLP and DLP; therefore, different configurations result in optimal performance [46]. Applications with greater TLP require more cores; applications with greater DLP require more lanes. Under a particular cost constraint, determining the best ratio of cores to lanes is a complex, and further depends on work set size (D$ requirement) and other factors. This variability in the design space for multi-core SIMT systems has an important consequence for yield enhancement: systems optimized for different applications may call for different redundancy. Fortunately, both the programming model and physical design of SIMT architectures make implementing redundancy at each of these

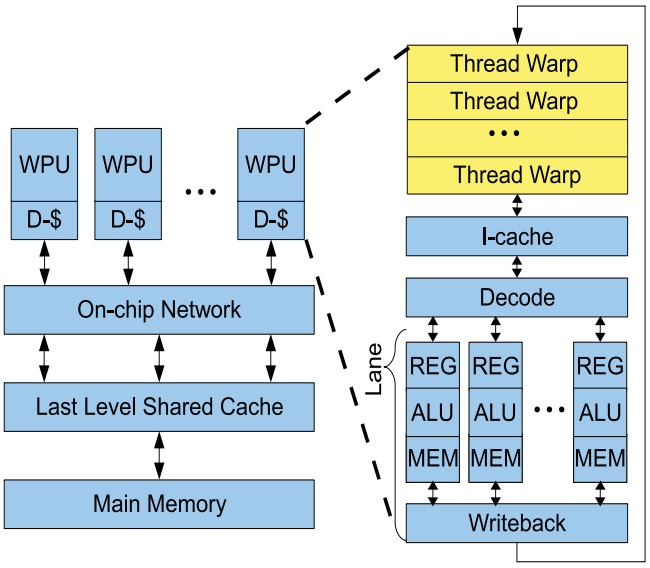


Fig. 3.1: SIMT architecture [1].

granularities both relatively easy and particularly beneficial, as compared with equivalent strategies in general purpose designs.

3.1.1 System Yield

Based on the system architecture in Figure 3.2, the system yield y_{sys} is the product of the yield of different groups of components, namely the the cores, y_{cores}, the L2 caches, y_{L2}, and the crossbar, $y_{crossbar}$:

$$y_{sys} = y_{cores} \times y_{L2} \times y_{crossbar}. \tag{3.1}$$

3.1.2 Core Sparing

One straightforward way to improve system yield in homogeneous multi-processors is to add spare cores. When a core is defective, a redundant core

can be substituted for it. System performance variation due to the topology of cores within a die can be minimized by using NoC-Virtualization [47]. In the context of a crossbar-based system, the only performance penalty is due to the increased size of the crossbar.

While a portion of core lanes can be individually covered with finer-grained redundancy, much of it cannot; we assume that hardware thread contexts and write-back logic cannot be easily protected, and so therefore are not. Consequently, a defect in one of these units results in core failure.

However, as the number of cores in the system increases, the relative overhead of a single redundant core goes down [12]. As a result, applications that perform optimally with many narrow cores benefit the most from core sparing. Given a system that requires m operational cores and integrates n spares, where each core has yield y_{core}, the yield of the set of cores y_{cores} is calculated with binomial distribution:

$$y_{cores} = \sum_{i=m}^{m+n} \binom{m+n}{i} (1 - y_{core})^{m+n-i} (y_{core})^{i}. \tag{3.2}$$

For the sake of simplicity, hereafter the binomial distribution will be denoted $Binom(y, b, s)$ for a system with b components and s spares, where each component has a yield of y.

3.1.3 Lane Sparing

Core yield is dependent on the yield of its components. For the purposes of this paper, we divide a core into (a) its lanes, (b) its L1 cache, and (c)

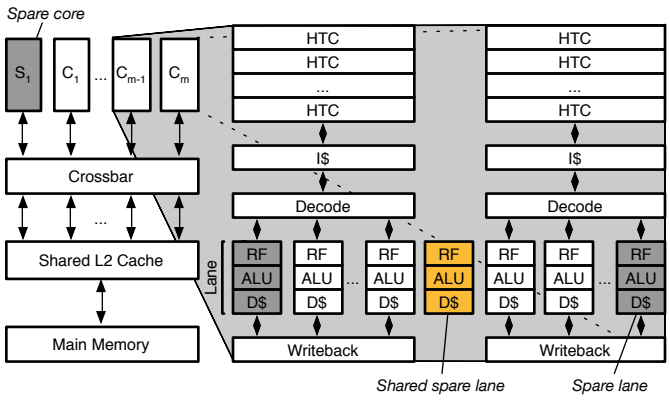


Fig. 3.2: The physical design of SIMT architecture allows them to incorporate coarse- and fine-grained redundancies.

everything else (skeleton):

$$y_{core} = y_{lanes} \times y_{L1} \times y_{skeleton}. \tag{3.3}$$

The yield of individual SIMT cores can be improved by allocating spare lanes. SIMT processors are distinguished from general-purpose ones by their two levels of granularity: cores, and per-core lanes. Each core is a multi-processor in its own right, but its redundant functional units (decoder, and write-back) have been removed to improve power and area efficiency for applications that exhibit data level parallelism. Just as homogeneity at the system level makes core sparing effective, homogeneity within cores makes lane sparing effective.

A spare lane can be integrated in a core as depicted in Figure 3.2. A core without lane sparing is illustrated in white. When cold spare lane (gray) is integrated, if one of the main lanes is defective, the core may continue to function. Swapping in a spare lane only requires the addition of multiplexors in

decode and write-back to direct control signals to the spare and away from the faulty lane. Note that the spare lane does not improve performance when all four lanes are functional; an investigation of the trade-offs under hot-sparing is the subject of Chapter 4.

Spare lanes effectively improve yield in wide cores with many lanes. The greater the number of lanes in a core, the higher the proportion of area dedicated to lanes, and the lower the relative overhead of a single spare lane. Given a system where each core requires k operational lanes and integrates l spares, and the yield of each lane is y_{lane}, the yield of the lanes can, like the yield of cores, be calculated with the binomial distribution: $y_{lanes} = Binom(y_{lane}, k, l)$.

3.1.4 Shared Spare Lane

The homogeneity of lanes, and the significant role they play in SIMT architecture, presents an additional, unique opportunity for low-cost yield enhancement: sharing a spare lane between two cores, illustrated in Figure 3.2. When the two cores share a spare lane, if either the white cores have a defective lane, shared-lane (depicted in orange) and the corresponding decode and write-back logic can be employed; this only prevents core failure when there is not more than one defective lane in both cores.

The performance penalty of accessing redundant resources (e.g., pipeline stage) in general purpose multi-cores is high [36]. However, due to hierarchical design of SIMTs, we hypothesize that a shared spare lane can be included in either core with only marginally greater performance loss than if the spare were dedicated to a single core. On the other hand, where yield improvement

is concerned, the shared spare lane improves yield and reduces costs when spare lanes may otherwise be too costly; from a redundancy perspective, a shared spare lane covers twice the lanes (and has half the relative overhead) compared with a spare lane.

Calculating the yield when using shared spare lanes is rather more complex than previous cases; rather than derive an exact analytical formula, we estimate yield as follows. The use of a shared spare lane couples the yield of the two cores. There are two cases in which both cores that share a lane are functional:

1. Neither core needs the shared spare lane ($y_{lanes\_ssl_1}$),
2. One core needs the shared spare lane ($2 \cdot y_{lanes\_ssl_2}$).

When neither core needs the shared spare lane, the yield of the set of lanes (given k lanes and l spares) is simply:

$$y_{lanes\_ssl_1} = Binom(y_{lane}, k, l) \times Binom(y_{lane}, k, l). \tag{3.4}$$

A pair of cores with one core suffering from $l + 1$ defective lanes only survives if the shared spare lane is available, i.e., if it both functional (not itself defective) and not needed by the other core. In this case, the yield is:

$$y_{lanes\_ssl_2} = y_{lane} \times \binom{k+l}{l+1} (1 - y_{lane})^{l+1} (y_{lane})^{k-1} \times Binom(y_{lane}, k, l). \tag{3.5}$$

Symmetry in the system means there are two ways to have a core that requires the shared spare lane. The total yield of the pair of cores is therefore

$y_{lanes\_ssl_p} = y_{lanes\_ssl_1} + 2 \cdot y_{lanes\_ssl_2}$. As the cores are identical, the yield of a single core's lanes can be estimated as $y_{lanes\_ssl} = \sqrt{y_{lanes\_ssl_p}}$.

When spare cores are not present, this model is exact. However, error is introduced when spare cores are available. In this case, the system may yield despite the failure of one of a pair of a cores sharing a spare lane. We approximate the yield with an even number of spare cores as $y_{cores} = Binom(y_{core_{ssl}}, m, n)$, where $y_{core_{ssl}}$ is the yield of a core with a shared spare lane. If an odd number of spare cores are allocated and each but the last has a shared spare lane,

$$y_{cores} = Binom(y_{core_{ssl}}, m, n)(1 - y_{core}) + \sum_{i=m}^{m+n+1} \binom{m+n}{i-1} (1 - y_{core_{ssl}})^{m+n-(i-1)} (y_{core_{ssl}})^{(i-1)} (y_{core}). \tag{3.6}$$

The introduced error isn't significant, and we observe in general that combinations of redundancy are too expensive to fall on the Pareto-optimal front.

3.1.5 Model Validation

To verify the accuracy of the above yield formulas, we developed a Monte Carlo Simulation (MCS) framework. We considered 100 large configurations. We divide this set into four equal subsets in a way that three of them utilize only one kind of redundancy (core-, lane-, or shared-lane-sparing), and the last one has no redundancy. Then, we used the MCS to calculate their yield under unrealistically high defect densities ($0.7/cm^2$), in order to exaggerate any potential error in the model. 100K MCS samples were collected in each

case of defect regimes (e.g., one failed lane per core). On average, the relative difference in yield values between MCS and the analytical result is 0.21%, with a 95% confidence interval of 0.76%.

3.2 Experimental Setup

In order to (i) determine what form of redundancy is most applicable given a particular (a) configuration and (b) application, and (ii) evaluate GPRS for the purpose of predicting the performance of SIMT configurations in the context of identifying the Pareto-optimal front, we performed two sets of experiments. First, we conducted exhaustive performance and cost evaluation of a wide variety of multi-core SIMT systems on a set of diverse benchmarks. Second, we performed a series of training and testing experiments using GPRS.

3.2.1 SIMT Configurations and Benchmarks

We used MV5 [43] to simulate the performance of SIMT configurations. For each multi-core configuration, we fixed a number of architectural parameters; these are consistent with the experimental environment in [46]. The configurations we considered varied the number of cores N cores, $N \in \{1, 2, 4, 6, \ldots, 20\}$, the number of lanes per core (SIMT width) W and hardware threads per core (SIMT depth) D, $W, D \in \{1, 2, 4, 8, \ldots, 64\}$ and L1 data cache size S, $S \in \{16, 32, 64\}KB$. From the parameter space defined above, we selected all configurations with die area within [50, 250] mm^2.

We selected benchmarks from several suites: Minebench [48], SPLASH-2 [49], and Rodinia [50]. The set we selected, Fast Fourier Transform (FFT),

Filter Edge Detection (Filter), Thermal Simulation (HotSpot), LU Decomposition (LU), Merge Sort (MergeSort), Shortest Path (ShortestPath), KMeans Clustering (KMeans), and SVM Learning (SVM), have been previously used in the literature to evaluate multi-core SIMT performance [43].

3.2.2 Cost Model

The cost of a chip or die C_{chip} is a function of the cost of a wafer C_{wafer}, the number of dice per wafer, and the yield of the die, y_{sys} [51]:

$$C_{chip} = \frac{C_{wafer}/(Dice/Wafer)}{y_{sys}}. \tag{3.7}$$

The number of dice per wafer can be estimated as:

$$Dice/Wafer = \frac{\pi \times (Radius_{wafer})^2}{Area_{die}} - \frac{\pi \times Radius_{wafer}}{\sqrt[2]{\frac{Area_{die}}{2}}}. \tag{3.8}$$

We assume a wafer radius of $150mm$, a wafer yield of 1, and that wafers cost \$3000 each. Note that in Eq. 3.7 we do not include *test cost* since 1) in this research we focus on on *manufacturing cost*. 2) Adding repetitive components such as lanes and cores as redundancy does not increase *test cost*. This is the case, as there are methods that do testing concurrently on repetitive elements in multi-core processors to prevent any increase in *test time* [53, 54]. 3) As Shamshiri et al. [52] mentioned, we can escape *burn-in* and detailed test procedures in multi-core processors and incorporate *test cost* into *service cost* when we have redundancy in the systems and yield is high enough ($> 95\%$). On the other hand, *service cost* compare to *manufacturing cost* would be

negligible when yield is high in multi-core processors [32]. In other words, our analysis shows that *test cost* would be a constant term in overall cost when we add cores and lanes as redundancy, and therefore, not adding *test cost* and *service cost* to *manufacturing cost* as C_{chip} (refer to Eq. 3.7) would not change any of the conclusions that we will draw.

y_{sys} is calculated as described in Section 3.1.1. To calculate the yield of individual components, such as lanes, we adopt the negative binomial yield model, a function of three parameters, the clustering parameter (α), defect density (λ_b), and block area (A_b) [55, 56]:

$$y_b = \left(1 + \frac{\lambda_b \times A_b}{\alpha}\right)^{-\alpha}. \tag{3.9}$$

Smaller α results in stronger defect clustering and vice versa, a function of the process technology. We assume $\alpha = 4$ in $65nm$ manufacturing technology [51].

To estimate chip area we measured the area of functional units based on a die photo of an AMD Opteron processor. As this processor is fabricated in 130 nm, we used a scaling factor of 0.7 per generation to scale the processor to $65nm$.

3.2.3 Genetically Programmed Response Surfaces

Performance simulation is the clear bottleneck in the design space exploration of SIMT architectures, requiring two hours on average using a $2.8GHz$ Intel Core i5 with 24 GB of RAM. We have therefore evaluated genetically programmed response surfaces (GPRS) [44] for the performance estimation

of SIMT configurations. GPRS uses genetic programming to fit a surface (the objective function) to samples (training data) given a set of configuration parameters. In our case, GPRS requires an input training set: SIMT configurations that vary with respect to the targeted parameter set (number of cores, number of lanes per core, and L1 D$), and the resulting execution latency (determined with MV5). It returns an analytical model relating configuration parameters and execution latency.

To explore the trade-off between training time (a function of the size of the input set of configurations), and prediction accuracy, we experimented with using different fractions of the design space as input, {0.1, 0.15, 0.20, 0.25}. The makeup of the input set affects the predictive strength of the resulting surface. We used the Audze-Eglais [57] method, suggested by the creators of GPRS, to identify appropriate input points.

3.3 Results

We began with experiments to identify the set performance-cost Pareto-optimal designs for each benchmark. First, we performed exhaustive performance simulation, considering an average of 850 configurations for each benchmark. The simulated configurations were not allocated redundancy. Second, we added redundancy to all configurations and then selected those on the performance-cost Pareto-optimal front (POF) for each benchmark, and in the average performance case.

Cost reduction results for the average performance case is illustrated in Figure 3.3. Designs have been binned by area on the x-axis; average cost re-

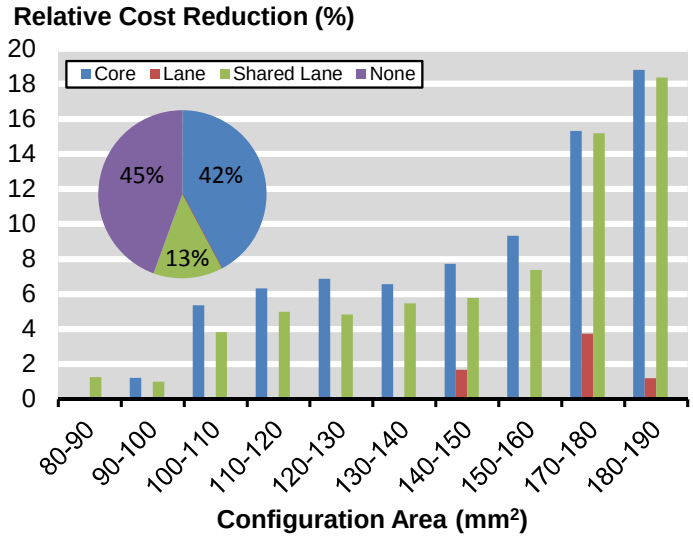


Fig. 3.3: Relative cost reduction when designing for average performance (across all benchmarks).

duction for designs in each bin are then plotted on the y-axis. Each series, *Core*, *Lane*, and *Shared Lane*, plot the set of Pareto-optimal designs implementing that sort of redundancy. The pie chart then plots the fraction of designs of each type represented in the global Pareto-optimal set, as well as what fraction utilize no redundancy at all.

We observe in the average case that redundancy does not improve yield inexpensively enough to reduce the die cost of performance-cost Pareto-optimal configurations until processors are larger than 82 mm^2. While some low cost designs, 80 to 100 mm^2 experience minor cost reduction with shared spare lanes, in the average case core sparing dominates, reducing costs by over 18% for 180-190 mm^2 configurations.

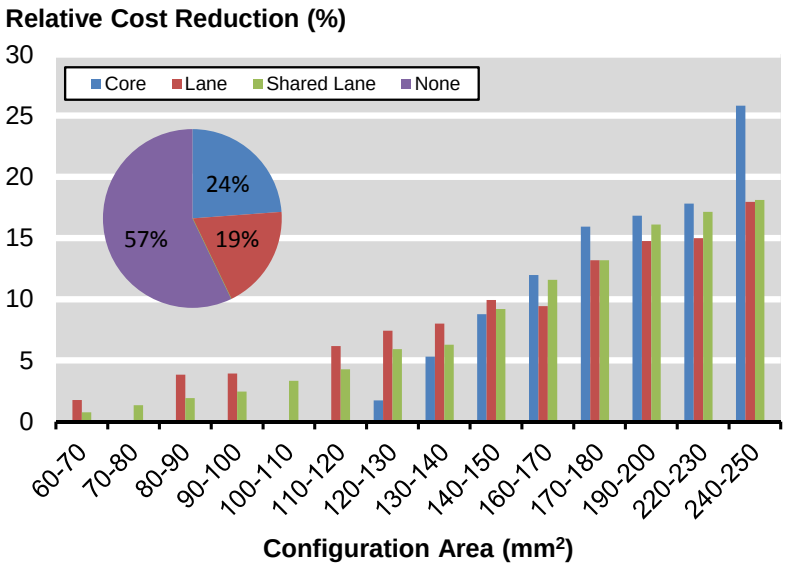


Fig. 3.4: Relative cost reduction for Filter. Lane and core sparing dominate when redundancy reduces cost.

3.3.1 Application-specific Results

While on average, spare cores reduce cost the most, individual applications often have very different needs. For example, consider the results for Filter in Figure 3.4. Redundancy begins reducing costs at $65mm^2$; lane *sparing* results in the greatest cost reduction for designs under $145mm^2$. Filter benefits from much larger configurations along the POF than other benchmarks. For large designs, core sparing dominates, with cost reduction peaking at 25% for a 248 mm^2 design. The global Pareto-optimal systems in this case mostly enjoys no redundancy while core sparing is a better regime than shared and spare lane ones.

Alternatively, consider the results for KMeans, illustrated in Figure 3.5. This benchmark uses shared lane sparing until the area exceeds 90 mm^2, and

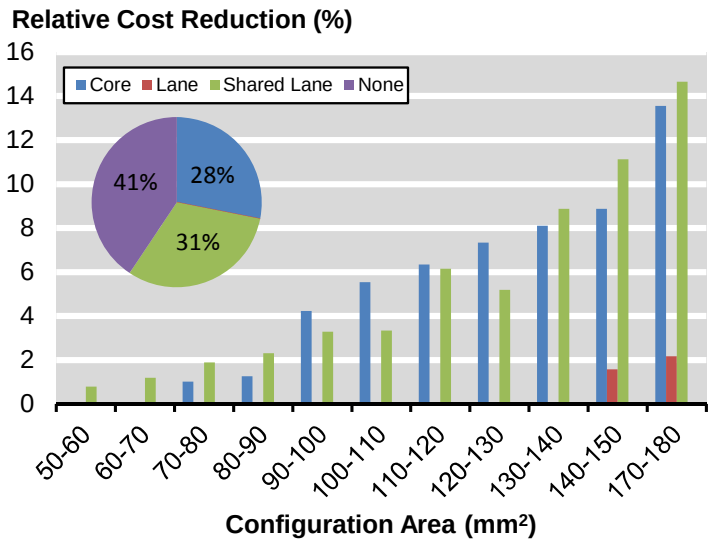


Fig. 3.5: Relative cost reduction for KMeans. Spare lane sharing and core sparing dominate in this case.

again when designs are larger than 130 mm^2, peaking at over 14% cost reduction. The Pareto-optimal systems, in this case, mostly utilize shared-spare-lane, which shows the result of having narrow cores in the POF set.

FFT, MergeSort and ShortestPath benefit the most from core sparing: the Pareto-optimal systems in employ core sparing almost exclusively, which reduces costs up to 8%, 9%, and 23% for designs at 145, 142, and 223 mm^2, respectively. On the other hand, spare lane sharing is more important for HotSpot and LU. For HotSpot, 59% of Pareto-optimal designs implementing redundancy use spare lane sharing, reducing costs up to 5% for systems in the 90-100 mm^2 range. For LU, it's 49%, with a peak cost reduction of 21%. SVM has only two Pareto-optimal points, one of which uses a spare core.

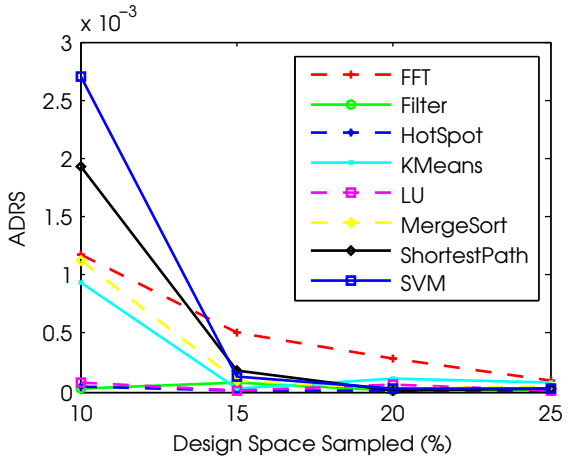


Fig. 3.6: ADRS results for different fraction of data set samples over different benchmarks.

3.3.2 Pareto-optimal Front Prediction

Next, we experimented with GPRS to evaluate its suitability for identifying Pareto-optimal SIMT configurations; in our case, exhaustive simulation required 800 hours of computation to simulate the fastest benchmark. The computational cost of GPRS increases with the size of the training set used to fit the surface. In our case, GPRS required, on average, 5.3, 11.8, 25.6, and 43.4 hours (not including simulation time) for input sets amounting to 10%, 15%, 20%, and 25% of the parameter space respectively running on Intel Core i7 Intel with 8GB of RAM, a 10 to 3.2 times speedup.

However, GPRS becomes more accurate the larger the training set. To evaluate the accuracy of GPRS, for each combination of sample size and benchmark we compared the predicted Pareto-optimal front (PPOF) with the reference Pareto-optimal front (R) and that generated by randomly sampling the design space (RaPOF). We compared the PPOF and R using the *average*

distance from reference (ADRS), a normalized-distance-based accuracy measure [58, 59]. Our results are summarized in Figure 3.6.

Compared with RaPOF (not illustrated), GPRS performs poorly for FFT, Filter, LU, MergeSort and SVM when training with just 10% of the design space. We observe in Figure 3.6 that as the input sample size increases to 15%, however, there is an average 0.004 improvement in ADRS for these benchmarks; at 15%, GPRS achieves an ADRS of 0.0002 on average across all benchmarks. Only incremental improvement is possible sampling up to 20% of the design space.

3.4 Conclusion

We introduced a new redundancy method in SIMT Multi-Core processors which is called shared spare lane. Then, we presented a yield formula to address that new redundancy method. After that, we evaluated that new redundancy technique to see how effective this would be in terms of cost reduction. We showed that, in some configurations and applications, shared spare lanes are preferable to other kinds of redundancies, and in most of the cases when a system utilizes narrow cores shared spare lane is a better sparing regime than spare lane to reduce the manufacturing cost.

As the size of SIMT performance-cost design space is big, and evaluating a single design point in this design space is computationally expensive, it is infeasible to exhaustively explored it. Therefore, we utilized GPRS (a response surface tool) to predict the performance of SIMT systems in this design space. The accuracy of this tool is highly dependent on the size of its training set. In

this regard, we showed that what percentage of the design space should be used as the training set to have enough accuracy to identify POF points. However, providing a large enough training set, might not be feasible in some cases (e.g., when a design space is huge). Consequently, designers cannot just rely on prediction models, and need to utilize intelligent design space algorithms as well.

Chapter 4

Hot-sparing for Performance-Cost Improvement in SIMT

In this chapter, we investigate the utilization of hot sparing to improve cost and performance of SIMT systems. In this regard, we introduce architectural and micro-architectural redundancies in SIMT processors, and their yield and cost models. Also, to better compare systems with hot sparing in terms of cost and performance, we introduce a new metric (*expected performance per cost*). Then we study which type of redundancy improves systems the most in terms of *expected performance per cost*.

Most yield improvement strategies utilize redundant components only when another component fails (i.e., cold spares) [8, 60, 11]. However, this may lead to a huge waste: a considerable portion of redundant components remain unused

after fabrication [11]. Gao et al. [12], use fabricated but unused redundancies in the field to improve general purpose multi-core processors' performance (hot-sparing). However, they show that the performance overhead to utilize micro-architectural redundancies in the field is considerable in a way that it may damage the performance of systems. We show that this is not the case for SIMT processors which their architecture let designers utilize hot-sparing in different granularities to improve performance.

In this chapter, for the first time, we utilize hot-sparing in SIMT processors and we investigate the cost and performance implications of utilizing this type of redundancy in SIMT processors. Hot spares are available to increase yield (and reduce costs) when the components are defective; otherwise, they can be used to improve performance in the field.

To evaluate the performance and cost of systems with hot-sparing, we introduce a new metric $E[P]/C$. $E[P]/C$ captures the performance and cost of a multi-core SIMT system by considering the population of dice that results from a given defect density. In $E[P]/C$, $E[P]$ is the expected performance of a given configuration in the presence of defects. The performance of each possible configuration is weighted by its likelihood of occurrence, thereby accounting for the availability of hot spares to improve performance when they are not needed to replace defective components. $E[P]/C$ is similar to previously introduced metrics such as yield-adjusted throughput (YAT) [15], performance-averaged yield (PAY) [34]. However, it captures cost which is a better proxy than yield and area when we want to capture the effectiveness of redundancy in a system: while all redundancy increases area, not all redundancy improves

yield sufficiently to reduce cost.

Prior work has explored how to quantify performance in the context of defects [16, 15, 12]. Like our work, these all calculated system performance by simulating all possible systems with defects and weighting their contribution to expected performance by the likelihood of each configuration occurring. In our work, which requires detailed multi-core performance simulation, such an approach is intractable: Therefore, we introduce an approximation for $E[P]$ in order to reduce evaluation complexity with an acceptable loss in accuracy.

Starting with a baseline architecture with six cores, and 32 lanes each, we added three hot spare cores, with two lanes each. When we make the lanes of the hot spares available to replace defective lanes in the baseline cores, we observe that $E[P]/C$ improved more than 2.5 and 1.7 times relative to systems integrating no redundancy and cold spares, respectively.

4.1 Multi-Granularity Hot-Sparing in SIMT

As mentioned in Section 3.1, SIMT systems consist of components replicated at different levels of granularity (core and lane). This gives designers the opportunity to apply redundancy at the same granularities from fine- (lane) to coarse-grained (core), in order to address manufacturing defects. However, redundant units that are integrated but not utilized are wasted if left cold; we therefore propose integrating hot spare units that can be easily used to improve performance when not needed to cover defects.

We illustrate a multi-core SIMT system in Figure 4.1, implementing different sparing regimes. Three different types of cores are illustrated: 1) main

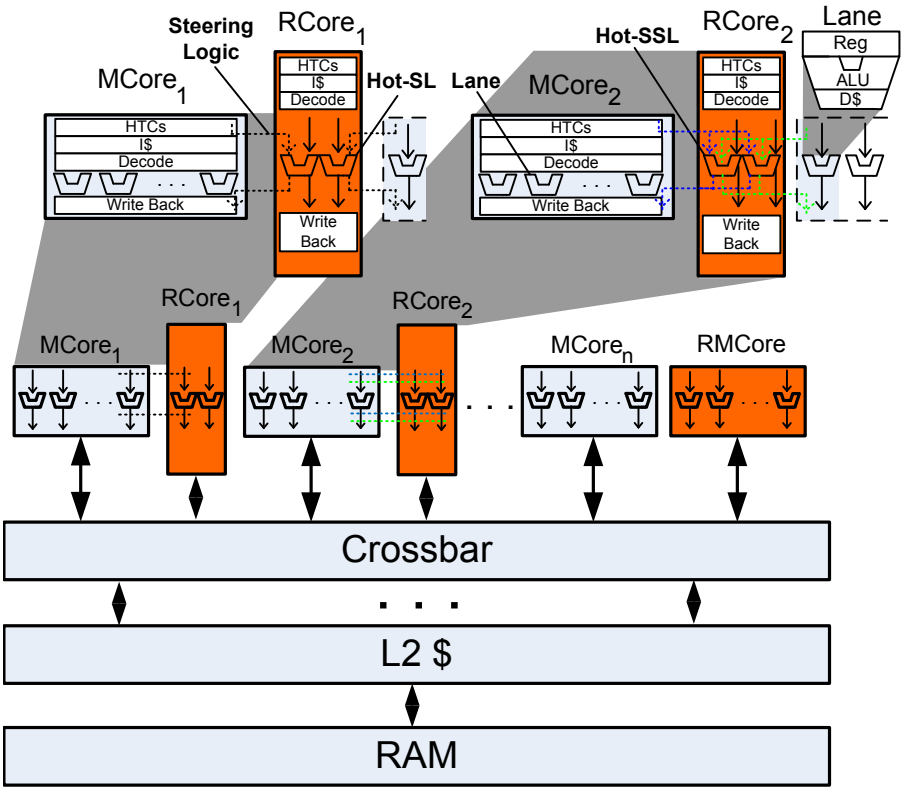


Fig. 4.1: Architecture of a multi-core SIMT system with hot spare components.

cores (MCores); 2) narrow, hot redundant cores (RCores) that make spare- and shared-spare-lanes available for use by MCores and RMCores; and, 3) wide hot redundant main cores (RMCores) for core sparing.

SIMT core and lane architectures are defined in Section 3.1. Each core (MCore, RCore, and RMCore), which is referred as a streaming multi-processor (SM) in NVIDIA G80 terminology [10], consists of a unified front-end unit, including L1 instruction cache (I$), hardware thread contexts (HTCs), and instruction decoder, processing elements (lanes), and back-end write back.

4.1.1 Redundancy Regimes

We propose allocating redundancy to salvage defective main cores (MCores) by scavenging for components in redundant cores (RCores) [36], by means of steering logic, or replacing them entirely with redundant main cores (RMCores). When not covering defects, RCores and RMCores can be used to improve performance.

We illustrate several multi-core SIMT redundancy strategies in Figure 4.1:

- *Hot spare lane (Hot-SL)*: a lane from an RCore that is dedicated to an MCore. Up to one defective lane in each of MCore_1 and MCore_2 are covered by the pair of lanes in RCore_1. Hot-SL implementation is different from Cold-SL (refer to Section 3.1.3): Hot-SLs are added in the context of RCores, while Cold-SLs are added as an extra lane in the context of MCores.

- *Hot shared spare lane (Hot-SSL)*: a lane that is not dedicated to a specific core, but can be used if needed by a small set (two, or three) of cores. Hot-SSL implementation is different from Cold-SSL (refer to Section 3.1.4): Hot-SSLs are added in the context of RCores, while Cold-SSLs are added as an extra lane between two MCores without any skeleton. Up to two defective lanes in the pair of cores MCore_2 and MCore_3 (not pictured) are covered by the pair of lanes in RCore_2. If MCore_2 has two defective lanes, and MCore_3 none, the system remains functional, while it would not if only spare lanes were available.

- *Hot spare core (Hot-SC)*: an RMCore that can completely replace an

MCore that cannot be otherwise salvaged. Integrating Hot-SC to SIMT systems is similar to Cold-SC (refer to Section 3.1.2), but with this difference that it can be utilized in the field to improve the performance of a system.

To support current practices in GPGPU programming, we limit RCores s.t. the number of active lanes is always a power of two, facilitating thread block mapping and execution without requiring changes to runtime thread management (e.g., to explicitly manage processor heterogeneity). Note that we also assume that spare unit allocation and defective unit substitution are performed at manufacturing time.

4.1.2 Expected Performance per Cost, $E[P]/C$

Performance and cost are influenced by several factors in multi-core SIMT with hot spares. The addition of hot spares can improve performance when the spares are not needed to mitigate defects, however, this is not the case for cold-sparing. We capture this effect by simulating the multi-core systems running a variety of benchmarks using the MV5 performance simulator [43]. Also, similar to cold-sparing, each type of hot redundancy, except core sparing, results in the addition of steering logic: muxes, de-muxes, and wires are needed to direct signals to/from a spare lane. These components lengthen the critical path of the core utilizing them, thereby decreasing its operating frequency. We capture this effect by synthesizing steering logic in the context of the FlexGrip GPGPU [61]. Note that since hot micro-architectural redundancies (Hot-SL, and Hot-SSL) are defined in the context of RCores, their performance overhead

is different from similar cold sparing methods (Cold-SL, and Cold-SSL) as the steering logic that connects them to MCores can result in a longer critical path and lower clock frequency. When we calculate the overall performance of a system with hot-sparing, we take into account both these performance loss and gain. System cost is also affected: area goes up with the inclusion of redundancy, but cost may decrease if the resulting improvements in yield are significant enough. Area overhead of utilizing Hot-SL, and Hot-SSL is higher than similar Cold-SL, and Cold-SSL methods as hot micro-architectural redundancy is defined in the context of a complete core (RCore), while this is not the case for cold sparing.

Given these various effects, a question arises: what is the best configuration to jointly optimize performance and cost? In order to answer this question, we propose *expected performance per cost* ($E[P]/C$), a new metric that captures the effect of hot-sparing on system performance and cost.

$E[P]/C$ is similar to previously introduced metrics such as yield-adjusted throughput (YAT) [15], performance-averaged yield (PAY) [34], and $E[P]/Area$ [8], with two consequential differences. First, area is not an appropriate proxy for cost: while all redundancy increases area, not all redundancy improves yield sufficiently to reduce cost. Second, while previous metrics evaluate the set of possible configurations resulting from defects in a given system, this is computationally intractable in our case.

Cost Model

In $E[P]/C$, the cost, C, is the fabrication cost of a die (refer to Section 3.2.2). As stated in Eq. (3.7), C_{die} is a function of the cost of a wafer (C_{wafer}), the number of dice per wafer, and the yield of the die, y_{sys}. System yield, y_{sys} in Eq. (3.8), is defined as the ratio of working ICs to the total fabricated [3], and is a function of the component yield. In this section, we adopt and extend the yield model we developed in Section 3.1. We calculate yield based on the type of redundancy that is utilized in the system (None, Hot-SC, Hot-SL, Hot-SSL); we previously observed that neither systems with more than one type of redundancy nor more than one redundant component of a given type are cost-performance cost optimal solutions [11].

For multi-core SIMT, yield is

$$y_{sys} = y_{cores} \times y_{L2} \times y_{crossbar} \times (y_{RC_{skel}} \times y_{RC_{L1}})^p, \tag{4.1}$$

where y_{cores}, y_{L2}, and $y_{crossbar}$ are the yield of the set of cores, the L2 cache, and the crossbar, respectively. RC_{skel} and RC_{L1} are RCore's skeleton (anything that is not a RCore-lane or L1 cache inside the RCore) and L1 cache yield values, respectively. p is the number of hot spare RCores in the system. The yield of the main cores (y_{cores}) is:

$$y_{cores} = Binom(y_{core}, m, n) = \sum_{i=m}^{m+n} \binom{m+n}{i} (1 - y_{core})^{m+n-i} (y_{core})^i, \tag{4.2}$$

where m and n are the number of required MCores and spare MCores (RMCores), respectively, in the system. The yield of a single core (y_{core}) is:

$$y_{core} = y_{lanes} \times y_{L1} \times y_{skel}, \tag{4.3}$$

where y_{lanes}, y_{L1}, and y_{skel} are the yield of the lanes, L1 cache, and everything else, respectively.

When RCore lanes are utilized as spare lanes, the yield of the lanes $y_{lanes} = Binom(y_{lane}, k, l)$, where k is the number of required lanes in each MCore (e.g., 32) and l is the number of spare lanes for each MCore (the number of redundant lanes available to an MCore from an RCore).

If RCore lanes are utilized as shared spare lanes, instead of calculating the yield of an individual core (y_{core}, Eq. (4.3)), we divide the system into groups of three cores (two MCores and one RCore) and calculate y_{3core}. This is necessary since the yield of the two MCores and the RCore are interdependent (refer to Section 3.1.4). The yield of each of these groups is:

$$y_{3core} = Binom(y_{lane}, 2k, l) \times y_{L1}^2 \times y_{skel}^2. \tag{4.4}$$

Note that the yield of each RCore's $L1$ and $skel$ are accounted for in the system yield formula Eq. (4.1).

Expected Performance

$E[P]$ is the expected performance of a given configuration in the presence of defects:

$$E[P] = \sum_{C_i \in \{S\}} Perf(C_i) \times Prob(C_i), \tag{4.5}$$

where S is the *generating set* of the baseline (no defects) and derivative configurations (with defects) of a system. $Perf$ is the configuration's performance (the inverse of benchmark execution time), and is determined with detailed simulation that accounts for the presence of defective components.

$Prob$ is the probability of the configuration's occurrence in the population of dice given a particular defect density. Some configurations with many defects and consequently low performance may be unlikely to occur. Therefore, the performance of a given configuration must be weighted by how often it appears in a given population of dice. The number of resulting performance simulations may be large, however. Consider a system with six cores (MCores), each with 32 lanes, and three redundant cores (RCores), each with two shared lanes that can replace defective lanes in the MCores. For such a system, there are more than six million derivative configurations that meet the performance requirement of at least six functional MCores.

To reduce the cost of evaluation, most configurations can be grouped into a much smaller number of generating configurations. For instance, some defects may result in configurations that are identical from a performance perspective, such as when any single MCore in the system has a defective lane, or when any single core is defective. Such symmetric configurations can be safely ex-

cluded from S, and their probability of occurrence combined with that of the generating configuration. Doing so in the case of our example above reduces the number of required performance simulations to 33. Unfortunately, even in this case evaluating each configuration in the generating set requires more than 792 hours on a 3 GHz Core i7 platform with 8GB of RAM.

Estimating $E[P]$

To control the computational cost of design evaluation, we estimate $E[P]$ with $\hat{E}_m[P] \leq E[P]$ by limiting the configurations we consider possible. By evaluating only those configurations with high likelihood of occurring, we can achieve a reasonable estimate at a fraction of the computational cost.

First, we assume that $S = \{C_0, C_1, ..., C_n\}$ such that $Prob(C_{i+1}) \leq Prob(C_i)$. Then,

$$\hat{C}_i = \begin{cases} C_i & \sum_{j=1}^{i} Prob(C_j) \leq Th \\ Null & otherwise \end{cases} \tag{4.6}$$

where $0 < Th \leq 1$ is selected to control the computational cost of the set of simulations. Now, $\hat{S} = \{\hat{C}_0, \hat{C}_1, ..., \hat{C}_n\}$, and therefore

$$\hat{E}_m[P] = \sum_{\hat{C}_i \in \{\hat{S}\}} Perf(\hat{C}_i) \times Prob(\hat{C_i}), \tag{4.7}$$

when $Perf(Null) = 0$.

We observe that $\hat{E}_m[P] \leq E[P]$ and that as $Th \to 1$, $\hat{E}_m[P] \to E[P]$. This conservative calculation helps us to avoid costly performance calculations that contribute little to $E[P]$. Determining how to select Th is the subject of

Chapter 5. We have observed in practice that dice with more than one defective lane, for instance, are highly unlikely [11]; we therefore limit ourselves to the cases where all components are functional (no defects), where one lane in the system is defective (when considering hot- SL and SSL), and where one core in the system is defective (when considering hot-SC). These cases constitute more than 94% of the dice population for our example system whether considering hot- SL, SSL, or SC, reduce the required simulation by 90%. This significantly reduces the number of systems for which we must calculate $Prob$.

Probability of Occurrence

Calculating the probability of occurrence $Prob$ is similar to calculating yield, but rather than determine the fraction of systems that satisfy a particular set of requirements (number of cores, lanes, etc.), we calculate the fraction of systems that have exactly one given configuration. The probability that a given configuration occurs is

$$p_{C_i} = p_{cores} \times p_{crossbar} \times p_{L2\$} \tag{4.8}$$

where $p_{L2\$}$ and $p_{crossbar}$ are the probability of occurrence of the L2 cache and crossbar, respectively. We assume the yield of these components is 1 (see Section 4.2.2); $p_{L2\$}$ and $p_{crossbar}$ are therefore 1.

p_{cores} is the probability of observing a particular set of cores given the components that are present: the number of MCores n_{mc}, the number of RCores n_{rc}, the number of total MCore lanes n_{ml}, and, the number of total RCore lanes n_{rl}; and, the number of components that are defective: the number of

defective MCores n_{dmc}, the number of defective RCores n_{drc}, the number of defective lanes in an MCore n_{dml}, and, the number of defective lanes in an RCore n_{drl}.

When considering the application of spare lanes, for instance, the probability of observing a system with a one MCore with one defective lane is determined by

$$p_{cores} = \binom{n_{mc}}{n_{dmc}} \times p_{core'}^{n_{dmc}}(n_{ml}, n_{dml}) \times p_{core}^{(n_{mc}-n_{dmc})}(n_{ml}) \times \binom{n_{rc}}{n_{drc}} \times p_{core'}^{n_{drc}}(n_{rl}, n_{drl}) \times p_{core}^{(n_{rc}-n_{drc})}(n_{rl}), \quad (4.9)$$

where p_{core} is the probability of having all n_l lanes operational (n_{ml} and n_{rl} for MCores and RCores respectively),

$$p_{core}(n_l) = y_{lane}^{n_l} \times y_{skel} \times y_{L1\$}, \quad (4.10)$$

and where $p_{core'}$ is the probability of having $n_l - n_{dl}$ operational lanes (in the case study system and utilizing SL, $n_{dl} = 1$),

$$p_{core'}(n_l, n_{dl}) = \binom{n_l}{n_{dl}} \times y_{lane}^{(n_l - n_{dl})} \times (1 - y_{lane})^{n_{dl}} \times y_{skel}. \quad (4.11)$$

p_{cores} can be derived for the SSL and SC cases in a similar fashion.

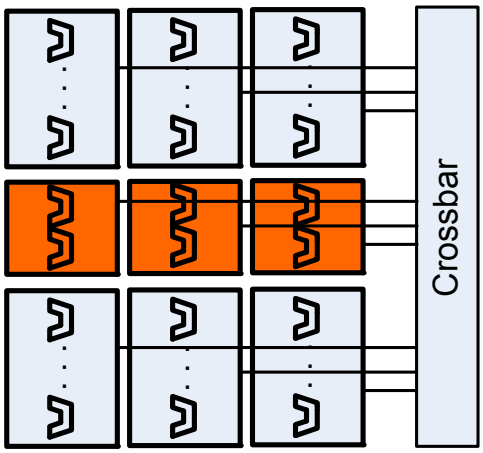


Fig. 4.2: Case study system.

4.2 Experimental Setup

To determine what form of hot spare redundancy is most beneficial for a particular (a) configuration and (b) application, and compare these results with that when cold spares are used, we performed a set of experiments on a case study system inspired to by the GeForce GTX-260 GPGPU from NVIDIA Co. We evaluated the performance and cost of a variety of multi-core SIMT configurations with hot and cold spare lanes (SL), shared spare lanes (SSL), and spare cores (SC), on a set of diverse benchmarks. The GTX-260 has six streaming processors (cores), each with 32 CUDA-cores (lanes) [62]; we integrate an additional three redundant RCores with two lanes each when considering SL and SSL redundancy, or an entire core (RMCore) when considering SC redundancy. RCores (red) are distributed among MCores in such a way that each MCore is located beside only one RCore, as illustrated in Figure 4.2). We have previously observed that many redundant lanes are not helpful in terms of cost reduction in the context of SIMT processors (refer to Section 3.3). Therefore,

we did not consider more than two lanes per each RCore.

4.2.1 SIMT Configurations and Benchmarks

We used MV5 [43] to simulate the performance of each SIMT configuration we considered. We assume a number of architectural parameters consistent with the literature [46], including: 0.6 GHz processor clock frequency, 16 KB L1 instruction cache and a unified 4 MB L2 cache per core, 300 MHz crossbar, and 300 cycle memory access latency. Each MCore has 64 KB L1 data cache, while each RCore has 8 KB L1 data cache. The number of hardware threads per core is set to twice the number of lanes per core (SIMT-depth = 2) for both kinds of cores (R- and M-Cores) in the system. We use the same list of benchmarks that is utilized in the simulation-setup of the previous section (refer to Section 3.2).

4.2.2 Cost Estimation

We use the cost model developed in Section 3.2.2. We extend this model to develop a cost model for systems utilizing hot-sparing. For wafers we assume the cost of \$3000, diameter $300mm$, and yield of 1. Also, we adopt the negative binomial yield model to calculate the yield of individual components (refer to Section 3.2.2). This model has three parameters: defect density (λ_b), the clustering parameter (α), and block area (A_b) [55, 56]:

$$y_b = \left(1 + \frac{\lambda_b \times A_b}{\alpha}\right)^{-\alpha}. \tag{4.12}$$

Table 4.1: SIMT Component Area.

Processor Configuration		Component	Area (mm^2)
Tech. size	65nm	ALU	0.0915
#(Int. Mult.)/ALU	1	Reg. File	0.9436
#(Int. ALU)/Lane	1	Lane	0.1319
#Lanes/Core	32	Skeleton	1.5801
L1 D$ Size	64KB	Core	7.1917
SIMT Depth	2	D$	1.3901
L2$ Size	4096KB	L2$	46.0652
#Cores	6	Processor	89.2154

In $65nm$ manufacturing technology, we assume $\lambda_b = 0.025/cm^2$ and $\alpha = 4$ [51]. Moreover, we consider the yield values of D$ and crossbar 1, since inexpensive redundancy can increase yield dramatically [32].

To estimate component area for yield estimation, we measured the area of functional units based on a gate-level synthesis of a SIMT processor, FlexGrip [61]. FlexGrip is a configurable GPUPU targeting FPGA, which is based on the NVIDIA G80 architecture. FlexGrip is configurable, and it is possible to define many architectural parameters such as the number of cores and lanes. We set FlexGrip to mimic the case study system (32 lanes per core), and modified it to be compatible with an ASIC flow by substituting HDL for the IPs it employs. We synthesized the processor for the 65nm TSMC process, considering both the timing and area overheads of wires in the design. We extracted the sizes and the number of ports for the memories and register files in the implementation and used CACTI [63] memory models to estimate their areas. The area measurement of different sub-components of the FlexGrip GPGPU is reported in Table 4.1.

4.2.3 Performance Degradation with Hot Redundancy

Adding redundancy often increases critical path delay and consequently ultimately reduces system performance [8, 11]. When we employ SC in a crossbar-based system, the only performance penalty is due to the increased size of the crossbar. MV5 does not support variation in crossbar delay; however, since crossbar delay grows slowly, we neglect this performance degradation in the system [64].

When using hot spare lanes, a main core that utilizes an SL must slow down to accommodate the additional delay introduced by steering logic. We measured this delay using FlexGrip and observed it to be less than 3.2%. Therefore, whenever an MCore uses an SL, its frequency is decreased by 3.2%; other cores operate at their highest clock rates.

We expect to observe similar performance degradation when SSL are utilized. However, the amount of this degeneration varies depending on the number of accessible SSLs (which affects the delay of the steering logic) as well as their physical distances to the MCore that they are shared with. Based on measurements from FlexGrip, this variation is between 4.5% to 6.8% of the operating frequency of the case study system, while the observed relative area overhead (the area portion of added steering logic against the area of a lane) is less than 0.1%.

These decreases in frequency, due to utilizing SL or SSL, 1) only affect the core that is defective and utilizes a spare lane, and does not degrade to the entire chip's working frequency if the processor can independently clock the cores, and 2) does not result in the same proportion of performance degradation

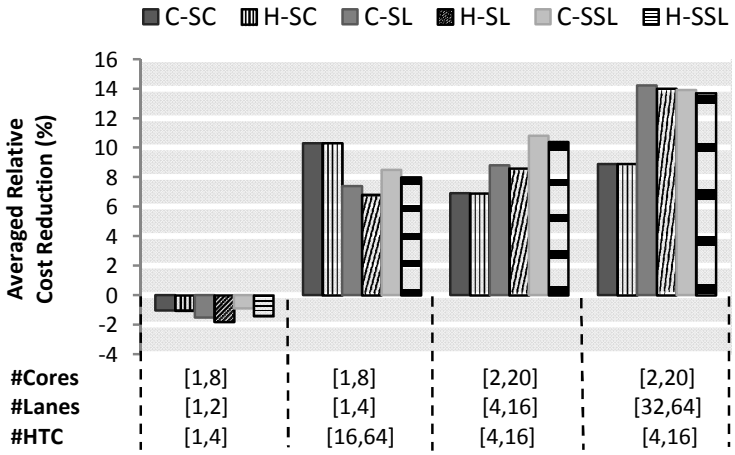


Fig. 4.3: Comparison of cold- and hot-sparing under different sets of configurations and redundancy techniques.

for the corresponding core. Memory access latency can, in some cases, hide the reduction in clock frequency.

4.3 Results

4.3.1 Cost-Effectiveness of Hot-Sparing

We conduct a variety of experiments to investigate the effectiveness of hot-sparing. We begin with a comparison of the cost-effectiveness of cold and hot-sparing across a wide variety of system configurations utilizing spare cores (SC), spare lanes (SL), and spare shared lanes (SSL). In each case, we allocate a single type of redundancy. We consider n SC, l SL, and zero to two SSL, where $n \in SC = \{0, 1, ..., m\}$, $l \in SL = \{0, 1, ..., k\}$, for a system with m cores and k lanes per core. SSLs are only allocated when there is more than one main core in the system. We then calculate the *average relative cost reduction* (ARCR) of the defect-tolerant system. ARCR is defined as the change in cost

relative to the baseline design with no redundancy. Cost is calculated as in Section 4.2.2.

In Figure 4.3, we divide configurations into four groups with similar parameters and cost reduction behavior: those that benefit from (a) no redundancy, (b) core sparing, (c) spare lane sharing, and (d) lane sparing. The systems in each group are specified by the tuple (Cores, Lanes, HTC), where each value is given as a range. The ranges associated with each group are indicated on the x-axis; the y-axis indicates the ARCR of the group of systems for each type of redundancy.

When systems utilize cold or hot spare cores, we observe that there is no difference in ARCR between hot and cold sparing. With both types of redundancy, main cores in the system are replicated and result in the same area overhead and yield improvement. When systems utilize Cold-SL or Cold-SSL, their ARCR are at most 1.5% better than Hot-SL and Hot-SSL respectively in all different groups of presented systems. This difference is observed because cold sparing does not require RCore front-end and back-end units when integrating redundant lanes to the system. This results in lower area overhead and consequently higher yield. Although cold sparing outperforms hot redundancy, this advantage is marginal; by trading 1.5% in cost, on average, designers can equip their systems with hot spares that often improve system performance.

4.3.2 E[P]/C Improvement of Hot-sparing

In Figure 4.4, we illustrate the normalized *expected performance per cost* ($E[P]/C$) for different types of redundancy (bar graphs), normalized to the

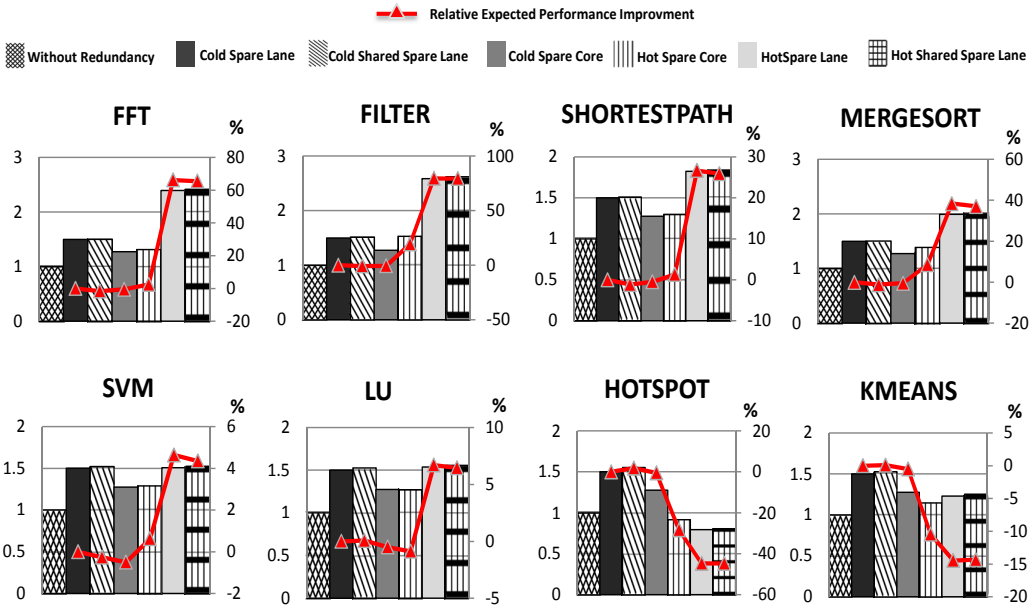


Fig. 4.4: Normalized $E[P]/Cost$ (bars, left) and relative $E[P]$ improvement (lines, right) and over different benchmarks.

baseline-system (without redundancy). The left y-axis (bars) indicates the $E[P]/C$ of the systems normalized to the baseline; the right y-axis (lines) indicates the percentage improvement in $E[P]$ alone relative to the baseline.

Four applications (FFT, Filter, ShortestPath, and MergeSort) show better $E[P]/C$ (nearly 1.6, 1.7, 1.2, and 1.3x, respectively) using hot spares than cold, over all types of redundancy. These application are able to make effective use of the narrow hot spares, resulting in performance improvement.

We do not observe such improvement for all applications. For example, SVM and LU show only marginally better $E[P]/C$ with hot-sparing than cold. For these applications, $E[P]$ improvements are small, at most 7% over than baseline system. These applications clearly make less effective use of the narrow spare cores, possibly as a result of presence of a greater number of

synchronization points in these applications [48, 49]: the wide cores (MCores) must wait for the completion of thread blocks mapped to narrow cores (hot RCores). While these applications experience poor $E[P]$ improvement, the reduction in cost results in nearly 1.5 times improvement in $E[P]/C$ compared to the baseline system; similar improvement is observed from cold sparing.

On the other hand, there are some applications (HotSpot and KMeans) that not only do not show any improvement in $E[P]/C$ in the presence of hot spares, but also experience significant performance degradation. Even though cost is reduced by adding redundancy to the system, $E[P]/C$ decreases: the cost reduction is not big enough to compensate for the loss of $E[P]$. We hypothesize that this performance loss is due to data-dependencies [50, 48]: performance is constrained by that of the narrow RCores when the wide MCores idle and wait to receive data from them.

Ultimately, the decrease in $E[P]/C$ for some applications in the presence of hot-sparing is not a cause for concern for designers: hot spare units can be disabled, even at runtime, rendering hot spares cold. In the case study system, in the worst case, this transformation has a marginal overhead in terms the of cost (less than 4%), while hot spare techniques improve cost of the baseline system more than 31%.

4.4 Conclusion

We investigated how the application of hot redundancy in multi-core SIMT systems can not only improve yield, but also increase performance. We introduced a new metric, *expected performance per cost* ($E[P]/C$), and showed that

there are some applications that benefit greatly from hot-sparing, and for those that do not, the overhead associated with leaving spares cold is not significant. To support this effort, we synthesized the FlexGrip GPGPU in an ASIC flow to determine the area of processor components for use in our yield models.

We observed that when systems consist of a few narrow cores, the area overhead of hot-sparing compared to cold is considerable and results in 1.5% lower average relative cost reduction (ARCR). However, this cost overhead becomes negligible (less than 1%, in terms of ARCR) when systems integrate many wide cores. On the other hand, we observed that the performance improvement gained by hot-sparing reaches nearly 20% for some applications in a case study system of six 32-lane cores and three two-lane redundant cores. Based on this performance improvement, we observe that when the case study system is equipped with RCores, some applications experience 1.7 to 2.5x improvement in $E[P]/C$ compared to the system without redundancy, while the addition of those RCores increases the processor's area less than 5%. By employing hot-sparing and tolerating a marginal increase in the size of a SIMT processor, designers can expect to see impressive performance per cost improvement.

Chapter 5

Efficient Performance Evaluation in SIMT Processors

In this chapter, we build upon the previous chapter and we introduce a new performance estimation technique for systems with hot-sparing ($\hat{E}_s[P]$). We show that $\hat{E}_s[P]$ can estimate *expected performance* with a cost not more expensive than traditional performance evaluation.

Hot spares complicate system evaluation: the presence of defects affects what resources are available [2]. Accurate performance evaluation thus requires the simulation of the entire population of resulting dice in order to determine the *expected performance*, $E[P]$, of the system. While simply expensive for single system evaluation, it is intractable for design space exploration. Prior work has quantified performance in the context of defects [16, 12, 15, 8]. They utilize performance estimation tools to avoid detailed multi-core performance simulation. However, it is not applicable for our work since we require de-

tailed multi-core performance simulation to distinguish designs [2]. Therefore, we extend the estimation method that is introduced in Chapter 4, $\hat{E}_m[P]$: we calculate a proper Th value for $\hat{E}_m[P]$ to make it accurate enough to distinguish SIMT systems in terms of performance, keeping it computationally efficient. We show that by selecting the proper value for Th ($\geq 95\%$), $\hat{E}_m[P]$ evaluates at most the three most likely configurations, and assumes the performance of all others is zero, reducing simulation by 93%, while $\hat{E}_m[P]$ remains accurate enough to distinguish designs. However, this remains computationally expensive for design space exploration when individual, detailed, simulations require hours. Therefore, we introduce another estimation techniques, $\hat{E}_s[P]$. $\hat{E}_s[P]$ evaluates only the most likely configuration, and assumes its performance for all other configurations, reducing simulation by 98%, with no more than 2.6% error in $E[P]$, sufficient for differentiating designs along the Pareto-optimal front during design space exploration. Consequently, designers may add redundancy, and evaluate system performance and cost, with no greater design effort than performance evaluation alone.

Expected Performance (Recall from Section 4.1.2)

As mentioned before, $E[P]$ is the expected performance of a given configuration in the presence of defects:

$$E[P] = \sum_{C_i \in \{S\}} Perf(C_i) \times Prob(C_i), \tag{5.1}$$

where S is the *generating set* of the baseline (no defects) and derivative config-

urations (with defects) of a system. $Perf$ is the configuration's performance (the inverse of benchmark execution time), and is determined with detailed simulation that accounts for the presence of defective components. $Prob$ is the probability of the configuration's occurrence in the population of dice given a particular defect density.

5.1 Probability of Occurrence

In Section 4.1.2, we introduced a model that calculates the probability of occurrence for a case study system. We expand upon that model, and we introduce a general model that covers the probability of occurrence for SIMT systems with different configurations. As mentioned in Section 4.1.2, calculating the probability of occurrence, $Prob$, is similar to calculating yield, but rather than determining the fraction of systems that satisfy a particular set of requirements (number of cores, lanes, etc.), we calculate the fraction of systems that have exactly one given configuration. The probability that a given configuration occurs is

$$p_{C_i} = p_{cores} \times p_{crossbar} \times p_{L2\$} \tag{5.2}$$

where $p_{L2\$}$ and $p_{crossbar}$ are the probabilities of occurrence of the L2 cache and crossbar, respectively. We assume the yield of these components is 1 (refer to Section 3.2.2); $p_{L2\$}$ and $p_{crossbar}$ are therefore 1 as well.

p_{cores} is the probability of observing a particular set of cores given the components that are present: the number of MCores n_{mc}, the number of RCores

n_{rc}, the number of MCore lanes n_{ml}, and, the number of RCore lanes n_{rl}; and the number of components that are defective: the number of MCores with defective MLanes n_{dmc}, the number of RCores with defective RLanes n_{drc}, the number of defective lanes in each MCore $n_{dl_{mi}}$, the number of defective lanes in each RCore $n_{dl_{ri}}$, and the number of *three cores* with defective lanes n_{d3c}. Given n_{dl} defective lanes and assuming the application of spare lanes (SL),

$$p_{cores} = \sum_{\{n_{dmc},n_{drc}\}\in E} \sum_{\{n_{dl_{m1}},n_{dl_{m2}},\ldots,n_{dl_{mi}}\}\in D} \sum_{\{n_{dl_{r1}},n_{dl_{r2}},\ldots,n_{dl_{rj}}\}\in Q} \Bigg[\binom{n_{mc}}{n_{dmc}} \times \Big(\prod_{k=1}^{i} p_{core'}(n_{ml}, n_{dl_{mk}})\Big) \times p_{core}^{n_{mc}-n_{dmc}}(n_{ml}) \times \binom{n_{rc}}{n_{drc}} \times \Big(\prod_{k=1}^{j} \times p_{core'}(n_{rl}, n_{dl_{rj}})\Big) \times p_{core}^{n_{rc}-n_{drc}}(n_{rl}) \Bigg], \quad (5.3)$$

where $D = \{0, 1, 2, ..., n_{dl}\}^{n_{dmc}}$, $Q = \{0, 1, 2, ..., n_{dl}\}^{n_{drc}}$, $E = \{0, 1, 2, ..., n_{dl}\}^2$, $n_{dml_1}, n_{dml_2}, \ldots, n_{dml_i} \leq n_{rl}$, $n_{drl_1}, n_{drl_2}, \ldots, n_{drl_j} \leq n_{rl}$, and $(n_{dml_1} + \ldots + n_{dml_i} + n_{drl_1} + \ldots + n_{drl_j}) \leq n_{dl}$.

The first summation takes care of where defective lanes might cause defective cores (MCores or RCores), while the second and the third cover the different cases where defective lanes are spread among cores. The rest of the formula counts the number of possible cases that the number of defective MCores, defective MLnaes per MCore, defective RCores, and defective RLanes per RCore out of the total number of MCores, MLanes per MCore, RCores,

and RLanes per RCore, respectively.

For SC, p_{cores} is calculated by

$$p_{cores} = \sum_{i=0}^{n_{dl}} \sum_{\{n_{dl_{m1}}, n_{dl_{m2}}, ..., n_{dl_{mi}}\} \in D} \left[\binom{n_{mc} + n_{rmc}}{i} \left(\prod_{k=1}^{i} p_{core'}(n_{ml}, n_{dl_{mk}}) \right) \times p_{core}^{n_{mc}+n_{rmc}-i}(n_{ml}) \right], \quad (5.4)$$

where $D = \{0, 1, 2, ..., n_{dl}\}^i$, $n_{dl_{m1}} + \ldots + n_{dl_{mi}} \leq n_{dl}$, and n_{rmc} is the number of RMCores in the system. Like in Eq.(5.3), the first summation takes care of the cases where defective lanes might cause cores with defects (MCores or RMCores), while the second summation covers all possible cases where defective lanes are spread among cores with defects. Note that a core with defect does not fail since its micro-architectural redudnancy can address its defect. The rest of the formula counts the number of ways that the number of defective MCores and RMCores and defective MLanes per MCore out of the total number of MCores plus RMCores and MLanes per core, respectively.

For SSL, we first group cores (MCores and RCores) into groups of three (two MCores and one RCore), and we call these groups *three cores* (3c). Then,

p_{cores} for SSL is calculated by

$$p_{cores} = \sum_{i=0}^{n_{d3c}} \sum_{\{n_{dl_{3c1}}, n_{dl_{3c2}}, \ldots, n_{dl_{3ci}}\} \in D} \Bigg[\binom{n_{3c}}{i} \times \Big(\prod_{k=1}^{i} p_{core'}(2 \times n_{ml} + n_{rl}, n_{dl_{3ck}}) \Big) \times p_{core}^{n_{3c} - n_{d3c}}(2 \times n_{ml} + n_{rl}) \Bigg], \quad (5.5)$$

where n_{d3c} is the number of defective *three cores* with defective lane-failure, $n_{dl_{3ck}}$ is the number of defective lanes in the kth *three cores*, $D = \{0, 1, 2, ..., n_{dl}\}^i$, $n_{dl_{3c1}}, n_{dl_{3c2}}, \ldots, n_{dl_{3ci}} \leq n_{rl}$, and $n_{dl_{3c1}} + \ldots + n_{dl_{3ci}} \leq n_{dl}$. The first summation takes care of the cases where defective lanes might cause defective *three cores*. The rest of the formula counts the number of possible cases that the number of *three cores* with defects and defective lanes per *three cores* out of the total number of *three cores* and lanes per *three cores*, respectively.

p_{core} is the probability of having all n_l lanes operational (n_{ml} and n_{rl} for MCores and RCores, respectively),

$$p_{core}(n_l) = y_{lane}^{n_l} \times y_{skel} \times y_{L1\$}, \quad (5.6)$$

$p_{core'}$ is the probability of having $n_l - n_{dl}$ operational lanes,

$$p_{core'}(n_l, n_{dl}) = \binom{n_l}{n_{dl}} \times y_{lane}^{(n_l - n_{dl})} \times (1 - y_{lane})^{n_{dl}} \times y_{skel}. \quad (5.7)$$

Note that Eqs. (5.3)-(5.5) can calculate, for different redundancy regimes, the probability of occurrence for individual derivative designs or a group of

symmetric derivative designs as well. To calculate the probability of occurrence for an individual design, we have to set every parameter in the formulas. For instance, by determining $\{\{n_{dmc}, n_{drc}\}, \{n_{dl_{m1}}, n_{dl_{m2}}, ...\}, \{n_{dl_{r1}}, n_{dl_{r2}}, ...\}\}$ in the p_{cores} formula for SL, one can find the probability of occurrence for a specific derivative design. However, when some of the specification variables are not determined, the formulas calculate the probability of occurrence for the group of corresponding symmetric derivative designs. For example, in the p_{cores} formula for SL, when the user does not determine the number of defective MCores (n_{dmc}) with the defective lanes, the formula will sum up the probabilities of occurrence for all possible cases in which the defective lanes can result in failure in MCores.

5.2 Estimating $E[P]$

In Chapter 4, we already have introduced a simplified estimation method, $\hat{E}_m[P]$, by limiting the number of configurations that are evaluated. We showed that $\hat{E}_m[P] \leq E[P]$. By simulating only those configurations with high likelihood of occurring, we can achieve a reasonable estimate of expected performance at a fraction of the computational cost of exact evaluation. However, we picked a value for one of the parameters in calculating $\hat{E}_m[P]$, Th, which is translated to the number of configurations with high likelihood of occurrence that should be simulated.

First, we justify a proper value for Th in $\hat{E}_m[P]$. $\hat{E}_m[P]$, determines the m most likely derivative designs, performs simulation to determine their performance, and assumes the performance of all others is 0. Then, we propose and

evaluate another estimation technique, $\hat{E}_s[P]$, which simulates the single most likely derivative design, and assumes its performance for all configurations.

5.2.1 Selecting Proper Th for $\hat{E}_m[P]$

We introduced $\hat{E}_m[P]$ in Section 4.1.2. $\hat{E}_m[P]$ reduces the computational complexity of evaluating $E[P]$ by evaluating only the m most likely designs. Derivative designs that are unlikely to occur contribute little to $E[P]$; the accuracy of $\hat{E}_m[P]$ can therefore be controlled by tuning the fraction of the population of dice covered, Th. The challenging in selecting Th is to minimize computational effort while maintaining sufficient accuracy to distinguish between designs. We observe that in general, few simulations are needed to cover nearly all of the population of dice, with $Th \geq 95\%$.

From Section 4.2.2, recall the example of a system of six MCores, each with 32 lanes, to which we allocate three RCores, each with two lanes. As previously noted, configurations with symmetric defects may be grouped; there are 33 different groups for this example. If we set $Th \geq 95\%$, only the first three most likely derivative designs are evaluated, and computation is reduced by 91%, to 72 hours from 792 hours.

In this case, these derivative designs correspond to the fully operational system, and those with one or two MLane failures, which contribute 88%, 6%, and 2% of the yield of the system, respectively, or of 96% of the population of dice; coincidentally, this results in a 4% underestimation of $E[P]$ for this design.

5.2.2 Single-Simulation Estimation of $E[P]$ with $\hat{E}_s[P]$

Unfortunately, the computational effort required to calculate $\hat{E}_m[P]$ (*i.e.*, performing three performance evaluations per design point), is still high considering design space exploration; consequently, we propose to reduce the cost of estimating $E[P]$ to its limit, a single performance evaluation. This is justified by the following observations. For the system above (6MCores + 3RCores): (1) the performance of operational, unevaluated, derivative designs is much closer to that of the fully functional design than zero; (2) in the worst case across all benchmarks, the performance of the second and third most likely configurations (one and two failed MLanes in the system, respectively) differ from the baseline 15% and 19%, respectively; and, (3) the probabilities of occurrence of the second and third most likely configurations are 6% and 2%, respectively.

We consequently hypothesize that if we (a) evaluate only the single most likely design, and (b) assume its performance for all operational derivative designs, that we further reduce the computational complexity of evaluation with minimal error. In the case of our example, $\hat{E}_s[P]$ estimates $E[P]$ with 1.43% error—less error than $\hat{E}_m[P]$, while reducing the cost of simulation to 24 hours. In general, if this trend holds across the design space of multicore SIMT processors, then $\hat{E}_s[P]$ would make it possible to evaluate the cost and performance implications of hot redundancy with *no overhead relative to conventional performance evaluation*.

5.3 Experimental Setup

To determine Th for $\hat{E}_m[P]$, and to evaluate the accuracy of $\hat{E}_s[P]$, we conduct several of experiments across a wide variety of multi-core SIMT configurations. We measure the performance of derivative designs by using MV5 [43]; the probabilities of occurrence for the designs are calculated based on the formulas in Section 5.1. The cost of each design is determined using the cost formulas in Section 4.2.2. To observe the performance behavior of the designs over different applications, we selected the same set of benchmarks as Section 3.2.

5.3.1 Small and Large Design Sets

Given that our approach makes use of the distribution of probabilities of occurrence and performance under defects, we performed experiments with two different sets of configurations in order to evaluate how effectively the estimation techniques distinguish design points in different circumstances: (a) configurations in the *Small* set of systems that employ few narrow cores, and (b) conversely, *Large* systems have many wide cores.

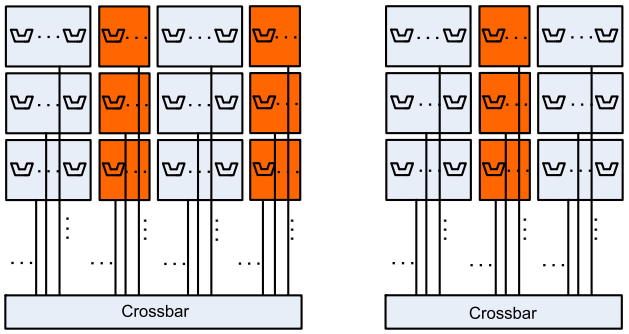
The *Small* set tests our hypothesis that when the baseline has high yield, its performance is the most significant contributor to $E[P]$; related derivative designs occur too infrequently to affect $E[P]$. The *Large* set tests our hypothesis that the $E[P]$ for large systems are dominated by a few derivative designs with approximately the same performance: a few failures have relatively little affect. The architectural parameters for *Small* designs are presented in Table 5.1. Note that we assume the size of $L2\$$ is constant and equal to 4MB.

Small systems have die areas in $[50, 100]$ mm^2 (e.g., NVIDIA Tegra series [65]); this set consists of 203 systems. The architectural parameters for *Large* designs are also presented in Table 5.1. *Large* systems have die areas in $[200, 250]$ mm^2; this set consists of 143 systems. Note that while real GPGPUs have die areas greater than 500 mm^2 (e.g., NVIDIA GTX 480), about two-third of their die areas (200 to 350) mm^2 are dedicated to streaming processors and L2$ [66], while the rest is dedicated to the memory controller, host interface, etc.

As indicated in Table 5.1, we added the same type of RCores for both *Small* and *Large* systems. RCores are distributed among MCores in such a way that each MCore is located next to only one or two RCores (Figure 5.1): the number of RCores that are added is either the same as the number of MCores in the system (lane sparing), or half that (shared lane sparing).

5.3.2 Primary Design Set

After evaluating $\hat{E}_m[P]$ and $\hat{E}_s[P]$ techniques over the *Small* and *Large* sets, we select the most efficient estimation technique ($\hat{E}_s[P]$) and define more experiments to evaluate its accuracy. Since performing performance evaluation over all practical designs is intractable, we select a very wide design set, but we perform an analytical evaluation of $\hat{E}_s[P]$ instead. We call this set of designs *Primary*; this set consists of 9582 systems. *Primary* covers the designs between the *Large* and *Small*, i.e., the medium sized designs with die areas in $[100, 200]$ mm^2. Under this constraint, we varied the number of MCores N, $N \in \{1, 2, 4, 6, \ldots, 20\}$, the number of lanes per MCore (SIMT width) W and hardware threads per MCore (SIMT depth) D, $W\&D \in \{1, 2, 4, 8, \ldots, 64\}$,



(a) Core placement when #RCores = #MCores, for utilizing SL.

(b) Core placement when #RCores = #MCores/2 for utilizing SSL.

Fig. 5.1: Cores placement in the presence of redundancy.

Table 5.1: Parameters for *Small* and *Large* system sets.

MCores Spec.	Small	Large	Redundant Cores Spec.	
MCores	{1, 2, 4, 8}	{16, 20}	RCores	{# MC, # MC/2}
MC D$ Size	{8, 16, 32, 64}	{32, 64}	RC D$ Size	8KB
MC HTCs	{1, 2, 4, 8}	{32, 64}	RC HTCs	{2, 4}
MC Lanes	{1, 2, 4}	{16, 32}	MC Lanes	{1, 2}
MC I$ Size	16KB	16KB	RMCores	{0, 1}
MC Freq.	1 GHz	1 GHz	RC Freq.	1 GHz

and L1 data cache size S, $S \in \{8, 16, 32, 64\}$ KB, while the number of RCores M, $M \in \{0, \#MCores, \#MCores/2\}$, the number of lanes per RCore V and hardware threads per RCore E, $V \in \{1, 2\}$ and $E \in \{2, 4\}$, L1 data cache size $= 8KB$, also there is zero or one RMCore for each system, and for all systems $L2\$ \in \{1, 2, 4\}$ M, $I\$ = 8$ KB. Note that we do not consider combinations of redundancy techniques; optimal solutions tend to utilize a single type of redundancy [11].

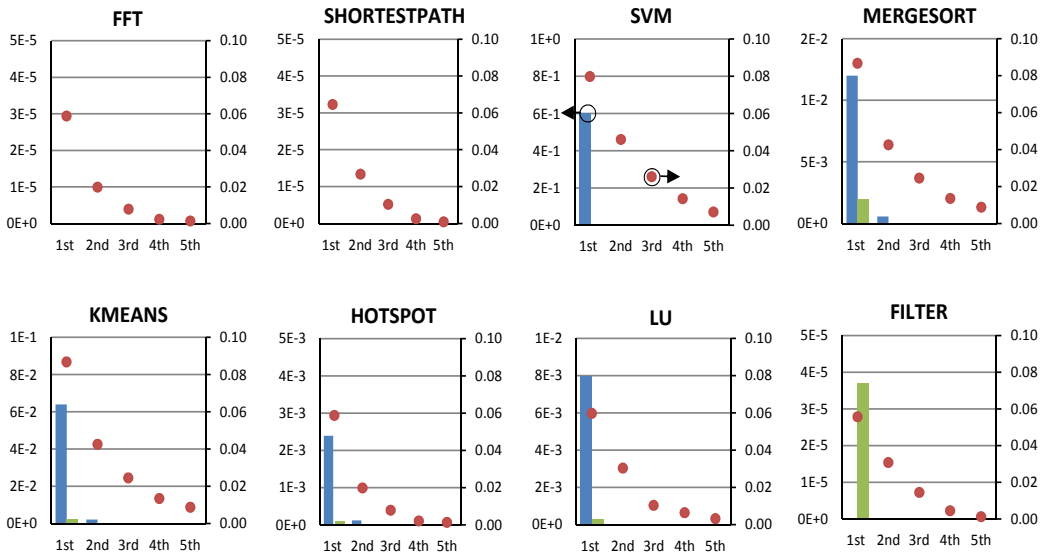


Fig. 5.2: Distance from true POF points, in terms of ADRS, for different levels of accuracy for $\hat{E}_m[P]$ (blue bars, left), ADRS numbers for the second method of calculating $\hat{E}_s[P]$ (green bars, left), average-relative-error of $\hat{E}_m[P]$ (red dots, right), and over different benchmarks.

5.4 Results

5.4.1 Validating $\hat{E}_m[P]$ Over the Small Set

When we evaluate $\hat{E}_m[P]$, we observe that, as expected, estimation accuracy increases as Th increases. Consider the situation when we evaluate the performance of only the most likely design, C_0; we call this the *first level of accuracy* (m=1). In this case, $\hat{E}_m[P] = Perf(\hat{C}_i) \times Prob(\hat{C}_i)$. We observe that when $Th \simeq 90\%$ since the probability of occurrence for the most likely derivative design is 90%. In this case, $\hat{E}_m[P]$ underestimates $E[P]$ by 6.7% on average for designs in the *Small* set.

We use *average distance from reference set* (ADRS) to compare the performance-cost Pareto-optimal fronts of $E[P]$ and $\hat{E}_m[P]$. Given a reference set R (deter-

mined by $E[P]$) and a solution set S (determined by $\hat{E}_m[P]$), ADRS measures the average normalized best-case distance along the worst-case axis (performance or cost) from each design in R to the nearest design in S [58]. In this way, it quantifies the difference in the considered objectives between the points in the solution set and those in the true Pareto-optimal front; lower values are better. Across all the applications we considered, we observe that $\hat{E}_m[P]$, at worst, has an ADRS of 0.6 (on a scale from 0 to 1), indicating that $\hat{E}_m[P]$ is not able to effectively identify designs on or near the Pareto-optimal front by utilizing the first level of accuracy (m=1).

When we use the second ($Th \simeq 95\%$), and the third ($Th \simeq 97\%$) *levels of accuracy*, we observe on average an ADRS of 0.002 and zero, respectively. ADRS values as a function of level of accuracy is illustrated for eight different benchmarks in Figure 5.2. The x-axis corresponds to the different levels of accuracy, while the y-axis on the left represents the magnitude of ADRS distance, and on the right corresponds to the magnitude of relative error. Blue-bar graphs correspond to the distance from true POF points, in terms of ADRS, for the different levels of accuracy for $\hat{E}_m[P]$, while red dots represent the average-relative-error, over the *Small* set for $\hat{E}_m[P]$. The average-relative-error is calculated by taking an average over the relative error of $\hat{E}_m[P]$ compared with the exact $E[P]$, derived by performing up to 51 performance evaluations for each system. As expected, both ADRS and the average-relative-error of $\hat{E}_m[P]$ decrease as we increase the levels of accuracy. However, improvement is greatest for the first few derivative designs, since these designs contribute the most to systems' yield.

As depicted in the graphs of Figure 5.2, $\hat{E}_m[P]$ can correctly detect the true POF sets for FFT, ShortestPath and Filter with just the first level of accuracy. For the rest of the applications, however, $\hat{E}_m[P]$ cannot identify the true POF set and has a considerable distance to it (e.g., ADRS of 0.6 for SVM). These experiments indicate that if we want to be able to distinguish design points, we must use at least the third level of accuracy. In this case, more than three days on a Intel Core i7 platform with 8 GB RAM is required to calculate the estimated expected performance of a single system.

5.4.2 Validating $\hat{E}_s[P]$ Over the Small Set

$\hat{E}_s[P]$ is based on evaluating the performance of the most likely derivative design of a system; therefore, it is 3× faster than $\hat{E}_m[P]$ when m=3. However, the accuracy of $\hat{E}_s[P]$ is directly related to how good an approximation the performance of the most likely derivative design is for other derivative designs. For small systems, since cores are not very wide (they utilize, at most, four lanes), we observed that the performance of derivative designs varies considerably. However, the low probabilities of occurrence for designs other than the most likely one lessen the impact of other derivative designs. We validate this intuition by calculating $\hat{E}_s[P]$ over the *Small* set and determining its POF set. Then, we measure its distance to the true POF set to see how well $\hat{E}_s[P]$ identifies Pareto-optimal solutions. We also calculate the average-relative-error for systems in the *Small* set.

In Figure 5.2, green bars represent the distance between the true POF and that found POF by utilizing $\hat{E}_s[P]$ in terms of ADRS. The true and

approximated POF are very close to each other, and in the worst case (Filter), $\hat{E}_s[P]$ is unable to identify just 16% of the true POF set. Moreover, the average-relative-errors for $\hat{E}_s[P]$ are 1.49%, 1.89%, 1.67%, 2.56%, 1.35%, 1.67%, 1.96%, and 2.23%, for FFT, Filter, ShortestPath, MergeSort, SVM, LU, HotSpot, and KMeans, respectively. In most of the cases, this error is less than that for $\hat{E}_m[P]$, resulting in better ADRS, despite the fact that fewer performance simulations are performed.

As depicted in these graphs, when running one performance simulation per system and utilizing the first level of accuracy (first blue bar in each graph), $\hat{E}_s[P]$ outperforms $\hat{E}_m[P]$ by one order of magnitude, in terms of ADRS, over all the benchmarks except Filter (where $\hat{E}_m[P]$ outperforms $\hat{E}_s[P]$), FFT, and ShortestPath (where the two techniques are equivalent). Moreover, the average-relative-error of $\hat{E}_s[P]$ is less than or equal to $\hat{E}_m[P]$ at the first level of accuracy. For *Small* systems, even though the performance differences between fully functional and other operational configurations are high, the low probabilities of occurrence for configurations other than fully functional one reduce their impact on $E[P]$.

The behavior of $\hat{E}_m[P]$ and $\hat{E}_s[P]$ on Filter can be explained by observing the characteristics of the application. Over the *Small* systems, the performance of Filter application is affected considerably by each defective lane (i.e, losing one RCore per lane failure in SL technique). Hence, estimating the performance of derivatives with the most likely one (fully operational system) introduces considerable error in $\hat{E}_s[P]$. However, this error does not result in a great distance between the $\hat{E}_s[P]$ identified POF set and the true POF over

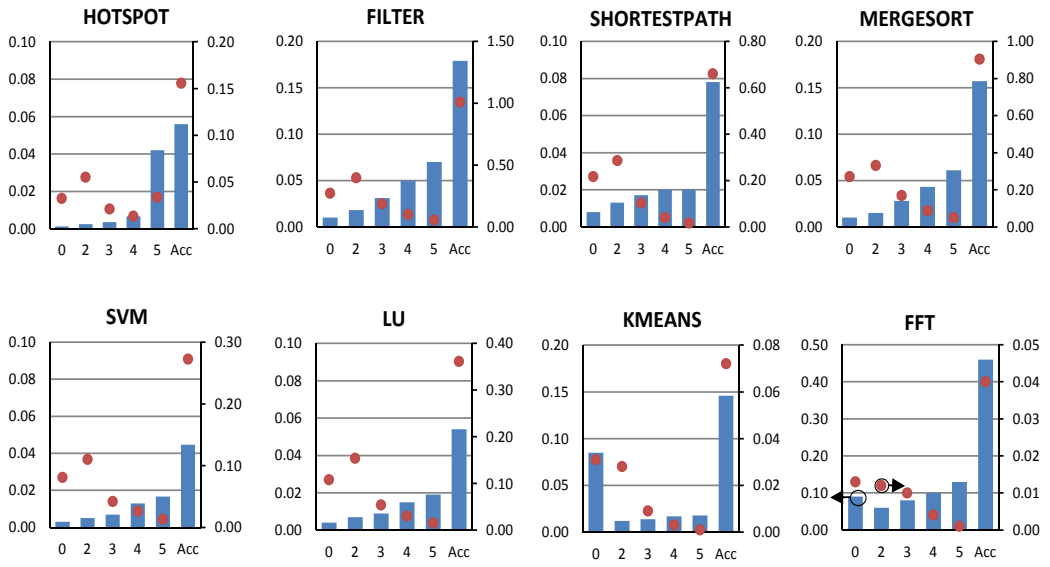


Fig. 5.3: Average relative performance differences to one failed main-lane configuration, for different derivative configurations (blue bars, left, in %), the errors of $\hat{E}_s[P]$ due to considering the performance values of all operational derivative configurations equal to one failed main-lane (red dots, right, in %), and over different benchmarks.

the *Small* systems (less than 5e-5 in terms of ADRS).

5.4.3 Validating $\hat{E}_s[P]$ over the Large Set

To see how effectively $\hat{E}_s[P]$ distinguishes systems in the *Large* set, we compared it with an accurate approximation of $E[P]$ using $\hat{E}_m[P]$. $\hat{E}_s[P]$ approximates the performance of all derived configurations from each system with one failure in main-core lanes (MLane): configurations with one MLane failure have the highest probability of occurrence in this set of designs. For *Large* systems (Table 5.1), each core is equipped with at least 16 lanes and there are more than eight cores per system. In these kinds of systems, it is very likely that at least one lane fails, since these systems utilize many lanes: for the

Large set, the average probabilities of occurrence for the first six most likely derivative configurations are 27%, 33%, 22%, 6%, 2%, and 0.8% for the fully functional, and one to five main-core lane failures, respectively. Moreover, the probabilities of occurrence are more evenly distributed over derivative designs (in contrast with the *Small* set, where the fully function system has, by far, the highest probability of occurrence).

As a baseline for comparison, we calculated $\hat{E}_m[P]$ for *Large* systems, covering more than 99.3% of each system's yield by simulating up to 51 different groups of configurations per system (m=51). Note that for *Large* systems many groups of derivative designs have to be simulated to cover the remaining 0.7% (more than 95 other groups). Simulating these groups requires significant computational effort (more than 500K hours), and is therefore intractable even for this limited number of systems.

For each system, we measured the relative performance difference of each system with one main-core lane failure configuration and other likely derivative configurations (zero, and two to five MLane failures). Then we took an average of those relative performance differences over all the systems for each derivative configuration (average-relative-performance-error, left axis, percentage); this is illustrated with the blue bars in Figure 5.3. The total accumulated error is illustrated with the last bar.

Then, we calculated the error due to taking the performance of one MLane failure as the performance for all derivative configurations for each system. We averaged this over all the systems (average-$\hat{E}_s[P]$-absolute-error, right axis, percentage); this is illustrated with red dots in Figure 5.3. The total accumu-

lated error is illustrated with the last dot.

As depicted in Figure 5.3, the accumulated error for the first five configurations—which cover, on average, more than 90% of the yield of the systems—is not more than 1.2% over all benchmarks. To further validate $\hat{E}_s[P]$, we again identified the true POF (determined by $\hat{E}_m[P]$ with m=51) and the compared it with the POF identified by $\hat{E}_s[P]$. We observe that the estimation error of $\hat{E}_s[P]$ does not result in any differences between the true and approximate POF. In other words, due to the low error that this technique has for *Large* systems, $\hat{E}_s[P]$ is accurate enough to identify the true POF set without performing more than a single performance evaluation.

Also, in Figure 5.3, we observe that the average-$\hat{E}_s[P]$-absolute-error of fully functional systems is higher than that for two failed MLane for KMeans and FFT applications. This originates with the considerable performance difference between fully operational systems and those with one MLane failure for *Large* systems.

5.4.4 Validating $\hat{E}_s[P]$ via Analytical Evaluation

We observed that, for the *Small* and *Large* systems, $\hat{E}_s[P]$ can distinguish the design points correctly by performing just one performance evaluation for the most likely derivative design. However, many interesting designs lie outside of these subsets; to validate the effectiveness of $\hat{E}_s[P]$, we must consider estimation error over a wider variety of designs. Unfortunately, it is intractable to evaluate all such designs, even if we utilize $\hat{E}_s[P]$. Hence, we performed an analytical evaluation to see how accurate $\hat{E}_s[P]$ is over a wide set of designs,

Primary.

We grouped the designs in *Primary* according to their areas ([100, 110), [110, 120), ..., and [190, 200) mm^2), and performed analysis to determine the tolerable error in performance estimation over each group of these systems. To conservatively estimate the accuracy of $\hat{E}_s[P]$, we assume that each system's performance degrades, relatively, 50% for each lane failure after the the first lane failure (i.e., 1st to 2nd, 2nd to 3rd, and so on), exaggerating the impact of performance difference in less likely derivative designs. In other words, if we assume that the performance of a system with one lane failure is an arbitrary number 100, it will decrease to 6.25 after five lane failures.

We then determined the amount of performance difference required under a single lane failure (on which the performance difference of further lane failures depends) to result in the 2.6% estimation error previously observed for $\hat{E}_s[P]$. If this required performance degeneration is less than that observed for designs in the *Small* and *Large* design sets, then we assume that $\hat{E}_s[P]$ is accurate enough to distinguish designs in *Primary*.

We observe that, over the systems grouped in [100, 110), if the relative performance difference is less than 23% for the first lane failure, $\hat{E}_s[P]$ does not result in more than 2.6% error. On the other hand, we observed that the relative performance differences between the fully functional configuration and those with one lane failure are not more than 15% over the *Small* set (systems smaller than 100 mm^2); the required 23% difference is even more unlikely, since we observed that as systems gets bigger, the performance differences due to defective lanes decreases. In other words, we expect a corresponding

performance difference less than 15% over the set of designs that are grouped into [100, 110), short of the 23% needed to exceed 2.6% estimation error.

Moreover, we observe that the error monotonically decreases over the remaining groups of designs. For the largest group of systems, [190, 200), the relative performance differences between the fully functional and one lane failure must be greater than 3.1% to have error greater than 2.6% in terms of $\hat{E}_s[P]$. We observed, however, that the error for the [200, 250) group is less than 0.5% (the calculated average performance difference between fully operational and one lane failure in the *Large* set). It is clearly not probable that the average estimation error exceed 2.6% over the *Primary* set.

5.4.5 The Limitations of Estimating $E[P]$

In Section 5.4.4, we observed that under typical defect densities ($0.025/cm^2$) derivative designs are unlikely to experience sufficient performance differences, relative to the most likely derivative design, for the estimation error of $\hat{E}_s[P]$ to be greater than 2.6%. However, as manufacturing processes scale, higher critical area results in higher effective defect densities [6]; this is expected to make $\hat{E}_s[P]$ less accurate. Since the probabilities of occurrence, especially for small systems, will be more evenly distributed over derivative designs, error increases, and $\hat{E}_s[P]$ will tend to mistakenly select designs that are farther from the true POF. We observe that when the defect density is 2.7× higher, for *Small* systems (area less than $100mm^2$) the performance difference for systems with one lane failure is larger than 15%, on average; this results in more than 2.6% error for $\hat{E}_s[P]$. Consequently, $\hat{E}_s[P]$ might fail to find optimal

solutions due to not being sufficiently accurate. To address this problem, we suggest adaptively selecting an estimation technique based on system area, e.g., using $\hat{E}_m[P]$ for smaller designs and $\hat{E}_s[P]$ for larger ones.

5.5 Conclusion

In this chapter, we investigated how to estimate the performance of the multi-core SIMT systems with hot redundancy. The performance of a system in the presence of defects depends on the defect density and resulting population of operational dice: systems with fewer defects have more hot redundant resources available to improve performance. We therefore utilized *expected performance* ($E[P]$), which depends on the probability of occurrence of different derivative designs with defects and their resulting performance. Unfortunately, evaluating $E[P]$ exactly is computationally expensive, requiring more than 792 hours per system, and making design space exploration intractable.

Consequently, we have developed techniques for estimating $E[P]$. First, we developed formulas for determining the probability of occurrence for multi-core SIMT systems. Then, we proposed and evaluated a value for Th for $\hat{E}_m[P]$, and evaluated the accuracy of this estimation method to distinguish designs in terms of performance. Afterward, we introduced another estimation techniques, $\hat{E}_s[P]$, to estimate $E[P]$. We showed that by employing $\hat{E}_m[P]$ and $\hat{E}_s[P]$ techniques and performing just three and one performance simulation per system, respectively, we can estimate $E[P]$ with sufficient accuracy to distinguish the Pareto-optimal front in a multi-core SIMT design space.

For our set of *Small* systems, $\hat{E}_s[P]$ is much more accurate than $\hat{E}_m[P]$

(on average 5.3×), identifying designs within 0.0024 to the true performance-cost Pareto-optimal front, in terms of ADRS. Moreover, we observed that by utilizing $\hat{E}_s[P]$ when considering *Large* systems (with many more derivative designs), the average relative error decreases to less than 0.5%, and the the true POF is found for all considered applications. Further analysis demonstrated that these results generalize to the entire design space: when using $\hat{E}_s[P]$, error is expected to remain in an acceptable range ($\leq 2.6\%$) under realistic defect densities.

Hence, by employing $\hat{E}_s[P]$, which performs a single performance evaluation (about three hours for each application per system), designers are able to distinguish designs accurately enough to consistently identify optimal and near-optimal designs in terms of $E[P]$. Not only does $\hat{E}_s[P]$ help designers evaluate system performance given hot spares and defective components, but it does so at no higher computational cost than traditional performance evaluation.

Chapter 6

Hot-Sparing for Cost and Performance-per-Watt Improvement

In this chapter, we utilizing hot-sparing to improve cost and performance-per-watt in SIMT systems. Likewise performance we should calculate *expected performance-per-watt* for systems with hot-sparing. We show that calculating *expected performance-per-watt* accurately is computationally too expensive, so we estimate it. Also, we introduce an adopted design space exploration algorithm specific for our problem to search the design space of cost and *expected performance-per-watt*.

Great attention has been paid to energy-efficient computing in multi-core processors since, in new technologies, performance and power do not scale similarly [37]. For instance, Ghasemazar et al., [17] and Hanumaiah et al., [18]

showed that by using system-level techniques (DVFS, task allocation, and task migration) we can significantly increase PPW for homogeneous multi-core processors. They utilized the existing redundancy (timing slack) in a system to reduce its power without changing the system's performance (throughput). However, none of these work addresses cost, regardless they rely on the presence of timing slack to optimize energy consumption. To address these problems, we propose to utilize hot-sparing that even can reduce energy consumption of applications with short balanced phases since it does not rely on timing slack to improve PPW: the relative performance gain of hot-sparing, for some applications, can be better than its associate relative power overhead. Moreover, note that hot-sparing can be utilized with other energy saving techniques (such as DVFS) since they are applied at different abstraction levels.

In this chapter, we extend our prior investigation in Chapter 4 and we investigate the cost and performance-per-watt (PPW) implications of utilizing hot-sparing in SIMT processors over different applications and configurations. PPW captures both performance and power consumption in a system. In Chapter 4, we introduced and utilized hot-sparing for cost and performance improvement in SIMT processors, but we do not consider the power overhead implications of hot-sparing. The same way we did in Chapter 4, we allocate spare cores, lanes, and shared-lanes in SIMT processors, but we additionally enable these components for cost and PPW improvement when possible. Also, we evaluate cost and PPW design space and we categorize applications with respect to the type of redundancy that they utilize the best. This helps designers to observe how different types of redundancies match with different types

of applications.

Moreover, we introduce a guided design space exploration (DSE) algorithm, which uses an artificial neural network (ANN) regression model, adapted for our cost-PPW optimization problem. DSE algorithms can be totally based on prediction models. If so, they decrease the complexity of design space exploration by predicting the design space (e.g., response surface modeling), rather than investigating it exhaustively [40]. However, response surface model (RSM) based methods suffer from prediction error. The prediction error could easily lead them to detect approximated optimal solutions with a considerable distance to the real optimal solutions in a design space. On the other hand, some DSE algorithms, like ReSPIR [20], use a combination of design of experiments (DoEs) and response surface modeling (RSM) techniques to find better optimized design points by performing fewer simulations. Our DSE approach uses RSM to predict where the optimal solutions that are located in a design space. However, in our algorithm we do not just rely on the optimal solution set that is being detected by RSM, instead we investigate the design points near to the predicted optimal solution set to find better approximated optimal solutions, while we use a different approach to select RSM model as well. Our design space exploration (DSE) algorithm reduces the design exploration time to one-sixth, while finding Pareto-Optimal fronts (POF) three times closer to the real POF than conventional methods such as ReSPIR [20]. Meanwhile, our algorithm evaluates less than one-fifth of the design space. By employing the framework presented in this chapter, system architects can explore a wide range of cost-PPW design alternatives in the early stages of the design

process, and characterize the configurations that will generate the maximum performance-per-watt per cost over different applications.

6.1 Expected Performance per Watt

Performance and power are influenced by several factors in multi-core SIMT with hot spares. When a system employs hot spares, system performance and power can be modeled with a random variables distributed based on the available redundancy and manufacturing defect density. A die is considered operational if it integrates the required number of functional cores, each with the required number of functional lanes, etc. However, not all operational systems (dice) are fault free when systems are fabricated. This is the case since redundancy has been added to the system and different derivative designs are considered operational: they may meet the minimum system requirements in different ways. To evaluate such systems, we introduce a metric, *expected performance per watt* ($E[PPW]$), that captures the performance and power of a multi-core SIMT system by considering the resulting population of operational dice (denoted *derivative configurations*). $E[PPW]$ is similar to previously introduced metrics such as *expected performance* [2] (refer to Chapter 4) with the difference that it also captures the power consumption of *derivative configurations*. *E[PPW]* is the *expected performance per watt* of a given system in the presence of defects:

$$E[PPW] = \sum_{C_i \in \{S\}} (Perf(C_i)/Pwr(C_i)) \times P(C_i|D), \tag{6.1}$$

where S is the *operational set*, derivative configurations with no defects plus the designs with defects that still meet the minimum system requirements. $Perf$ and Pwr are the system performance (the inverse of benchmark execution time) and the system's total power consumption when running the benchmark, respectively. P is the probability of a specific *derivative configuration's* occurrence in the population of dice given a particular defect density (D).

A huge number of performance as well as power simulations should be run to calculate $E[PPW]$ accurately. This is the case since, for instance, a single manufactured system with six cores, and 32 lanes per core, results in a population of dice with millions of possible configurations when three redundant cores with two lanes each are added. In Section 4.1.2, we introduced grouping symmetric *derivative configurations* to calculate $E[P]$. However, we showed that calculating $E[P]$ exactly is intractable even for small numbers of groups of *derivative configurations*, e.g., a few hundreds [13]. Calculating $E[PPW]$ requires one performance as well as one power simulation for each group of symmetric *derivative configurations*. Hence, $E[PPW]$ must be estimated.

6.1.1 Estimating E[PPW]

With inspiration from what we did in Chapter 5, we estimate $E[PPW]$ by evaluating just the most likely *derivative configuration* for each system, and use it to approximate the performance and power of other derivative configurations. Hereafter, we use $\hat{E}_s[PPW]$ to denote the *approximated expected performance per watt* using the most likely *derivative configuration* (*single-*

evaluation technique). We show that the relative error of $\hat{E}_s[PPW]$ is not only smaller than the measurement error of $E[PPW]$, but also it can be utilized to identify an acceptable set of near-optimal solutions.

Experimental Setup

To evaluate the accuracy of $\hat{E}_s[PPW]$ we conduct some experiments on three different design sets that cover small, medium, and large systems. The configurations of *Small*, *Medium*, and *Large* systems are selected based on real processors: NVIDIA Tegra series [65], Tensilica Vision P6 [67], and NVIDIA Tesla series [67], respectively. We utilize a cycle accurate SIMT performance simulator, MV5 [43], to measure performance, and we used the approach in Section 5.1 to calculate the probability of occurrence for the *derivative configurations*. To measure the power of each *derivative configuration*, we extract activity traces from MV5 and pass them to the GPU modeling infrastructure, GPUWattch [39]. GPUWattch simulates power (dynamic-, static-, and short-circuit-power) for different architectures of GPUs with an emphasis on NVIDIA GPU architectures in 65nm, the same technology node that we perform our performance simulates by MV5. We use the GPUWattch model for the Quadro FX5600 (G80 architecture) the closest architecture to the SIMT architectures that we are investigating. We simulate the power for different parts of a SIMT processor, cores (MCores as well as RCores), L2$ and crossbar network by using the GPUWattch power model. To have more reliable results, we measure the power of cores individually, and then add them up to calculate the entire cores' power. Otherwise, GPUWattch will lose accuracy

Table 6.1: Architectural parameters for *Small*, *Medium*, and *Large* systems.

MCores Spec.	Small	Medium	Large
MCores	{1, 2, 4}	{4, 6, 8}	{12, 14, 16}
MC D$ Size	{8}	{32}	{64}
MC HTCs	{2, 4, 8}	{16, 32}	{32, 64}
MC Lanes	{1, 2, 4}	{8, 16, 32}	{16, 32}
MC I$ Size (KB)	16	16	16
MC Freq. (GHz)	0.6	0.6	0.6
L2$ (MB)	2	4	4
Area (mm^2)	[50, 53]	[120, 222]	[210, 222]

by averaging the activity of all cores to calculate a single core's power. Note that our systems are not necessarily homogeneous (the systems under investigation may consist of RCores and MCores; refer to Section 4.1). Moreover, as GPUWattch does not support the L2$ and crossbar specifications that MV5 uses for its simulations, we extract their performance traces and utilize the McPAT power simulator [68] to measure their power. Afterward, we add them to the cores' power to calculate the entire system's power.

Design Set and Simulation Setup

As mentioned before, evaluating the $E[PPW]$ of a wide set of designs is intractable due to the huge number of *derivative configurations* that must be evaluated for each system. Therefore, we selected a few baseline designs (systems without redundancy) that are categorized as *Small*, *Medium*, and *Large* in terms of die area. Then, we added three different kinds of redundancies (SL, SSL, and SC) to the baseline systems (refer to Section 4.1) to have 60 systems for each set (*Small*, *Medium*, and *Large*), and we identify those systems where redundancy reduces cost. Next, we evaluate our estimation technique over

these designs to determine the accuracy of $\hat{E}_s[PPW]$. You can find the specifications of the baseline systems in Table 6.1. Note that we assume 8KB of D$ for redundant MCores, while we assume one or two lanes per MCore [13]. Furthermore, we use the same simulation setup that is introduced in Section 3.2.

Accuracy measurement for $\hat{E}_s[PPW]$

To evaluate the accuracy of $\hat{E}_s[PPW]$, we begin by measuring the performance and power of the ten most likely *derivative configurations* of each system from those 180 selected systems in the *Small*, *Medium*, and *Large* subsets (refer to Section 6.1.1). We compare $\hat{E}_s[PPW]$ and we observe that in the worst case this approximation leads to less than 2.1%, 6.3%, and 4.8% relative error over different applications for *Small*, *Medium*, and *Large* configuration sets, respectively. $E[PPW]$ has its own measurement error since power as well as performance measurements are performed by simulators: For example, GPUWattch has reported 13.4% error on average [39]. Thus, the error of approximating $E[PPW]$ with $\hat{E}_s[PPW]$ is less than the power simulator's inaccuracy.

Unfortunately, we do not have a more accurate power simulator for GPGPU architectures in our community. As an alternative solution, we considered implementing GPU configurations on a FPGA and measuring their power. However, implementing GPUs on a FPGA does not necessarily help to measure power more accurately since the implementation details of inner components (such as warp and lanes) of well-known GPU architectures are not known. Moreover, it is reported that the measurement inaccuracy of GPUWattch is

acceptable for PPW design space exploration of GPU processors [39].

As we have mentioned before, we did not include main memory power in our PPW calculation. Not considering main memory power will not increase error in PPW measurements in a way that changes the final results. This is the case since in applications for which execution time is reduced by using hot-sparing, we observe that the L2$ to main memory interactions do not increase with the same pace that hot-sparing reduces execution time. Therefore, the energy consumption of system caused by L2$ to main memory interactions decreases by using hot spares. Also, as execution time decreases, static energy dissipation on main memory decreases as well. Note that static energy consumption contributes a significant portion of total energy consumption for memories (more than 30% in 65 nm technology node [63]). Consequently, not considering main memory power consumption results in measuring PPW pessimistically for applications that hot-sparing reduces their execution time. Since, in this paper, we do not compare different applications in terms of PPW, pessimistic results will not change the conclusion.

6.2 Cost-PPW Design Space Exploration

We propose a DSE approach, which is inspired by ReSPIR [20], to find near optimal solutions in our two-dimensional (cost and PPW) design space. We use this approach to reduce the required processing time of exploring the design space to find optimal solutions; otherwise, we would spend 400K hours on a Core i7 with 8GB of RAM to explore the design space exhaustively, even if we utilize $\hat{E}_s[PPW]$.

6.2.1 Metrics

Cost and PPW are the two metrics that our DSE approach optimizes. We already have explained PPW above. Also, we introduced cost metric and the way that we calculate it for each system in Section 4.2.2.

6.2.2 Design Space Exploration Algorithm

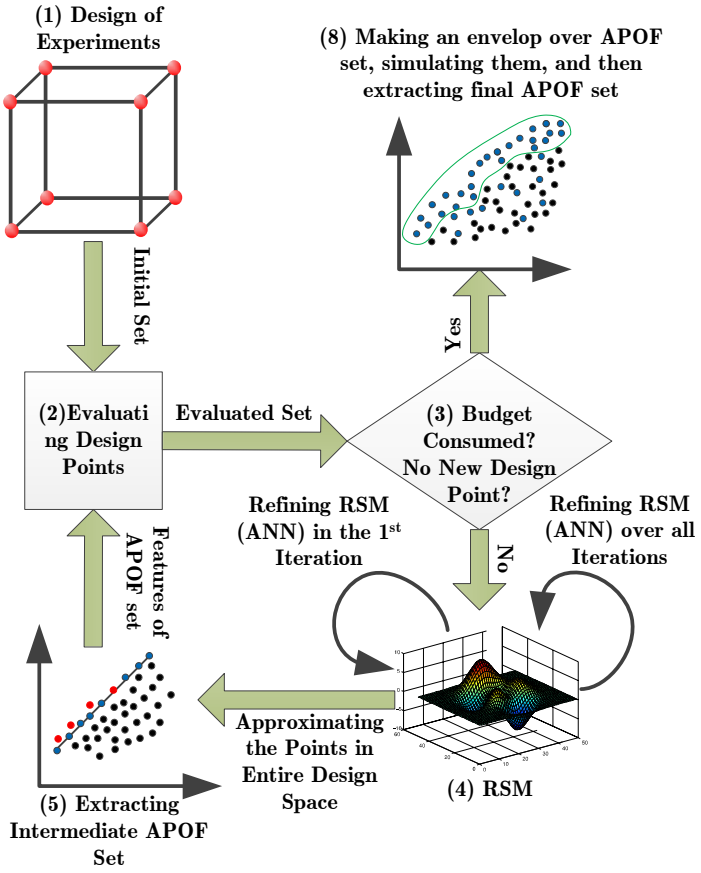


Fig. 6.1: Flow of our design space exploration tool.

We introduce a custom design space exploration (DSE) algorithm to explore the cost and PPW design space more efficiently. Otherwise, we would spend a

considerable amount of computational time (400K hours on a Core i7 with 8GB of RAM) exploring the design space exhaustively. Note that this algorithm is utilized off-line to explore the design space and find near optimal systems in terms of cost and PPW. Afterward, we investigate the optimality of the identified solutions in terms of cost and PPW to study how much improvement we receive by utilizing hot-sparing in SIMT processors.

Figure 6.1 illustrates our DSE approach. We have modified ReSPIR [20] to better address challenges with our particular design problem. In our DSE approach we make the differences between our approach and ReSPIR bold.

1. *Design of Experiments:* A combination of design of experiments (DoE) as well as **random selection** are used to pick up the set of initial design configurations from the feature space (*Initial Set*).

2. *Evaluating Design Points:* The approach uses performance, power, and cost models to simulate the *Initial Set* to have a coarse view of the target design space.

3. We check to see if the processing budget for making a response surface model (RSM) is not yet fully utilized.

4. *Response Surface Model:* The DSE approach uses the *Evaluated Set of Designs* to setup **a neural network based** RSM.

5. *Extracting Intermediate APOF Set:* Based on the RSM, the approach predicts the PPW of the entire design space and extracts the approximated Pareto optimal front (APOF).

6. If any of the APOF solutions have not been evaluated in the initial DoE, they will be evaluated (simulated) and will be added to the *Evaluated Set*.

7. Then, the approach goes back to (4), **until it utilizes all the dedicated budget for improving the RSM, or no new APOF solutions are found to be added to the *Evaluated Set***; otherwise,

8. ***Extracting Final APOF Set:*** **Our DSE approach uses a predetermined budget to evaluate design points near the predicted optimal solutions in hopes finding better solutions.**

Design of Experiments

Our approach begins by performing design of experiments (DoE). We selected full factorial [69] since our feature space (the configuration parameters for SIMT systems) is not big, six parameters, and full-factorial DoE can identify better design points for a space with a limited number of parameters [20].

In our DSE approach, we took a different approach than ReSPIR in DoE. Rather than evaluating just the design points that DoE identifies, we evaluate some random design points until we consume half of the evaluation budget that is dedicated for exploring the design space. We did this since we observed that in our problem, which does not consist of many design points (about 10K), it is better to decrease the initial inaccuracy of the RSM model by consuming a considerable portion of the processing budget (in our approach, 50%). For example, if DoE identifies 3% of design points, we select 7% of other design points randomly and we add them to that 3% to provide an *input set* with an

acceptable size for the first round of training the RSM. Note that we consider evaluating 20% of design space as a tractable computational budget. As we already have mentioned in the beginning of this section, evaluating 20% of our design space requires 80K hours on a Core i7 system, which is tractable using a high performance computing server. Therefore, we evaluate 10% of the entire design space (a half of processing budget) for training the RSM in the first round.

Evaluating Design Points

As mentioned in Section 6.1, we utilize a combination of the MV5 performance simulator and GPUWattch power models to evaluate $\hat{E}_s[PPW]$ of design points (processor configuration). We use our cost model to calculate the cost of each design points (see Section 6.2).

Then, we breakdown the evaluated design points, which are the *input set* for the RSM, into three non-overlapping sub-sets: 75% for a *training set* (that is used to tune the RSM) 15% for a *validation set* (a set that is used to evaluate the accuracy of the RSM). and 10% for a *test set*. We chose this breakdown after observing that if we make either the *validation set* or *training set* too small, *test set* accuracy suffers; the above breakdown appears to result in models with the least mean-squared-error (MSE).

Response Surface Model

In our DSE approach, the relation between architectural parameters and PPW is very complicated. Therefore, we utilize a non-linear artificial neural

network (ANN) regression model to predict $\hat{E}_s[PPW]$. An ANN is a model of computation that is based on biological neural networks which are especially useful in the field of estimating and forecasting in complex systems that can depend on a large number of features [70]. A large variety of ANNs have been developed and widely used in many fields, such as function approximation, classification, and data processing. Since there is no relation between the sequence of the design points (inputs to the ANN) in our design space, we use a fully connected feed-forward ANN, instead of a recurrent or a convolutional neural network. We use the *sigmoid* activation function for each hidden unit in the ANN model. Also, we use back-propagation with gradient descent optimization to train our network and minimize its error.

We perform two refinements on the RSM model: *Refining RSM (ANN) in the 1st Iteration*, and *Refining RSM (ANN) over all Iterations*. Regarding *Refining RSM (ANN) in the 1st Iteration*, it finds the best ANN topology (i.e., the numbers of hidden layers and hidden units per layer) that leads to the highest accuracy. Since the number of features (inputs) is limited, six, and the size of *input set* is moderate (couple of hundreds), searching for the $Best\_Topo$ of the ANN exhaustively is feasible. Note that each ANN evaluation requires roughly 10s on a Core i7 system with 8GB RAM. In this regard, we investigate a range of topologies for ANN (#hidden layers $\in$ [1, 10], #hidden-units per layer $\in$ [6 to 24]). Then, by using the *training set* (75% of *input set*), we evaluate this range of ANN topologies by using the five-fold cross validation, and we choose the topology for ANN that has the least MSE over the *validation set*, and we call it $Best\_Topo$. For example, the $Best\_Topo$ for

FFT application over the *Small* set, is an ANN with two hidden layers each with 12 hidden neurons.

During *Refining RSM (ANN) over all Iterations*, we re-train the ANN several times (e.g., 25 times) over a constant *training set*. This results in ANN with different inner weights. Then, we calculate their MSE over the *validation set*. However, we do not select the ANN with the lowest MSE for those 25 training trials. We have observed that if we select the ANN with the lowest MSE at this stage, this does not necessarily lead to the best ANN regression model for finding optimal solutions. For example, consider Figure 6.2. In this figure, we illustrate the performance of the ANN prediction model (MSE) versus the distance between approximated POF (APOF) and real POF (RPOF) for FFT. To do so, we re-train *Best_Topo* for the ANN several times with a constant *input set*, over each design sets (*Small*, *Medium*, and *Large*), and we measure the accuracy of the model in finding optimal solutions in terms of average distance from reference set (ADRS). ADRS calculates the distance between two POF (e.g., real POF (Π) and APOF (Λ) [45]):

$$ADRS(\Pi, \Lambda) = \frac{1}{\mid \Pi \mid} \sum_{x_R \in \Pi} \left(\min_{x_A \in \Lambda} \{\delta(x_R, x_A)\} \right), \tag{6.2}$$

where δ calculates a normalized distance of design points in Π from the corresponding closest design points in Λ. As is depicted in Figure 6.2, there is no obvious relation between the performance (MSE) of a ANN prediction model and its ability to find near-optimal solutions. This is the case since the RSM's accuracy does not have a direct relationship with finding better approximated optimal solutions in a design space. For example, assume a hy-

pothetical case in which we have an RSM that always estimates the metrics of design points pessimistically with a 30% relative error. In this case, the RSM will have a mean squared relative error of 9%, and even though this is a high error, the RSM can distinguish the real optimal solutions completely since the RSM does not change the relative positions of the design points in that design space. Therefore, we looked for another metric that can better represent the capability of a ANN response surface model to find optimal solutions.

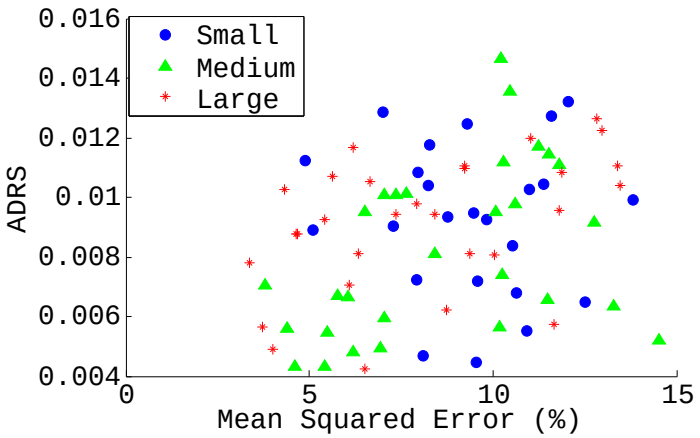


Fig. 6.2: Performance of the ANN prediction model (MSE) versus ADRS for FFT. There is no obvious relationship between accuracy and the quality of the resulting set.

We observed that if the correlation between the MSE of *training* and *validation* sets over epochs is high, we will have a lower distance (ADRS) for the approximated and real optimal solution sets. Epoch is a measure for the number of times that all of the training vectors are used to update the weights. In Figure 6.3, we illustrate the correlation between the *training set* and *validation set* of different ANN configurations ($Best\_Topo$ with different inner weights) against number of epochs as well as their corresponding ADRS numbers for

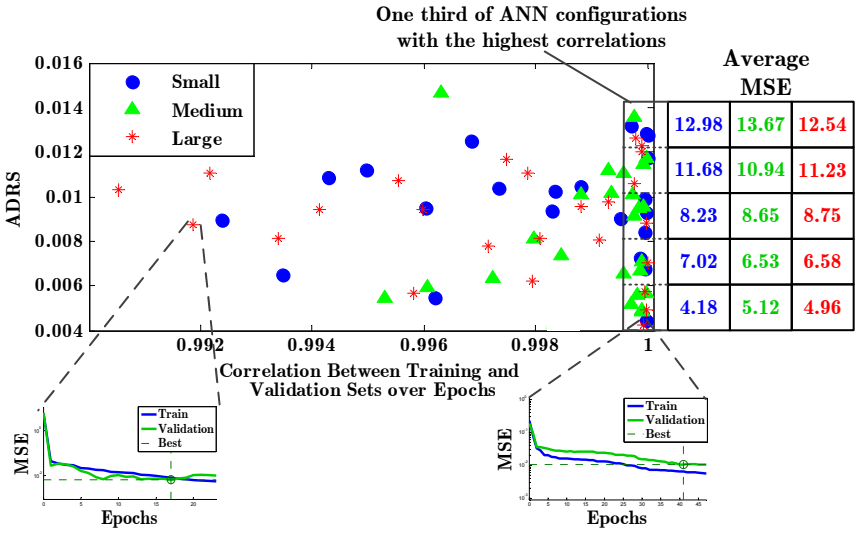


Fig. 6.3: Correlation of the *training set* and *validation set* over epochs and ADRS for FFT. The average MSE (%) for 30% of the most correlated ANN are reported in the right-hand side table.

the three design set (*Small*, *Medium*, and *Large*) for FFT. The best results in terms of ADRS happen when correlation is high. However, designs with the highest correlations spread over the ADRS axis (between 0.4% to 1.4%), and therefore, examining correlation is insufficient. For example, a ANN with the highest correlation for *Small* set leads to 0.9% ADRS, which does not give the best ADRS. Hence, we select the one-third of ANN that have the highest correlation between their *training* and *validation* sets for their MSE against number of epochs. Then, we calculate their average MSE. This is reported in the inset table. As is reported in the table, ADRS and MSE have a direct relationship in this selected 33% setups. Therefore, we select ANN weights that lead to the lowest MSE and call it $Best\_ANN$. We consider the 33% of ANNs with the highest correlation, since after performing several experiments, we observed that the best ANN to predict optimal solutions will be in that

33% ANN setups.

Extracting Intermediate APOF

Our DSE approach has higher accuracy when approximating design points near optimal solutions. The DSE approach uses the $Best\_ANN$ model to approximate the $\hat{E}_s[PPW]$ of the entire design space. Afterward, it compares them with the optimal solutions that are found from the previous iteration. It identifies any new dominant points that have not been evaluated (illustrated in red colors in Figure 6.1). Then, our approach simulates them. Consequently, the DSE approach adds them to the *input set* that is used to train and evaluate the ANN model. In this way, our DSE approach enriches the *input set* with design points from the APOF. In this way, selected design points are chosen close to real optimal solutions, consequently the RSM will have less error near optimal solutions.

Extracting Final APOF

Either when the processing budget for training the RSM (ANN) model is consumed, or no new dominant design point is found when extracting the intermediate APOF, our DSE approach *extracts the final APOF*. We do not consume all the processing budget (20% of the entire design space) for making the RSM model more accurate since we observed that the accuracy of RSM model does not improve after couple of iterations. Also, we observed that some points from the real POF are placed close to the APOF, though our DSE approach cannot find them. This is due to the error of the RSM model: some

dominant solutions are kept out of real POF, while the RSM locates them close to the APOF. Therefore, after couple of iterations and re-training the ANN regression model based on the evaluated points from the intermediate APOF, we stop this process, and we make an envelop over the APOF set to come up with a set that consists of the APOF solutions and some nearby design points (see the top chart in Figure 6.1). Then we consume the remaining budget (5%) to explore those nearby designs. Afterward, we extract the dominant designs and call them the *final APOF set*.

6.2.3 Experimental Setup

To evaluate our DSE approach, we conduct a set of experiments. Our design space has two objectives, cost and PPW. Manufacturing cost is the money required to fabricate a system in the presence of manufacturing defects. We introduced the cost model in Section 4.2.2. Note that since the calculation for system cost is neither complicated nor time consuming (on average it takes three seconds per design), we evaluate the cost of all the design points, instead of predicting them with a regression model. PPW is used to measure how efficient our processor is in terms of energy consumption. We already have defined PPW and how we estimate it ($\hat{E}_s[PPW]$).

Design Sets

We use design sets similar to those in Section 6.1.1. Rather than having a few number of design points per set (*Small*, *Medium*, or *Large*), we increase the systems' die-area intervals: [50, 67], [120, 146], and [210, 243] for *Small*,

Medium, and *Large* sets, respectively, resulting in 800 design points (SIMT processor configurations) per set.

6.2.4 Comparing With Conventional Methods

We compare the result of our proposed DSE algorithm, in terms of ADRS, against: 1) consuming all the budget to randomly evaluate design points, 2) selecting $Best\_ANN$ with respect to the least MSE, and 3) the original ReSPIR [20]. In Table 6.2, we report the ADRS for each approach over different applications and design sets. In most cases, our method for selecting $Best\_ANN$ (see Section 6.2.2) performs better than using the best MSE to select $Best\_ANN$. Also, our approach can find the real POF (RPOF) in cases where it consists of only a small design points (e.g., LU and SVM).

We observe that ReSPIR usually converges after consuming less than half of the budget that we considered tractable. However, it finds APOF that are almost three times farther from the POF that our approach identifies. For example, for ShortestPath application and over the *Small* set, if designers select the near optimal solutions using ReSPIR, those systems will be relatively 7% and 8.2% worse than the selected points by our approach in terms of cost and performance respectively. We hypothesize that the design points that DoE identifies to train the RSM for the first iteration are not sufficient, and the resulting error is quite high. For example, for the *Small* sets, the DoE numbers just about 60 candidates (design points). This leads to high MSE (e.g., 15.3% for FFT application). Even though MSE decreases in the following iterations to 7.7%, the weights of the initial RSM have not been set properly to let the

next iterations reduce the RSM's error effectively. Therefore, ReSPIR's cannot find more *intermediate* dominant solutions, and it stops after a few iterations.

Our approach finds considerably better near-optimal solutions over all applications except LU. We compare the APOF that are found by our proposed tool to the original ReSPIR in terms of cost and PPW separately. Afterward, we calculate the average relative difference between these APOF in terms of cost and PPW. Then, we report their minimum differences, over different design sets, in the column *Avg. Rel. Diff.* Regarding LU, in the RPOF of the *Small* set there are a few outliers that none of the methods can find.

Table 6.2: Comparison of the distances of APOF and RPOF for different methods (*Random, Best MSE, Proposed, and ReSPIR*) in terms of ADRS, and over *Small*, *Medium*, and *Large* sets.

	Random, ADRS (%)			Best-MSE, ADRS (%)			Proposed, ADRS (%)				ReSPIR, ADRS (%)			
	Small	Med.	Large	Small	Med.	Large	Small	Med.	Large	Avg. Rel. Diff. (%) {Cost, PPW}	Small	Med.	Large	Eval. (%)
SP*	4.5	3.8	2.6	1	1.1	0.8	0.4	0.6	0.8	{-4.4, +8.4}	2.8	2.8	1.4	7.4
FFT	2.8	3.2	2.6	1.1	0.7	0.8	0.4	0.4	0.5	{-5.1, +6.9}	1.0	1.9	2.3	10.4
HOTSPOT	2.5	2.8	2.4	1.3	0.9	1	0.5	0.6	0.4	{-3.8, +8.2}	1.7	2.0	1.4	8.5
FILTER	4.5	3.6	5.4	1.5	1.7	0.9	0.3	0.6	0.4	{-4.7, +6.6}	1.9	1.9	2.2	11.4
KMEANS	2.7	3.7	3.9	1.3	1.7	0.9	0.3	0.7	0.7	{-4.28, +6.8}	2.9	1.1	4.1	7.8
LU	5.4	4.6	3.9	3.3	1.2	0	3.1	0	0.3	{+3.7, -4.3}	2.7	1.4	2.8	8.4
SVM	3.7	3.9	2.6	0	0.5	0	0	0.5	0	{-5.2, +8.8}	1.2	2.2	1.6	9.4
MS+	4.3	3.7	4.9	0	0.4	1.3	1.7	0.5	1.3	{-2.6, +3.4}	2.1	1.9	2.3	10.2

*: SHORTESTPATH, +: MERGESORT

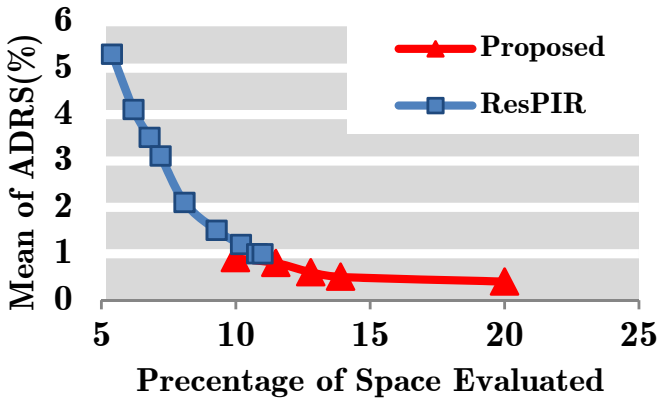


Fig. 6.4: Mean ADRS changes against varying the percentage of design space evaluated over *Small* designs and FFT application.

To test the sensitivity of our approach to the evaluation budget, we decreased the evaluation budget. The average ADRS of both approaches (our approach and ReSPIR) for FFT over the *Small* set is illustrated in Figure 6.4. We ran each approach several times over this set and application, and plotted the average ADRS (the 95% confidence interval is 0.75%). Our approach consumes 10% of the evaluation budget to develop the first RSM, and its APOF is better in terms of mean ADRS than ReSPIR. This is the case since our approach builds its model based on $Best\_ANN$, rather than $Best\_Topo$. However, the main difference is when ReSPIR cannot find a better APOF and stops, while our approach continues.

The way that the ANN-based RSM is developed (e.g., choosing between $Best\_Topo$ and $Best\_ANN$) has a considerable effect on how well it finds near-optimal solutions. An ANN model with lower error (MSE) does not necessarily provide better results in terms of near-optimal solutions. In our future work,

we intend to change the ANN's objective function to optimize the model with respect its APOF, rather MSE.

6.3 Results

6.3.1 Effectiveness of Adding Hot Spares on PPW

We have conducted experiments to study the effectiveness of adding hot-spares to baseline systems (SIMT processors without redundancy) in terms of PPW. We have also performed a design space exploration to determine what types of redundancy most cost-efficiently improve PPW.

6.3.2 Experimental Setup

In Section 6.2.3, we defined and compared our ANN model with ReSPIR over three sets (*Small*, *Medium*, and *Large*). Each set consists of 800 designs. We study these sets to observe the effectiveness of adding hot-spares to systems in terms of PPW.

We model the performance and power of each application over each configuration by using MV5 and GPUWattch tools. We utilize the cost model that is presented in Section 4.2.2 to extract the yield and cost of each configuration.

6.3.3 Analyzing Performance and PPW

In Figure 6.5, we use expected performance ($\hat{E}_s[P]$ refer to Chapter 5) to measure the performance of a multi-core SIMT system by considering the population of dice that results from a given defect density ($0.025/\text{cm}^2$). To

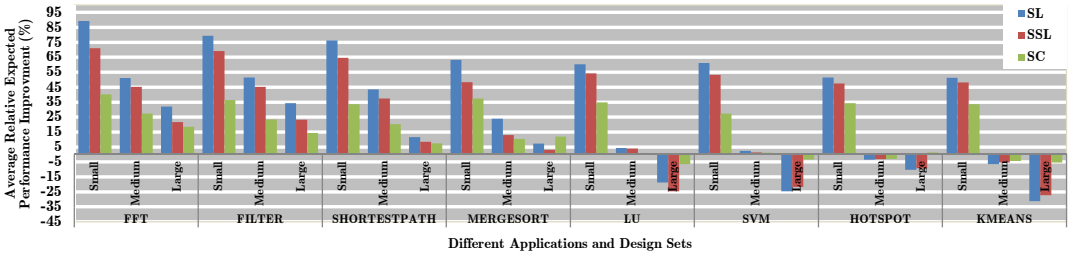


Fig. 6.5: Average relative estimated expected performance (%) [2] improvement by adding hot spares to different configurations of SIMT processors for different design sets (*Small*, *Medium*, and *Large*) over different applications.

calculate $\hat{E}_s[P]$ we simulate the performance of each possible derivative configuration of a system (with defects) and weight it by its likelihood of occurrence. For example, we observe that when *Large* systems utilize SL, on average 29% of the dice are fabricated fully operational in the field, while 37% have one failed lane per core, and 22% experience two failed lanes per core (all of the redundancy is consumed to increase yield). Therefore, a considerable portion (66% = 29% + 37%) of dice for these systems have operational redundancies in the field, which can be utilized to improve system's performance (refer to Chapter 4). Note that this portion of dice increases as the size of systems decreases due to the lower failure probability of smaller systems. As we expected, hot-sparing is effective the most for *Small* systems (it can increase the performance of these systems up to 85% on average). This is mainly due to the fact that *Small* systems do not have many processing elements (cores and lanes). Consequently, any type of hot redundancy is less likely used as spares during fabrication (the system baseline yield is high). As a result, most dice have functional hot spare units which can be used to gain a considerable average performance improvement in the field for these systems.

As systems become larger, this relative performance improvement becomes smaller, since 1) the relative number of redundant processing components (hot cores and lanes) that are added to the baseline systems decreases, 2) their yield is lower than *Small* systems, and therefore, these systems use a larger portion of hot spare units to address defects. Also, we observe that for some applications (e.g., KMEANS and HOTSPOT) adding hot-spares not only does not increase performance of *Medium* and *Large* designs, but degrades it. This performance degradation is more significant when we utilize micro-architectural hot spare elements (SL and SSL). We hypothesize that this performance loss is due to inter-thread data-dependencies as well as the many synchronization points that these applications have [48, 50]: performance is restricted by that of the narrow hot RCores when the wide MCores wait to receive data from them. Furthermore, we observe that for some applications (e.g., LU and SVM), adding hot spare cores for large systems causes a degradation of $\hat{E}_s[P]$. We hypothesize that this originates from greater L1$ coherency latency in *Large* systems. We expect this is due to their limited number of parallel threads. Moreover, we observe that hot-SL outperforms hot-SSL in terms of improving $\hat{E}_s[P]$ as the probability of having functional spares after fabrication is higher in this case.

In Figure 6.6, we illustrate the average relative improvement of systems with hot-sparing compare to baseline, in terms of $\hat{E}_s[PPW]$, over different applications and different design sets. For *Small* systems, on average, we observed improvement for every application (up to 17%). This is the case since the relative improvement in performance is greater than the resulting

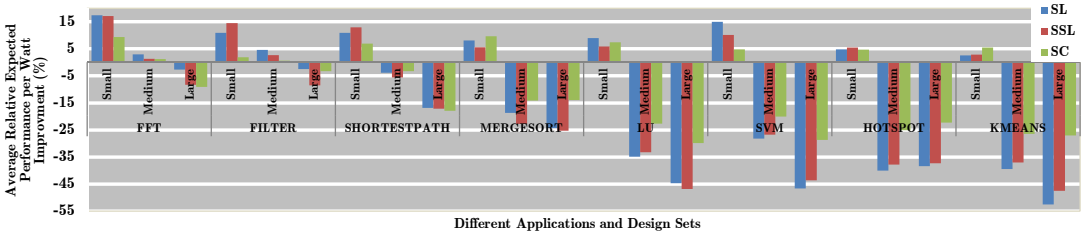


Fig. 6.6: Average relative expected performance per watt improvement by adding hot spares to different configurations of SIMT processors for different design sets (*Small*, *Medium*, and *Large*), and over different applications.

relative the power overhead. This phenomena originates from the fact that, in *Small* systems, a considerable portion of power is consumed in the L2$ and crossbar (e.g., ∼34% in FFT). The relative power increase due to adding redundancy in the L2$ is not as large as the relative power overhead of adding hot spares. Consequently, when we add hot spares in the form of MCores or RMCores, power overhead for the system does not scale with the number of redundant cores, while performance benefits significantly from the hot spares. Also, we observed that $\hat{E}_s[PPW]$ improvement for Hot-SL is lower than Hot-SSL over *Small* systems. This is because the greater relative performance improvement that Hot-SL provides over *Small* systems cannot compensate for its larger relative power consumption compare to Hot-SSL.

However, as systems become larger, average relative $\hat{E}_s[PPW]$ does not improve like in *Small* systems. In *Large* systems, adding hot spares causes a degradation in terms of $\hat{E}_s[PPW]$. This happens since the relative performance gain that the systems receive by adding hot spares are not considerable enough to compensate for the power overhead that redundant cores (RCores or RMCores) add to the systems. Moreover, some applications experience

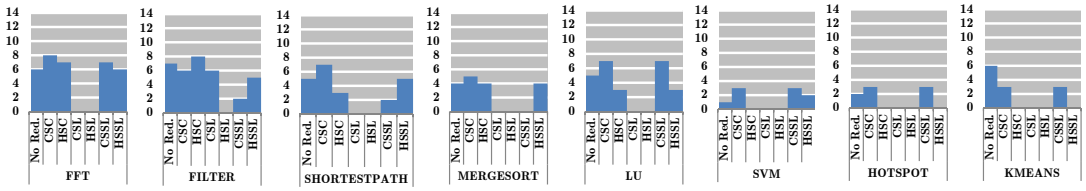


Fig. 6.7: Histogram of optimal solutions based on the type of redundancy that they utilize—cold spare core (Cold-SC), hot spare core (Hot-SC), cold spare lane (Cold-SL), hot spare lane (Hot-SL), cold shared spare lane (Cold-SSL), and hot shared spare lane (Hot-SSL).

performance degradation due to the performance overhead that heterogeneity (adding hot narrow cores beside large wide cores in *Large* systems) have, and consequently, these applications suffer more by adding hot spares in terms of $\hat{E}_s[PPW]$.

Hence, hot-sparing is observed as a quite effective method to improve PPW for small systems (SIMT processors consisting of a few narrow cores). It can improve PPW significantly (e.g., up to 17% for hot-SSL over FFT) without degrading system performance.

6.3.4 Cost-PPW Design Space Characterization

Experimental Setup

To find and study near-optimal solutions in cost-PPW, we conducted a set of experiments. We selected a range of SIMT multi-core processors, and added to them hot as well as cold redundancies in different forms (SL, SSL, and SC). The specifications for the systems that we studied are in Table 6.3. We added redundancies as described in Section 6.1.1. After adding redundancy, we considered the systems with die-area $\in$ [50 to 250] mm^2. These assumptions

Table 6.3: Parameters for under-investigation design space.

MCores Spec.		Redundant Cores Spec.	
MCores	{1, 2, 4, ..., 16}	RCores	{#MC, #MC/2}
MC D$ Size	{8, 16, 32, 64}	RC D$ Size	8KB
MC HTCs	{1, 2, 4, ..., 64}	RC HTCs	{#MC HTC}
MC Lanes	{1, 2, 4, ..., 64}	MC Lanes	{0, 1, 2}
MC I$ Size	16KB	RMCores	{0, 1, 2}
MC Freq.	0.6 GHz	RC Freq.	0.6 GHz

and specifications for SIMT configurations lead to more than 10K different designs.

We use the same simulation setup that is introduced in Section 6.3.2 to simulate the performance of different systems over different applications. Moreover, to find optimal solutions in this design space we utilize our proposed DSE approach (see Section 6.2) to find near-optimal solutions. Then, we evaluate the performance and power of these near optimal solutions with MV5 and GPUWattch simulators and we estimate their PPW with $\hat{E}_s[PPW]$.

In systems that utilize cold- or no-redundancy regimes, $\hat{E}_s[PPW] = E[PPW] = PPW$. We observe that cold redundant components have no or a negligible effect on performance and power in SIMT processors [11]. Therefore, in a SIMT system, the performance and power of derivative configurations that meet the minimum system requirement are the same as the most likely one.

Characterizing Near-Optimal Solutions

Designers cannot ignore hot-sparing when they are looking for optimal solutions in the cost and PPW design space. In Figure 7.4, we present the histogram of observed optimal solutions over different redundancy types for

different applications. In most applications, hot spares contribute to APOF sets. If we replace those solutions that utilize hot-sparing with their corresponding cold systems, we will lose 19.1% $\hat{E}_s[PPW]$ on average over all applications. Moreover, the relative cost overhead of utilizing hot spares instead of cold sparing varies between 3.4% to 7.5% with a mean of 5.0% over these approximated optimal solutions.

We also observe that SSL dominates SL in terms of $\hat{E}_s[PPW]$. Except FILTER, no application utilizes SL redundancy (cold or hot) in its cost-PPW APOF set. Also, if we compare the optimal solutions that utilize Hot-SL in FILTER with the corresponding Hot-SSL solutions, we observe that Hot-SSL systems have relatively only 2.2% less improvement, in terms of $\hat{E}_s[PPW]$. Therefore, there is no point for designers to consider Hot-SL. This reduces the time complexity for exploring this design space (cost and PPW) significantly: blueby safely removing Hot-SL from the list of redundancies that we study, we can save 4000 processing hour even when we utilize the DSE algorithm.

In terms of PPW, hot-sparing cannot help applications that do not benefit from a considerable performance improvement when hot redundancy is added. This happens since the power overhead due to hot redundancy is greater than the resulting performance improvement for these sort of applications. Consequently, no improvement is observed in terms of $\hat{E}_s[PPW]$ in these applications (e.g., HOTSPOT, and KMEANS). For example, consider a case where a hot spare is halted due to the data dependencies between its threads and other cores' threads. In this case, the hot spare consumes energy, but it does not improve performance. These cores still improve cost, as they are available to

replace defective components; furthermore, clock gating can be used to disable them when PPW would otherwise be degraded.

Hot-sparing does not help large systems in terms of PPW. Optimal solutions that utilize hot-sparing are generally among small to medium sized systems (their die-area varies between $61mm^2$ and $123mm^2$). This is mainly due to the fact that in large systems relative performance improvement of hot spares cannot compensate for their relative power overhead. Our observation can save considerable computational effort for designers that perform design space search. As systems become larger, the complexity of calculating $\hat{E}_s[PPW]$ increases dramatically. For example, PPW simulation of a system consisting of eight MCore with 32 lanes per MCore takes about 15 times longer than a system with two MCores with four lanes each.

Moreover, designers can safely ignore the solutions that their cost increases by adding redundancy as they do not contribute into APOF. We observed that systems with no redundancy in the APOF are always small systems where adding any type of redundancy (cold or hot) increases cost. The area of these designs varies between 50 and 54 mm^2. Note that even though adding redundancy improves fabrication yield, not all redundancy improves yield sufficiently to reduce cost.

6.3.5 Discussion

In 65 nm technology, we observed that hot-sparing is more effective for small systems in terms of PPW improvement. This is because of two facts: 1) In small systems, the probability of manufacturing defect is low (and thus yield

is high). Therefore, small systems with hot-sparing have a higher chance to have functional redundancy. 2) Hot-sparing performance gains are significant since the relative number of processing elements that hot spares add to small systems is substantial.

Furthermore, we observe that these systems are frequently used in different embedded systems. However, as systems become larger, we observed that PPW gains due to adding hot spares decrease. This happens since: 1) Hot redundancies are more often used to increase yield. 2) Hot-sparing performance gains decrease.

While our experiments assumed 65 nm technology, we expect to observe similar trends in more advanced manufacturing technologies. As features shrink, sources of failure multiply. Consequently, we expect that designers will be required to allocate at least as much redundancy as in less aggressive technology nodes, if not more. Therefore, applications that benefited from increased performance and PPW when utilizing hot-spares will continue to do so.

Finally, we observe that hot-sparing is a promising method when compared to conventional methods such as DVFS and clock gating:

1. Unlike conventional methods, hot-sparing addresses both cost and PPW.

2. While redundancy is not expected to decrease, the margins for available for DVFS are [71].

3. Hot-sparing does not increase design complexity significantly compared to conventional methods (e.g., DVFS must be applied in a fine-grained

manner to be effective for SIMT processors [39]).

4. Hot-sparing can be applied jointly with conventional methods to even more effectively improve PPW.

6.4 Conclusion

We showed that hot-sparing is a promising method for some systems and applications to improve PPW and cost. It can be used jointly with conventional PPW improvement methods (such as clock gating and DVFS) to even more effectively improve PPW. In this regard, we studied three classes of SIMT configurations, *Small*, *Medium*, and *Large*, each consisting of 800 design points. We showed that hot-sparing is beneficial in terms of PPW with more than 16% improvement, on average, for applications that experience significant performance improvement by utilizing hot-sparing. Also, we showed that hot-sparing should not be used for large processor configurations. The relative performance improvement that is gained by utilizing hot-sparing cannot compensate for the associated relative power overhead in these systems. Moreover, we observed that micro-architectural redundancy (e.g., Hot-SSL) performs better than architectural redundancy (i.e., Hot-SC) in terms of PPW.

Afterward, we studied cost and PPW in a typical design space, consisting of about 10K different SIMT processor configurations. To make this tractable, we proposed a DSE approach to explore this design space. Our approach performs almost three times better than ReSPIR in terms of finding better near-optimal solutions (measured in terms of ADRS). Then, we showed that systems that employ hot spares are often among the near-optimal solutions

of the design space. Moreover, we showed that if we replace near-optimal solutions that utilize hot-sparing with their corresponding cold redundancy, $\hat{E}_s[PPW]$ degrades more than 19%, while in the worst case, the average relative cost overhead of hot-sparing is not greater than 5.5%. Furthermore, we observed that among micro-architectural redundancy techniques, Hot-SSL improves PPW better than Hot-SL in such that almost no optimal solution utilizes Hot-SL.

Ultimately, designers cannot ignore hot-sparing when exploring cost PPW in SIMT processors. However, by using the hints and the framework that are provided in this work, they can explore this design space more efficiently.

Chapter 7

Hot-sparing for Lifetime-Chip-Performance Improvement

In this chapter, we investigate the usage of hot-sparing to improve lifetime-chip-performance of SIMT systems. In this regard, we study a wide range of SIMT systems, and we show that hot-sparing is a very effective method to improve lifetime-chip-performance for some applications and systems.

Several techniques have taken advantage of existing micro-architectural redundancy to improve lifetime or yield [41]. At the system level, task scheduling and dynamic thermal and reliability management (DTM, and DRM) techniques improve system lifetime, and consequently LCP, [19]. These techniques only target lifetime or LCP; hot-sparing which is orthogonal to dynamic thermal or lifetime management, can improve cost as well.

In this chapter, we investigate the effect of hot spares on lifetime-chip-performance (LCP) in multi-core SIMT processors. The same way we did in the previous chapters, we allocate spare cores, lanes, and shared-lanes in SIMT processors, but we, additionally, enable these components for cost and LCP improvement when possible. blueWe use the power model that we developed in Chapter 6 to measure the temperature and lifetime of systems with hot-sparing. At first glance, it would seem that utilizing hot-sparing would result in lower system lifetime. Utilizing hot-sparing would increase power, and consequently, the temperature of a system, resulting in lower system lifetime: lifetime decreases exponentially with temperature [72]. Consequently, if the performance benefit due to utilizing hot-sparing is not greater than lifetime reduction, LCP decreases. However, likewise what we observed in Chapter 6, the relative performance gain of hot-sparing, for some applications, can be higher than its associated relative power and temperature overhead: hot-sparing can improve LCP while decreasing lifetime. We show that hot-sparing is outstandingly effective for specific types of SIMT processor configurations (small and medium systems) and applications (FFT and FILTER), likewise what we observed for performance improvement in Chapter 5. However, like what we observed for cost and PPW, we show that its effectiveness on LCP decreases as systems become larger. applications as well. For example, hot-sparing can improve LCP more than 75% compared with conventional methods (i.e., cold sparing), on average, for applications that experience significant performance improvement when adding hot spares (e.g., FFT and FILTER). In particular, micro-architectural hot redundant resources (e.g., hot spare lanes) achieve

better LCP improvement than conventional architectural redundancies (e.g., hot spare cores).

7.1 Expected Lifetime-Chip-Performance

When a system employs hot spares, system performance and lifetime can be modeled with a random variable distributed based on the available redundancy and manufacturing defect density (refer to Chapter 4). A die is considered operational if it integrates the required number of functional cores, each with the required number of functional lanes, etc. However, not all operational dice are fault free: because redundancy has been added, systems with a variety of sets of defects may still meet the minimum system requirements.

To evaluate such systems, we introduce a metric, *expected lifetime-chip-performance* ($E[LCP]$), that captures the performance and lifetime of a multi-core SIMT system by considering the resulting population of operational dice (denoted *derivative configurations*). $E[LCP]$ is similar to previously introduced metrics, such as $E[P]$ and $E[PPW]$ (refer to chapters 4 and 6) with the difference that it also captures the lifetime of *derivative configurations*.

E[LCP] is the *expected lifetime-chip-performance* of a given system in the presence of defects:

$$E[LCP] = \sum_{C_i \in \{S\}} (Perf(C_i) \times MTTF) \times P(C_i|D), \tag{7.1}$$

where S is the *operational set*, derivative configurations with no defects plus the designs with defects that still meet the minimum system requirements. $Perf$

and $MTTF$ are the system performance (the inverse of benchmark execution time) and the system's mean time to failure, respectively. P is the probability of a specific *derivative configuration's* occurrence in the population of dice given a particular defect density (D). Note that we consider the performance of a system to be constant during its lifetime. We have observed that redundant resources (SL, SSL, and SC) cannot be utilized to extend the MTTF of the systems. This originates from the fact that for the highly parallel applications that fully utilize SIMT systems, lanes age at roughly the same rate. Therefore, a few redundant resources are not enough to extend the lifetime of the system in any significant way.

A huge number of detailed simulations must be run to calculate $E[LCP]$ accurately. As we mentioned before in Section 4.1.2, a system manufactured with six cores, and 32 lanes per core, results in a population of dice with millions of possible configurations when three redundant cores with two lanes each are added. In that sub-section, we grouped symmetric derivative configurations, calculating a similar metric, *expected performance* ($E[P]$). However, we showed that calculating $E[P]$ exactly is intractable even for small numbers of groups of derivative configurations, e.g., a few hundreds [13]. Calculating $E[LCP]$ requires, for each derivative configuration: one simulation for each of performance, power, and temperature, and at least one lifetime calculation for each configuration sub-component. Hence, $E[LCP]$ must be estimated.

7.1.1 Estimating E[LCP]

Like what we did in Chapter 5, we estimate $E[LCP]$ by evaluating just the most likely derivative configuration for each system, and use it to approximate the performance and LCP of other derivative configurations. Hereafter, we use $\hat{E}_s[LCP]$ notation to denote the *estimated expected lifetime-chip-performance* using the *single* most likely derivative configuration. We will see that not only is the relative error of $\hat{E}_s[LCP]$ smaller than the measurement error of $E[LCP]$, but $\hat{E}_s[LCP]$ can also be used to identify near-optimal solutions in the cost and LCP design space.

Experimental Setup

To evaluate the accuracy of $\hat{E}_s[LCP]$ we conduct experiments on three different design sets that cover small, medium, and large systems. We calculate LCP using $\int_{t=0}^{MTTF} Perf(t)dt$, where $Perf(t)$ is the instantaneous performance of a system while it is running an application. To evaluate the MTTF of SIMT processors, we adopt a Monte Carlo Simulation (MCS) based lifetime evaluation framework [73]. It models component failure by using three temperature-dependent wear-out failure mechanisms: electro-migration (EM), time-dependent dielectric breakdown (TDDB) and thermal cycling (TC) [74]. A Weibull failure distribution is used to model each of these mechanisms [75]. The MTTF of each failure mechanism per component is normalized to 30 years for the characterization temperature of 345K [76].

To calculate system MTTF, we first generate a system's floorplan using ArchFP [77]. Then, to measure the power of each derivative configuration, we

follow the steps that are introduced in Section 6.2.3. We extract activity traces from MV5 [43], a cycle-accurate SIMT performance simulator. These traces are then processed by GPUWattach, a GPU dynamic-, static-, and short-circuit-power simulator, with an emphasis on NVIDIA GPUs in 65nm [39]. Afterward, using the system's floorplan and the per-component power data, the steady-state temperature for each component is calculated using Hotspot [78]. For a fair comparison between the chips' temperature, and consequently, the lifetime of different systems, we make some assumptions regarding heat transfer characteristics to calculate the heat dissipation over a system's die: 1) heat sink sizes do not grow proportionally with respect to a system's die size. It is very unlikely that a chip maker design a specific heat sink for each chip. We consider three different heat sink sizes to match systems' die sizes (refer to Table 7.1). Also, we assume that the heat spreader's size is proportional to the heat-sink size. 2) We utilize non-active heat-sink models. 3) For the rest of thermal model parameters, we assume they are constant over all systems (i.e., we use HotSpot's defaults).

Each components' temperature is used to shape its failure distribution for each failure mechanism. This distribution is sampled to determine the failure time of each component for a sample system. During the lifetime simulation of a sample system, a mode detector that checks the number of remaining components after each component failure; system failure occurs when insufficient resources remain for the system to meet minimum system requirements. We observe that 10,000 sample systems is sufficient to estimate system MTTF with a 95% confidence interval of less than 1%.

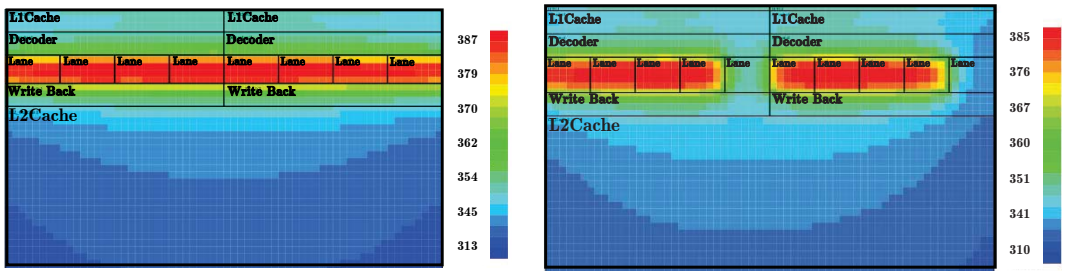
Table 7.1: Heatsink specification for *Small*, *Medium*, and *Large* systems.

MCores Spec.	Small	Medium	Large
Heatsink size (mm^2)	400 [79]	1500 [79]	3000 [80]
Heatsink thickness (mm)	30	40	60

Design Sets and Simulation Setup

Given that calculating LCP for each derivative configuration on average takes two hours on a Core i7 system with 8 GB RAM, evaluating $E[LCP]$ of a wide a set of designs is intractable. Therefore, we selected a few baseline designs (systems without redundancy) that are categorized as *Small*, *Medium*, and *Large* in terms of die area; likewise what we did in Section 6.1.1. The specifications of the baseline systems are in Table 6.1. Also, the specification of heatsink for these systems are presented in Table 7.1. Also, we use the same simulation setup that is mentioned in Section 3.2, while we assume that a system runs a single application during its lifetime.

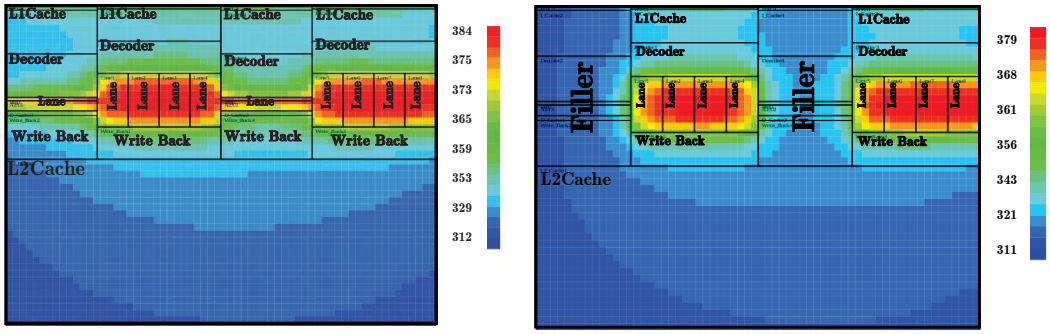
Accuracy measurement for $\hat{E}_s[LCP]$



(a) (b)

Fig. 7.1: Baseline (a) and cold spare lane (b) heat maps.

To evaluate the accuracy of $\hat{E}_s[LCP]$, we begin by measuring the performance and lifetime of the ten most likely derivative configurations of each



(a) (b)

Fig. 7.2: Hot spare lane (a) and Filler (b) heat maps.

design. These measurements cover 99.1%, 98.2%, and 96.3% of the operational derivative configurations for *Small*, *Medium*, and *Large* systems, respectively. In the worst case, $\hat{E}_s[LCP]$ estimates $E[LCP]$ with less than 5.6%, 15.2%, and 11.6% average relative error over different applications for *Small*, *Medium*, and *Large* configuration sets, respectively. Note that the lifetime model is probabilistic, and contributes its own error, that, in some cases, could be near 30%, based on the selected model and the accuracy of the calculation for the system's temperature [75].

To better understand how adding redundancy affects system lifetime, and consequently LCP, we experiment on a case study system consisting of two cores with four lanes per core. We add Cold-SL (Cold-SL), Hot-SL (Hot-SL), and Filler to the baseline system; we observe that the system's temperature decreases when adding redundancy, whether cold or not.

As we infere from the comparison of Figures 7.1 (a) and (b), the baseline system dissipates its heat on a larger die-area when we add cold spare lanes. This reduces system temperature by 1.3°C, and consequently, increases system lifetime by 1.8x. If we add one Hot-SL per core (Figure 7.2(a)), the overall

die's temperature decreases by 2.3°C, a greater drop than with cold sparing. Hot spares require more area, increasing the contact surface between the heat spreader and die, leading to better heat transfer between the die and heat-sink, and compensating for their additional power consumption.

For the sake of comparison, we substitute hot spares with Filler (inactive silicon) on die (Figure 7.2(b)). We observe that the average temperature of the system with Filler is 1.8°C cooler than with hot-sparing, improving system lifetime by 2.2x compared to the baseline. However, this change increases cost significantly (18.3%), since Filler cannot compensate for manufacturing defects in cores, like hot spare (we assume that defects in Filler do not affect system yield.) This overhead makes Filler impractical compared with hot-sparing.

Redundancy does not work as expected because the thermal behavior on die changes in ways we would not have guessed. By investigating this case study system, we observed that the average temperature of the system with hot spares is lower than its equivalent with cold spares, and it leads to 1.56x higher lifetime eventhough its power is higher than cold spares. However, hot spares cost is 3.4% higher and 2.3% lower than cold sparing and baseline, respectively. Also, we should recall that in many applications hot-sparing improves performance, which in the end helps to improve LCP even more. Therefore, hot-sparing improves LCP the best, while it can reduce cost as well.

7.2 Results

We conduct experiments to evaluate how effectively hot spares improve performance and lifetime (LCP), as well as cost, relative to baseline SIMT systems. We subsequently perform a design space exploration to determine what types of redundancy most cost-efficiently improve LCP.

7.2.1 Experimental setup

We use the same sets of designs that are introduced in Section 6.2.3. We study *Small*, *Medium*, and *Large* sets individually to observe the effectiveness of adding hot-spares to these systems in terms of LCP and cost. We model the performance and LCP of each application over each configuration by using MV5 and the lifetime modeling framework in Section 7.1.1. Yield and cost are determined with the model in Section 4.2.2.

7.2.2 Analyzing LCP

To analyze lifetime-chip-performance of the systems, first we need to analyze their performance in the presence of hot-sparing. We already have done this analysis for the same sets of SIMT systems in Section 6.3.3. In Figure 7.3, we illustrate the cost and average relative improvement of $\hat{E}_s[LCP]$ over different applications and design sets when different types of redundancy (hot and cold SL, SSL, and SC, and Filler) are allocated. When hot spares are added, the lifetime of *Small* systems improves the most (on average up to 3.5x). This is the case since: 1) the relative improvement in performance is more significant in *Small* systems when we use hot spares (refer to Section 6.3.3); and, 2) the

relative increase in die area is more pronounced when hot spares are allocated in *Small* systems (refer to Section 4.3.1), leading to a greater decrease in heat density, and consequently, longer lifetime. Also, we observe that the $\hat{E}_s[LCP]$ improvement of Hot-SSL is always lower than Hot-SL. This is the case since 1) Hot-SL improves performance more than Hot-SSL, as it adds more lanes; and 2) SL expands the die more than SSL, better distributing heat and extending lifetime. When we compare Hot-SSL with Hot-SC, we observe that even though the average lifetime improvement due to adding Hot-SC (1.69x over *Small* systems) is greater than hot SSL (1.67x), the LCP improvement of Hot-SSL is greater in almost all applications and design sizes. This is the case since: 1) the performance of the parallel applications we have used for evaluation often increases with more cores [2] (e.g., FFT, FILTER, SHORTESTPATH), and SSL increases core count more dramatically than SC; and, 2) the area overhead introduced by Hot-SSL has lower power density than that introduced by Hot-SC, as each SSL is narrower than a monolithic SC, resulting in lifetime improvement even when performance degrades (e.g., LU, SVM, HOTSPOT, KMEANS).

We also observe that even though hot-sparing has power overhead, it always outperforms cold sparing in terms of LCP (by at least 30% over all applications). This is the case since: 1) hot-sparing leads to a larger die area than cold sparing, which decreases the on-chip power density more effectively. As a result, paradoxically, systems with hot-sparing experience a longer lifetime on average. 2) In some applications (e.g., FFT, FILTER, SHORTESTPATH), hot-sparing leads to higher performance. Another factor is related to the way

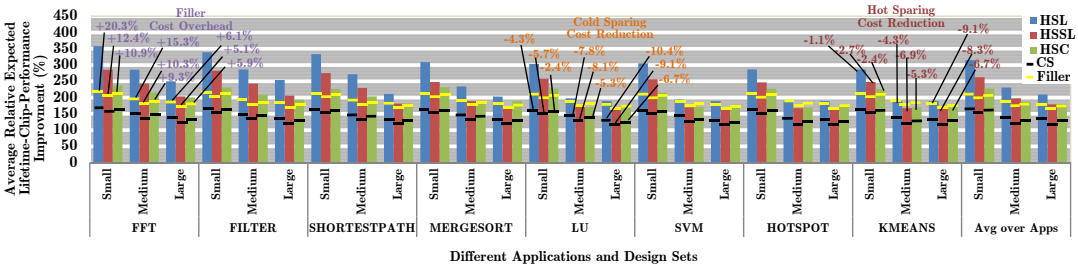


Fig. 7.3: Cost and average relative expected lifetime-chip-performance (ARELCP) improvement compare to baseline systems when adding hot spares (Hot-SL, Hot-SSL, Hot-SC), cold spares (Cold-SL, Cold-SSL, Cold-SC), and Filler to different SIMT processor configurations, across *Small*, *Medium*, and *Large* design sets, and different applications. The lines corresponding to cold sparing (black) and Filler (yellow) indicate the ARELCP improvement that occurs when hot spares are cold instead, or replaced with filler on die.

components accumulate wear. Wear tends to accumulate evenly over hot components: by the time a failure occurs, most of the rest of the system has aged as well, limiting the lifetime extension resulting from cold sparing, as additional failures are likely coming soon.

As systems become larger, the improvement in average relative $\hat{E}_s[LCP]$ is not as significant as for *Small* systems. This happens since the relative performance gain is smaller, as is the relative increase in area. Consequently, we observe the lifetime improvement of disabled hot spares approaches that of cold spares in this case.

Moreover, some applications experience performance degradation due to the performance implications of heterogeneity: adding hot narrow cores beside large wide cores in *Large* systems can slow the entire system if wide cores wait at synchronization points for narrow cores. Whenever hot-sparing reduces performance (e.g., *Medium* and *Large* designs running LU, SVM, HOTSPOT,

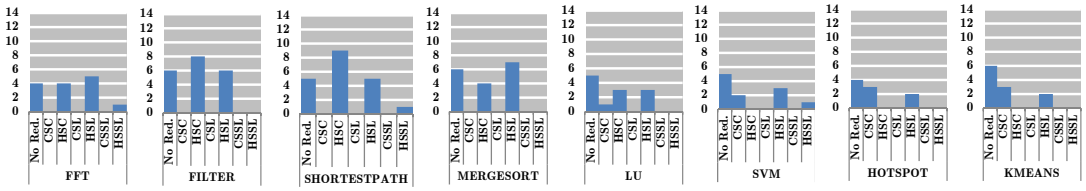


Fig. 7.4: Histogram of optimal solutions based on the type of redundancy that they utilize—cold spare core (Cold-SC), hot spare core (Hot-SC), cold spare lane (Cold-SL), hot spare lane (Hot-SL), cold shared spare lane (Cold-SSL), and hot shared spare lane (Hot-SSL).

KMEANS), designers may disable hot spares; in the worst case, hot-sparing adds 3.2% cost overhead on average compared to cold sparing. We assume that hot spares are turned off in those systems.

Finally, hot-sparing in most cases outperforms simply using the same area as filler. While filler performs better relative to hot-sparing when systems are larger, it never makes sense to use filler, even when hot redundancy degrades performance. Note that if we dedicate the area used by hot spares to filler instead, we add a considerable overhead in terms of cost (e.g., more than 22% over *Small* systems with hot SL), since filler cannot be used to compensate for defects.

In summary, hot-sparing is observed to be quite effective method at improving LCP for small and medium sized systems. These systems consist of a few, relatively narrow cores. When hot redundancy is added, performance usually improves, and power density usually falls, the result being that on average improves LCP up 3x across all applications.

7.2.3 Cost-LCP Design Space Characterization

Experimental Setup

To see how often different types of redundancy strike the best trade-offs in terms of cost and LCP, we select a range of SIMT multi-core processors, and allocate to them hot as well as cold redundancies in different forms (SL, SSL, and SC). We use the same sets of designs as Section 6.3.4. Also, we add redundancy the same way as it is described in that sub-section. We use the same simulation setup that is introduced in Section 3.2 to simulate the performance, power, and $\hat{E}_s[LCP]$ of different systems over different applications. To find optimal solutions in this design space, we utilize the design space exploration algorithm that is introduced in Section 6.2.2.

When systems lack redundancy or employ cold spares, $\hat{E}_s[LCP] = E[LCP] = LCP$. It has been observed that cold spares have a negligible effect on performance [11]. Therefore, in a SIMT system, the performance and LCP of derivative configurations that meet the minimum system requirement are the same as the most likely one.

Characterizing Near-Optimal Solutions

Designers should consider using hot-sparing when they are looking for optimal solutions in the cost and LCP design space. In Figure 7.4, we present a histogram of the distribution of redundancy type among approximately Pareto-optimal (APO) solutions for different applications. In most applications, hot spares contribute to the APO set, and often represent a majority of the solutions. If we replace those solutions that utilize hot-sparing with their cor-

responding cold systems, we will lose 53.2% in terms of $\hat{E}_s[LCP]$ on average over all applications. Moreover, the relative cost overhead of utilizing hot spares instead of cold sparing varies between 2.3% to 6.1% with a mean of 4.3% over these optimal solutions. For a cost increase of 4.3%, we can improve LCP 53.2%, a significant result compared to conventional LCP improvement methods [19].

We also observe that SL dominates SSL in terms of $\hat{E}_s[LCP]$. Except for SHORTESTPATH and SVM, no application utilizes SSL redundancy (cold or hot) in its cost-LCP APOF set. This is mainly due to the fact that SL expands die more and improves performance more than SSL. Also, if we compare the optimal solutions that utilize Hot-SSL in SHORTESPATH and SVM with the corresponding Hot-SL solutions, we observe that Hot-SL systems have relatively only 27.9% less improvement, in terms of $\hat{E}_s[LCP]$. Therefore, there is no point for designers to consider Hot-SSL. This reduces the time complexity for exploring this design space (cost and LCP) significantly: blueby safely removing Hot-SSL from the list of redundancies that we study, we can save 4000 processing hour even when we utilize the DSE algorithm.

In terms of LCP and cost, hot-sparing can even help applications that do not benefit from a considerable performance improvement when hot redundancy is added (e.g., *Large* systems over HOTSPOT and KMEANS applications). This happens since first the systems that do not enjoy hot-sparing in terms of performance are *Medium* or *Large* (refer to Figure 6.5), therefore their cost overhead due to utilizing hot-sparing rather than cold-sparing is low (refer to Figure 7.3). Second, hot-sparing occupies more area than cold sparing on

die. Therefore, its resulting temperature reduction and consequently lifetime extension are more significant.

Moreover, designers do not need to look for optimal solutions among small systems which their cost increases by adding redundancy. In other words, no optimal small systems employ redundancy. We observed that systems with no redundancy in the approximated Pareto optimal front (APOF) are always small systems where adding any type of redundancy (cold or hot) increases cost. The area of these designs varies between 50 and 54 mm^2. Note that even though adding redundancy improves fabrication yield, not all redundancy improves yield sufficiently to reduce cost.

7.3 Conclusion

We showed that hot-sparing is a promising method to improve LCP and cost that can be used jointly with conventional LCP improvement methods (such as DVFS and DTM) to even more effectively improve LCP. In this regard, we studied three classes of SIMT configurations, *Small*, *Medium*, and *Large*, each consisting of 800 design points. We showed that hot-sparing is beneficial in terms of LCP with more than 3.5x improvement, on average, for applications that experience significant performance improvement when utilizing hot-sparing. Moreover, we observed that micro-architectural redundancy (e.g., Hot-SL) performs better than architectural redundancy (i.e., Hot-SC) in terms of LCP.

Afterward, we studied cost and LCP in a typical design space, consisting of about 10K different SIMT processor configurations. Then, we showed that

systems that employ hot spares are often among the near-optimal solutions of the design space. Moreover, we showed that if we replace near-optimal solutions that utilize hot-sparing with their corresponding cold redundancy, $\hat{E}_s[LCP]$ degrades more than 27.9%, while in the worst case, the average relative cost overhead of hot-sparing compared to cold sparing is not greater than 6.1%. Furthermore, we observed that among micro-architectural redundancy techniques, Hot-SL improves LCP better than Hot-SSL in such that almost no near-optimal solution utilizes Hot-SSL.

Ultimately, designers should take into account utilizing hot-sparing when exploring cost and LCP in SIMT processors. However, by designers can explore cost-LCP design space more efficiently by utilizing the hints and the framework that are provided in this work.

Chapter 8

Conclusion and Future Work

As semiconductor feature sizes shrink, devices become more prone to failure in the field, resulting in early performance degradation. Redundancy is now routinely allocated in circuits, micro-architectural structures, or at the system level, to mitigate mounting manufacturing yield losses in multi-core processors. The structure of multi-core SIMT systems makes them particularly suitable for applying redundancy at multiple levels of granularity.

In this regard, we utilized architectural (SC) and micro-architectural (SL) cold redundancy in SIMT systems, and we introduced a new fine-grained redundancy method, SSL. Afterward, we proposed hot-sparing in SIMT systems and we developed their yield and cost models. Also, we investigated cost and performance implications in SIMT systems with redundancy. Since calculating performance of a system with hot-sparing is intractable, we introduced two estimation techniques, $\hat{E}_s[P]$, and $\hat{E}_m[P]$. Then, we investigated the implications that utilizing hot-sparing have on PPW and LCP. We explored cost-PPW and cost-LCP design spaces to find and characterize their optimal solution. To perform this design space exploration, we adopted a DSE algorithm.

In Chapter 3, we utilized core-, lane-, and shared-spare-lane to address yield, and consequently, cost in SIMT processors (cold-sparing). In this regard, we developed yield and cost models for SIMT systems with redundancy and we measured the performance overhead that adding each type of redundancy (spare core, spare-lane and spare-shared-lane) has on a SIMT system. We evaluated SIMT systems with redundancy when they execute a variety of benchmarks. We found that configurations in performance-cost Pareto-optimal front for some applications benefit most from core sparing, with up to 25% cost reduction. However, for most of applications, shared-spare-lane outperforms lane-sparing, reducing cost by up to 20%.

In Chapter 4, we showed that hot-sparing's performance overhead is not considerable for SIMT systems and just by placing a micro-architectural redundancy (spare-lane and shared-spare-lane), in the context of a core, we can turn that redundant core on and improve system's performance. To capture the performance and cost of a SIMT system with hot-sparing, we introduced a new metric, expected performance per cost ($E[P]/C$), and showed that there are some applications that benefit greatly from hot-sparing, and for those that do not, the overhead associated with leaving spares cold is not significant. For a case study system, we observe that $E[P]/C$ improves more than 2.5 and 1.7 times relative to systems integrating no redundancy and cold spares, respectively.

Also, we showed that calculating $E[P]$ accurately is intractable even for a single SIMT system with hot-sparing. Therefore, we introduced a new estimation technique for $E[P]$, $\hat{E}_m[P]$ that trade performance with computational

complexity to estimate $E[P]$ via simulating m most likely derivative designs. Afterward, in Chapter 5, we showed that calculating $\hat{E}_m[P]$ is computationally expensive for design space exploration when individual, detailed, simulations require hours. Therefore, we introduced a new estimation technique for $E[P]$, $\hat{E}_s[P]$. $\hat{E}_s[P]$ evaluates only the most likely configuration, and assumes its performance for all other configurations, reducing simulation by 98%, with no more than 2.6% error in $E[P]$, sufficient for differentiating designs along the Pareto-optimal front during design space exploration. Consequently, designers may add redundancy, and evaluate system performance and cost, with no greater design effort than performance evaluation alone.

In Chapter 6 and 7, we introduced hot-sparing as a technique that addresses cost, and PPW or LCP in SIMT processors for some applications. In this regard, we investigated the performance improvement of hot-sparing to see if it can be used to improve PPW, and LCP in SIMT processors, respectively. Then, we studied optimal solutions in the cost-PPW and cost-LCP design spaces to see what kind of redundancy improves cost-PPW, and cost-LCP the most, respectively. However, since evaluating individual design points (different SIMT processor configurations with redundancy) is time consuming, we adopted a design space exploration algorithm to find near-optimal solutions without evaluating the design space exhaustively. Furthermore, we showed that hot-sparing's PPW improvement on these applications is comparable with the results of conventional techniques (e.g., voltage scaling) and can be utilized together with them to more effectively improve PPW in the systems. Also, we showed that hot-sparing can improve LCP up to 75% on average on different

applications and configurations compared to cold-sparing.

In Table 8.1, we summarize the average relative difference of systems with redundancy (cold or hot) to the baseline systems. We compare these systems in terms of cost, $\hat{E}_s[P]$, $\hat{E}_s[PPW]$, and $\hat{E}_s[LCP]$ over a wide range of SIMT configurations. We observe that as systems become larger SC becomes more effective in terms of cost, while SSL always outperforms SL. We also observe that hot-sparing is the most effective for *Small* systems in terms of $\hat{E}_s[P]$. However, as systems become larger the performance improvement of hot-sparing decreases while it becomes more effective for cost reduction. Cold-sparing has a negligible effect on $\hat{E}_s[PPW]$ as it does not change the performance and power of a system. However, it can considerably improve lifetime and $\hat{E}_s[LCP]$ in *Small* systems by expanding their system die. Also, hot-sparing improves the $\hat{E}_s[PPW]$ of *Small* systems only. However, it is quite an effective method in terms of $\hat{E}_s[LCP]$ over all configurations: it improves performance significantly over different applications and systems, while extending the lifetime of systems by expanding their die as well.

In sum, this research show the alternative choices that designers have to improve SIMT systems with redundancy. For example, when a system architect has an application and finds a system that satisfies its performance, she can select the proper type of redundancy to maximize other design factors (such as PPW or LCP) per cost. Afterward, she can use the frameworks that have been introduced in Chapter 5 to 7 to explore the design space of cost and the target design factor (performance, PPW, or LCP), and characterize any better alternative configuration.

Table 8.1: Comparison of different SIMT systems (*Small*, *Medium*, and *Large*) in terms of average relative difference to baseline systems (%) for cost, $\hat{E}_s[P]$, $\hat{E}_s[PPW]$, and $\hat{E}_s[LCP]$ over different applications. To observe the specifications of *Small*, *Medium*, and *Large* systems refer to Table 6.1.

			SIMT Systems											
		Application	*Small* ([50, 67]mm^2)				*Medium* ([120, 146]mm^2)				*Large* ([210, 243]mm^2)			
			cost	$\hat{E}_s[P]$	$\hat{E}_s[PPW]$	$\hat{E}_s[LCP]$	cost	$\hat{E}_s[P]$	$\hat{E}_s[PPW]$	$\hat{E}_s[LCP]$	cost	$\hat{E}_s[P]$	$\hat{E}_s[PPW]$	$\hat{E}_s[LCP]$
Cold	SL		-4.3	-1.3	-1.3	+75	-7.8	-1.3	-1.3	+51	-10.4	-1.3	-1.3	+30
	SSL		-5.7			+53	-8.1			+42	-9.1			+24
	SC		-2.4	0	0	+68	-5.3	0	0	+48	-6.7	0	0	+28
Hot	SL	FFT	-1.1	+89	+17	+257	-4.3	+51	+3	+186	-9.1	+32	-3	+150
		FILTER		+79	+11	+239		+51	+5	+184		+34	-2	+154
		SP*		+76	+11	+233		+44	-4	+171		+12	-17	+111
		MS$^+$		+63	+8	+209		+24	-19	+134		+7	-24	+103
		LU		+60	+9	+203		+4	-35	+97		-19	-45	+89
		SVM		+61	+15	+205		+2	-28	+93		-25	-47	+86
		HOTSPOT		+51	+5	+186		-3	-40	+89		-10	-38	+89
		KMEANS		+51	+3	+186		-6	-39	+89		-31	-53	+89
	SSL	FFT	-2.7	+71	+17	+286	-6.9	+45	+1	+143	-8.3	+22	-8	+103
		FILTER		+69	+15	+283		+45	+3	+143		+23	-9	+106
		SP*		+65	+13	+175		+37	-6	+129		+9	-17	+81
		MS$^+$		+48	+6	+148		+13	-23	+89		+3	-25	+72
		LU		+54	+6	+158		+4	-33	+74		-25	-47	+67
		SVM		+53	+10	+156		+1	-27	+69		-22	-44	+66
		HOTSPOT		+47	+5	+146		-3	-38	+67		-9	-37	+67
		KMEANS		+48	+3	+148		-5	-37	+67		-27	-47	+69
	SC	FFT	-2.4	+40	+9	+137	-5.3	+27	+1	+115	-6.7	+18	-9	+100
		FILTER		+36	+2	+131		+36	+1	+109		+14	-3	+93
		SP*		+34	+7	+126		+20	-3	+103		+7	-18	+82
		MS$^+$		+37	+10	+132		+10	-14	+87		+12	-14	+89
		LU		+35	+7	+128		+1	-23	+71		-6	-30	+69
		SVM		+27	+5	+115		+1	-20	+71		-3	-29	+68
		HOTSPOT		+34	+5	+127		-3	-25	+69		+1	-22	+71
		KMEANS		+32	+5	+126		-4	-26	+69		-5	-27	+66

*: SHORTESTPATH, +: MERGESORT

Table 8.2: Comparison of different SIMT systems (*Small*, *Medium*, and *Large*) in terms of type of redundancy that they utilize to optimize $\hat{E}_s[P]$, $\hat{E}_s[PPW]$, and $\hat{E}_s[LCP]$ over different applications.

App	SIMT Systems								
	Small ([50, 67])mm^2			*Medium* ([120, 146])mm^2			*Large* ([210, 243])mm^2		
cost	C-SSL			C-SSL			C-SC		
	E[P]	E[PPW]	LCP	E[P]	E[PPW]	LCP	E[P]	E[PPW]	LCP
FFT	H-SL	H-SL	H-SSL	H-SL	H-SL	H-SL	H-SL	NR†	H-SL
FILTER	H-SL	H-SSL	H-SSL	H-SL	NR	H-SL	H-SL	NR	H-SL
SP*	H-SL	H-SSL	H-SL	H-SL	NR	H-SL	H-SL	NR	H-SL
MS+	H-SL	H-SL	H-SL	H-SL	NR	H-SL	H -SC, -SL	NR	H-SL
LU	H-SL	H-SL	H-SL	H -SSL, -SL	NR	H-SL	H-SC	NR	H-SL
SVM	H-SL	H-SSL	H-SL	H -SSL, -SL, -SC	NR	H-SL	NR	NR	H-SL
HOTSPOT	H-SL	H-SL	H-SL	NR	NR	H-SL	NR	NR	H-SL
KMEANS	H-SL	H-SL	H-SL	NR	NR	H-SL	NR	NR	H-SL

*: SHORTESTPATH, +: MERGESORT, †: No Redundancy

Also, in Table 8.2, we summarize systems and the redundancy techniques that they utilize the best for each application. This table provides a guidance for designers to select a proper redundancy technique when they want to optimize their systems for a specific design objective. For example, we show that H-SL outperforms H-SSL and H-SC in almost every applications and system sizes in terms of $E[P]$ and $E[LCP]$, but it losses in terms of cost.

8.1 Future Work

We suggest to improve the applicability of this research by, 1) sharing resources between cores and lanes rather than having one or two spare components per processor or core, 2) activating redundancies adaptively in the field with respect to an application that is running on a system, and 3) estimating the performance of a system with respect to defect density that is fabricated in.

8.1.1 Utilizing a Pool of Shared Redundancy Resources

In multi-core processors, the effect of providing a pool of shared resources on cost, performance, PPW, and LCP needs to be investigated. In Chapter 3, we investigated the effect of architectural and micro-architectural redundancies in terms of cost in SIMT systems. We observed that neither systems with more than one type of redundancy nor more than one redundant component of a given type are performance-cost optimal solutions due to their high area cost. However, as feature sizes shrink and systems become larger, yield losses are mounting due to systematic and random defects. We anticipate the requirement to utilize a pool of shared redundant resources in future processors that are fabricated in more advance technology nodes (e.g., 14nm).

8.1.2 Adaptive Redundancy Regime

For general purpose SIMT systems, an adaptive system configuration controller needs to be designed to determine the best system configuration with respect to the application that is running on the system. This controller can have different objectives to optimize during run-time (e.g., performance, PPW, or LCP). In Chapter 4, we assumed that spare units status (hot or cold) are determined at manufacturing time and due to the application that they are running. We analyzed applications and configurations, and we observed that some of them utilize hot-sparing to improve performance, PPW, or LCP, while there are some that do not. Therefore, when an application and a configuration are determined, a designer can decide how to manage the resources (hot and cold cores) to optimize the systems for performance, PPW, or LCP. However,

this is not the case for general purpose SIMT systems that their applications are determined at run-time only. Therefore, for these systems, designers need to utilize an adaptive redundancy regime to optimize an objective design factor (performance, PPW, and LCP) in the field with respect to an application that is running on the system.

8.1.3 Adaptive Performance Estimation Technique

We suggest to adaptively select a performance estimation technique for systems that their defect density are higher than 65nm. In Section 5.4.5, we observed that when the defect density is 2.7x higher, the accuracy for for Small systems (area less than 100 mm^2) drops: the performance difference for systems with one lane failure is larger than 15%, on average; this results in more than 2.6% error for $\hat{E}_s[P]$. Note that the introduced estimation technique for SIMT systems with hot-sparing that is based on evaluating only the most likely derivative configuration from a system with hot-sparing. Consequently, if we utilize $\hat{E}_s[P]$ to do a design space search when the defect density is high, we might fail to find optimal solutions due to not being sufficiently accurate in the estimating of the performance of systems. To address this problem, it is needed to adaptively in-cooperate a more number of most likely derivative configurations to estimate E[P], $\hat{E}_m[P]$. To do so, the adaptive algorithm should calculate the needed accuracy for an accurate enough design space search, and afterward, it choose a proper m, the number of most likely configurations that are in-cooperated into calculating $\hat{E}_m[P]$. Note that this adaptive performance estimation technique can be applied to estimate PPW

($\hat{E}_m[PPW]$), and LCP ($\hat{E}_m[LCP]$) as well.

References

[1] J. Meng, J. W. Sheaffer, and K. Skadron. Robust SIMD: Dynamically Adapted SIMD Width and Multi-Threading Depth. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, pages 107–118, May 2012.

[2] S. H. Mozafari and B. H. Meyer. Hot spare components for performance-cost improvement in multi-core SIMT. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*, pages 53–59, Oct 2015.

[3] I. Koren and Z. Koren. Defect tolerance in VLSI circuits: techniques and yield analysis. *Proceedings of the IEEE*, 86(9):1819–1838, Sept 1998.

[4] S. Borkar. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. *IEEE Micro*, 25(6): 10–16, 2005.

[5] T. Gupta and A. H. Jayatissa. Recent advances in nanotechnology: key issues amp;amp; potential problem areas. In *2003 Third IEEE Conference on Nanotechnology, 2003. IEEE-NANO 2003.*, volume 2, pages 469–472 vol. 2, Aug 2003.

[6] International Technology Roadmap for Semiconductors. `http://www.itrs.net/Links/2013ITRS/Home2013.htm`, 2013. Accessed: 2015-01-30.

[7] Y. Markovsky and J. Wawrzynek. On the opportunity to improve system yield with multi-core architectures. In *the Proceedings of IEEE International Workshop on Design for Manufacturability and Yield*, pages 1–9, October 2007.

[8] Y. Gao, M. A. Breuer, and Y. Wang. A new paradigm for trading off yield, area and performance to enhance performance per wafer. In *Design,*

Automation Test in Europe Conference Exhibition (DATE), pages 1753–1758, March 2013.

[9] M. Gschwindand, H.P. Hofstee, B. Flachs, M. Hopkin, Y. Watanabe, and T. Yamazaki. Synergistic Processing in Cell's Multicore Architecture. *Annual IEEE/ACM International Symposium on Microarchitecture*, 26 (2):10–24, March 2006.

[10] Maitre, O. Understanding NVIDIA GPGPU Hardware. In Massively Parallel Evolutionary Computation on GPGPUs. *Springer Berlin Heidelberg*, pages 15–34, 2013.

[11] Seyyed Hasan Mozafari, Brett H. Meyer, and Kevin Skadron. Yield-aware Performance-Cost Characterization for Multi-Core SIMT. In *Proceedings of the 25th Edition on Great Lakes Symposium on VLSI*, pages 237–240, 2015.

[12] Y. Gao, Y. Zhang, D. Cheng, and M. A. Breuer. Trading off area, yield and performance via hybrid redundancy in multi-core architectures. In *IEEE 31st VLSI Test Symposium (VTS)*, pages 1–6, April 2013.

[13] S. H. Mozafari and B. H. Meyer. Efficient Performance Evaluation of Multi-Core SIMT Processors with Hot Redundancy. *IEEE Transactions on Emerging Topics in Computing*, 6(4):498–510, Oct 2018.

[14] Premkishore Shivakumar, S. W. Keckler, C. R. Moore, and D. Burger. Exploiting microarchitectural redundancy for defect tolerance. In *Proceedings 21st International Conference on Computer Design*, pages 481–488, Oct 2003.

[15] E. Schuchman and T. N. Vijaykumar. Rescue: a microarchitecture for testability and defect tolerance. In *32nd International Symposium on Computer Architecture (ISCA'05)*, pages 160–171, June 2005.

[16] Premkishore Shivakumar, S. W. Keckler, C. R. Moore, and D. Burger. Exploiting microarchitectural redundancy for defect tolerance. In *Proceedings 21st International Conference on Computer Design*, pages 481–488, Oct 2003.

[17] M. Ghasemazar, E. Pakbaznia, and M. Pedram. Minimizing the power consumption of a Chip Multiprocessor under an average throughput constraint. In *Quality Electronic Design (ISQED), 2010 11th International Symposium on*, pages 362–371, March 2010.

[18] V. Hanumaiah and S. Vrudhula. Energy-Efficient Operation of Multicore Processors by DVFS, Task Migration, and Active Cooling. *IEEE Transactions on Computers*, 63(2):349–360, Feb 2014.

[19] A. Das, A. Kumar, and B. Veeravalli. Reliability and Energy-Aware Mapping and Scheduling of Multimedia Applications on Multiprocessor Systems. *IEEE Transactions on Parallel and Distributed Systems*, 27(3): 869–884, March 2016.

[20] G. Palermo, C. Silvano, and V. Zaccaria. ReSPIR: A Response Surface-Based Pareto Iterative Refinement for Application-Specific Design Space Exploration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(12):1816–1829, Dec 2009.

[21] S. H. Mozafari and B. H. Meyer. Characterizing the Effectiveness of Hot Sparing on Cost and Performance-per-Watt in Application Specific SIMT. *Submitted to Journal of Sustainable Computing: Informatics and Systems*, 2018.

[22] S. H. Mozafari and B. H. Meyer. Hot Sparing for Lifetime-Chip-Performance and Cost Improvement in Application Specific SIMT Processors. *Submitted to Integration, the VLSI*, 2018.

[23] Z. Yu, M. J. Meeuwsen, R. W. Apperson, O. Sattari, M. Lai, J. W. Webb, E. W. Work, D. Truong, T. Mohsenin, and B. M. Baas. AsAP: An Asynchronous Array of Simple Processors. *IEEE Journal of Solid-State Circuits*, 43(3):695–705, March 2008.

[24] B. Khailany, T. Williams, J. Lin, E. Long, M. Rygh, D. Tovey, and W. J. Dally. A Programmable 512 GOPS Stream Processor for Signal, Image, and Video Processing. In *2007 IEEE International Solid-State Circuits Conference. Digest of Technical Papers*, pages 272–602, Feb 2007.

[25] H. Kim, Y. Kim, J. Oh, and L. Kim. A Reconfigurable SIMT Processor for Mobile Ray Tracing With Contention Reduction in Shared Memory. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 60(4): 938–950, April 2013.

[26] J. Yoon, C. Yu, D. Kim, and L. Kim. A Dual-Shader 3-D Graphics Processor With Fast 4-D Vector Inner Product Units and Power-Aware Texture Cache. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 19(4):525–537, April 2011.

[27] Machine Learning on Amazon Web Service. `https://aws.amazon.com/machine-learning/`. Accessed: 2018-02-10.

[28] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang. A Survey on the Edge Computing for the Internet of Things. *IEEE Access*, 6:6900–6919, 2018.

[29] F. Hatori, T. Sakurai, K. Nogami, K. Sawada, M. Takahashi, M. Ichida, M. Uchida, I. Yoshii, Y. Kawahara, T. Hibi, Y. Saeki, H. Muroga, A. Tanaka, and K. Kanzaki. Introducing redundancy in field programmable gate arrays. In *Proceedings of IEEE Custom Integrated Circuits Conference - CICC '93*, pages 7.1.1–7.1.4, May 1993.

[30] V. V. Kumar and J. Lach. Heterogeneous redundancy for fault and defect tolerance with complexity independent area overhead. In *Proceedings 18th IEEE Symposium on Defect and Fault Tolerance in VLSI Systems*, pages 571–578, Nov 2003.

[31] Brett H. Meyer, Adam S. Hartman, and Donald E. Thomas. Cost-effective Lifetime and Yield Optimization for NoC-based MPSoCs. *ACM Trans. Des. Autom. Electron. Syst.*, 19(2), March 2014.

[32] S. Shamshiri and K. Cheng. Modeling Yield, Cost, and Quality of a Spare-Enhanced Multicore Chip. *IEEE Transactions on Computers*, 60 (9):1246–1259, Sept 2011.

[33] D. Cheng and S. K. Gupta. Maximizing Yield per Area of Highly Parallel CMPs Using Hardware Redundancy. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(10):1545–1558, Oct 2014.

[34] J. Srinivasan, S. V. Adve, Pradip Bose, and J. A. Rivers. Exploiting structural duplication for lifetime reliability enhancement. In *32nd International Symposium on Computer Architecture (ISCA'05)*, pages 520–531, June 2005.

[35] B. F. Romanescu and D. J. Sorin. Core Cannibalization Architecture: Improving lifetime chip performance for multicore processors in the presence of hard faults. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 43–51, Oct 2008.

[36] S. Gupta, S. Feng, A. Ansari, and S. Mahlke. StageNet: A Reconfigurable Fabric for Constructing Dependable CMPs. *IEEE Transactions on Computers*, 60(1):5–19, Jan 2011.

[37] H. P. Hofstee. Power efficient processor architecture and the cell processor. In *11th International Symposium on High-Performance Computer Architecture*, pages 258–262, Feb 2005.

[38] A. Ejlali, B. M. Al-Hashimi, and P. Eles. Low-Energy Standby-Sparing for Hard Real-Time Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(3):329–342, March 2012.

[39] Jingwen Leng, Tayler Hetherington, Ahmed ElTantawy, Syed Gilani, Nam Sung Kim, Tor M. Aamodt, and Vijay Janapa Reddi. GPUWattch: Enabling Energy Optimizations in GPGPUs. volume 41, pages 487–498, June 2013.

[40] H. Cook and K. Skadron. Predictive design space exploration using genetically programmed response surfaces. In *Design Automation Conference, 2008. DAC 2008.*, pages 960–965, June 2008.

[41] S. Gupta, S. Feng, A. Ansari, and S. Mahlke. StageNet: A Reconfigurable Fabric for Constructing Dependable CMPs. *IEEE Transactions on Computers*, 60(1):5–19, Jan 2011.

[42] R. Rodrigues, A. Annamalai, I. Koren, S. Kundu, and O. Khan. Performance Per Watt Benefits of Dynamic Core Morphing in Asymmetric Multicores. In *2011 International Conference on Parallel Architectures and Compilation Techniques*, pages 121–130, Oct 2011.

[43] J. Meng and K. Skadron. Avoiding cache thrashing due to private data placement in last-level cache for manycore scaling. In *2009 IEEE International Conference on Computer Design*, pages 282–288, Oct 2009.

[44] H. Cook and K. Skadron. Predictive design space exploration using genetically programmed response surfaces. In *45th ACM/IEEE Design Automation Conference*, pages 960–965, June 2008.

[45] T. Okabe, Y. Jin, and B. Sendhoff. A critical survey of performance indices for multi-objective optimisation. In *The 2003 Congress on Evolutionary Computation, 2003. CEC '03.*, volume 2, pages 878–885 Vol.2, Dec 2003.

[46] J. Mengte, A. Raghunathan, S. Chakradhar, and S. Byna. Exploiting the forgiving nature of applications for scalable parallel execution. In *2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, pages 1–12, April 2010.

[47] L. Zhang, Y. Han, Q. Xu, and X. Li. Defect Tolerance in Homogeneous Manycore Processors Using Core-Level Redundancy with Unified Topology. In *2008 Design, Automation and Test in Europe*, pages 891–896, March 2008.

[48] R. Narayanan, B. Ozisikyilmaz, J. Zambreno, G. Memik, and A. Choudhary. MineBench: A Benchmark Suite for Data Mining Workloads. In *2006 IEEE International Symposium on Workload Characterization*, pages 182–188, Oct 2006.

[49] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: characterization and methodological considerations. In *Proceedings 22nd Annual International Symposium on Computer Architecture*, pages 24–36, June 1995.

[50] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, and Kevin Skadron. A performance study of general-purpose applications on graphics processors using CUDA. *Journal of Parallel and Distributed Computing*, 68(10):1370 – 1380, 2008.

[51] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., 5 edition, 2012. ISBN 1558607242.

[52] S. Shamshiri, P. Lisherness, S. Pan, and K. Cheng. A Cost Analysis Framework for Multi-core Systems with Spares. In *2008 IEEE International Test Conference*, pages 1–8, Oct 2008.

[53] I. Parulkar, T. Ziaja, R. Pendurkar, A. D'Souza, and A. Majumdar. A scalable, low cost design-for-test architecture for UltraSPARC/spl trade/ chip multi-processors. In *Proceedings. International Test Conference*, pages 726–735, Oct 2002.

[54] Y. Li, S. Makar, and S. Mitra. CASP: Concurrent Autonomous Chip Self-Test Using Stored Test Patterns. In *2008 Design, Automation and Test in Europe*, pages 885–890, March 2008.

[55] J. A. Cunningham. The use and evaluation of yield models in integrated circuit manufacturing. *IEEE Transactions on Semiconductor Manufacturing*, 3(2):60–71, May 1990.

[56] De Sousa J. T. and Agrawal V. D. Reducing the complexity of defect level modeling using the clustering effect. In *Design, Automation and*

Test in Europe Conference and Exhibition 2000. Proceedings, pages 640–644, 2000.

[57] Fabian Fuerle and Johann Sienz. Formulation of the AudzeEglais uniform Latin hypercube design of experiments for constrained design spaces. *Advances in Engineering Software*, 42(9):680 – 689, 2011.

[58] Piotr Czyzak and Andrzej Jaszkiewicz. Pareto simulated annealing- A metaheuristic technique for multiple-objective combinatorial optimization. volume 7, pages 34 – 47, 01 1998.

[59] C. Silvano, W. Fornaciari, G. Palermo, V. Zaccaria, F. Castro, M. Martinez, S. Bocchio, R. Zafalon, P. Avasare, G. Vanmeerbeeck, C. Ykman-Couvreur, M. Wouters, C. Kavka, L. Onesti, A. Turco, U. Bondik, G. Mariani, H. Posadas, E. Villar, C. Wu, F. Dongrui, Z. Hao, and T. Shibin. MULTICUBE: Multi-objective Design Space Exploration of Multi-core Architectures. In *2010 IEEE Computer Society Annual Symposium on VLSI*, pages 488–493, July 2010.

[60] M. Mirza-Aghatabar, M. A. Breuer, S. K. Gupta, and S. Nazarian. Theory of redundancy for logic circuits to maximize yield/area. In *Thirteenth International Symposium on Quality Electronic Design (ISQED)*, pages 663–671, March 2012.

[61] K. Andryc, M. Merchant, and R. Tessier. FlexGrip: A soft GPGPU for FPGAs. In *2013 International Conference on Field-Programmable Technology (FPT)*, pages 230–237, Dec 2013.

[62] GeForce GTX-260 graphic card. `http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-260/specifications`, 2010. Accessed: 2015-04-25.

[63] CACTI Tools. `http://www.hpl.hp.com/research/cacti`, 2008. Accessed: 2015-04-20.

[64] G. Passas, M. Katevenis, and D. Pnevmatikatos. Crossbar NoCs Are Scalable Beyond 100 Nodes. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(4):573–585, April 2012.

[65] NVIDIA Tegra Series a GPU processor that is used in low-power embedded applications. `https://www.nvidia.com/object/tegra-superchip.html`, 2018. Accessed: 2018-08-18.

[66] GeForce GTX-480 graphic card and its die specifications. http://hexus.net/tech/reviews/graphics/24000-nvidias-geforce-gtx-480-finally-unleashed-reviewed-rated/?page=2. Accessed: 2015-10-16.

[67] Tensilica Vision P6 an embedded processor that is used in neural network models' inferencing. https://ip.cadence.com/news/564/330/Cadence-Announces-New-Tensilica-Vision-P6-DSP-Targeting-Embedded-Neural-Network-Applications, 2018. Accessed: 2018-08-18.

[68] 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 International Symposium on Microarchitecture (MICRO)*, pages 469–480, Dec 2009.

[69] L. M. Collins, J. J. Dziak, K. C. Kugler, and J. B. Trail. Factorial experiments: Efficient tools for evaluation of intervention components. *American Journal of Preventive Medicine*, pages 498–504, 2014.

[70] Lang Zhang, Hai Wang, and S. X. D. Tan. Fast stress analysis for runtime reliability enhancement of 3D IC using artificial neural network. In *2016 17th International Symposium on Quality Electronic Design (ISQED)*, pages 173–178, March 2016.

[71] Etienne Le Sueur and Gernot Heiser. Dynamic voltage and frequency scaling: The laws of diminishing returns. *2010 HotPower*, 10 2010.

[72] Ayse K. Coskun, Tajana Simunic Rosing, Kresimir Mihic, and Giovanni De Micheli. Analysis and Optimization of MPSoC Reliability. *Journal of Low Power Electronics*, 2(1):56–69, 2006.

[73] B. H. Meyer, A. S. Hartman, and D. E. Thomas. Cost-effective slack allocation for lifetime improvement in NoC-based MPSoCs. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1596–1601, March 2010.

[74] Joint Electron Device Engineering Council. Failure mechanisms and models for semiconductor devices. In *JEDEC Publication JEP122C*, 2006.

[75] L. Huang and Q. Xu. Energy-efficient task allocation and scheduling for multi-mode MPSoCs under lifetime reliability constraint. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1584–1589, March 2010.

[76] Z. Gu, C. Zhu, L. Shang, and R. P. Dick. Application-Specific MPSoC Reliability Optimization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 16(5):603–608, May 2008.

[77] Gregory G. Faust, Runjie Zhang, Kevin Skadron, Mircea Stan, and Brett H. Meyer. ArchFP: Rapid Prototyping of pre-RTL Floorplans. In *VLSI-SOC*, pages 259–263, 2012.

[78] Kevin Skadron, Mircea R. Stan, Karthik Sankaranarayanan, Wei Huang, Sivakumar Velusamy, and David Tarjan. Temperature-aware Microarchitecture: Modeling and Implementation. *ACM Trans. Archit. Code Optim.*, 1(1):94–125, March 2004.

[79] Intel Atom Processor D400 and D500 Series Thermal/Mechanical Specifications and Design Guidelines. Technical Report 322856-001, Intel, Dec. 2009.

[80] Dual-Core Intel Xeon Processor 3000 Series Thermal and Mechanical Design Guidelines. Technical Report 314917-001, Intel, Sept. 2006.