Hot Spare Components for Performance-Cost Improvement in Multi-core SIMT

S. Hasan Mozafari and Brett H. Meyer
Department Electrical and Computer Engineering
McGill University
Montréal, QC, Canada
seyyed.mozafari@mail.mcgill.ca, brett.meyer@mcgill.ca

***Abstract*—Adding redundant components is a well known technique for replacing defective components either before shipment or in the field, resulting yield improvement and consequently cost reduction. However, most yield improvement strategies utilize redundant components only when another component fails (i.e., cold spares). In this paper, we investigate the cost and performance implications of employing hot spares in multi-core single-instruction, multiple-thread (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. 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 *expected performance per cost* improved more than 2.5 and 1.7 times relative to systems integrating no redundancy and cold spares, respectively.**

I. Introduction

As manufacturing processes scale to smaller feature sizes, devices are more likely to experience early performance degradation, and even failure in the field [1]. Furthermore, manufacturing devices that operate according to their specification in the first place is also increasingly challenging: yield losses are mounting due to systematic and random defects [2], as well as parametric failure due to process variation [3].

Yield losses increase the cost of each die; to control such cost increases, ICs can be over-provisioned in several ways [4]:

- by increasing design margins (over-engineering),
- by implementing redundant circuitry (functional unit or microarchitectural redundancy), and
- by integrating spare cores (and other components).

Strict design margins (e.g., which require larger features that are more resilient to defect) can be used to improve every aspect of a system, at the cost of system performance [5]. Alternatively, micro-architectural redundancy can be added to improve yield in a targeted fashion. However, this requires steering logic (e.g., multiplexors) to connect redundant components when needed, which also results in performance loss [6]. Entire redundant cores can also be allocated to improve yield. At the first glance, core sparing seems to be a simple and effective way to not only address defects in the system, but also improve performance. Coarse-grained redundancy, however, is not cost-effective in multi-core systems with few cores, where a redundant core's relative area is considerable [7]; such techniques have seen adoption in larger systems [8], [9], [10].

Multi-core single-instruction, multiple-thread (SIMT) systems are especially well-suited to incorporating redundancy at multiple granularities, as there are multiple levels of design abstraction at which components are replicated for application performance improvement: (a) the system, which consists of multiple thread-parallel cores; and, (b) the cores, each of which consist of a single front-end unit and multiple, data-parallel processing elements (*lanes*). We have previously investigated cost trade-offs in application-specific SIMT architecture with cold spare components [11]. Such components are only activated when needed to replace a defective component. We observed that shared redundant microarchitectural elements reduce cost the most effectively, in this case.

In this paper, we extend our prior investigation by exploring the performance and cost impliciations of allocating hot spare components. These components improve yield by covering defects, when present, but can otherwise be used to improve performance over the baseline. Integrating such components presents additional challenges compared with cold spares: hot spare components must be allocated in such a way that the programming model is not disrupted. As before, we allocate spare cores, lanes, and shared-lanes, but this time enable these components for performance improvement when possible. To determine the relative effectiveness of these different applications of redundancy, we utilize them in the context of a case study system inspired by the NVIDIA GeForce GTX-260 GPGPU. 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. 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 aren't needed to replace defective components. By employing the framework presented in this paper, system architects can explore a wide range of performance-cost enhancing design alternatives in the early stages of the design process, and arrive at the configurations that will generate maximum performance per cost per wafer.

II. Related Work

Multi-core processors and SoCs are quickly becoming the dominant architecture as technology scaling improves the performance and reduces the fabrication cost, and the power of complex cores [12], [13], [14]. While individual core complexity is tapering off, however, the larger die size, increasing device counts, and higher defect rates in advanced manufacturing technologies are leading to lower yields [15]. Redundancy is commonly added to address this problem [2]; over-engineering is a well-known alternative [5].

Redundancy is a well-studied technique for improving yield and reliability, and has been previously employed at a variety of levels of design abstraction. Repetitive structures have been leveraged in memories, PLAs, and FPGAs [16], as well as processors [17]. Prior work has explored the allocation of cold spare microarchitectural units [7], [18], [19]. The cost of microarchitectural 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 [6]. 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 [6], [20], [11]. We expand upon this work by allocating hot-spare cores as well as hot-spare-, and hot-shared-spare-subcomponents.

Prior work has explored how to quantify performance in the context of defects [7], [18], [6]. 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 [19], [18], [21], [22]. 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.

III. Multi-Granularity Hot Sparing in SIMT

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 1, implementing different sparing regimes. Three different types of cores are illustrated: 1) main 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.

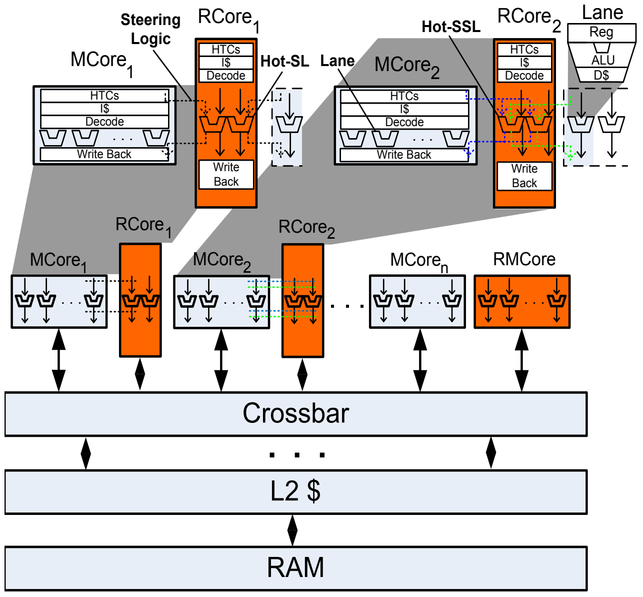

Fig. 1: Architecture of a multi-core SIMT system with hot spare components.

Each core (MCore, RCore, and RMCore) consists of a front-end unit, including an L1 instruction cache (I$), hardware thread contexts (HTC), and instruction decoder, processing elements (lanes), and a back-end, write-back unit. Each lane likewise consists of a arithmetic and logic unit (ALU), register file (Reg), and L1 data cache (D$). Cores communicate with L2 cache, at the system level, through a crossbar, while L2 cache is connected to RAM via a bus.

A. Redundancy Regimes

We propose allocating redundancy to salvage defective main cores (MCores) by scavenging for components in redundant cores (RCores) [22], by means of steering logic, or replacing them entirely with spare 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 1:

- *Spare lane (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$_1$.
- *Shared spare lane (SSL)* [11]: 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. 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.
- *Spare core (SC)*: an RMCore that can completely replace an MCore that cannot be otherwise salvaged.

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. It would be, however, possible to do this at runtime to increase system lifetime when components wear out; this is the subject of future work.

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

Performance and cost are influenced by several factors in multi-core SIMT with hot spares. 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 [23]. The addition of hot spares can improve performance when the spares are not needed to mitigate defects. We capture this effect by simulating the multi-core systems running a variety of benchmarks using the MV5 performance simulator [24]. 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.

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) [18], performance-averaged yield (PAY) [19], and $E[P]/Area$ [25], 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.

1) Cost Model: In $E[P]/C$, the cost C is the fabrication cost of a die [26]:

$$C_{die} = \frac{C_{wafer}/(Dice/Wafer)}{y_{sys}}, \tag{1}$$

where, 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}; the number of dice per wafer can be estimated as:

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

System yield, y_{sys} in Eq. (1), is defined as the ratio of working ICs to the total fabricated [2], and is a function of the component yield. In this paper, we adopt and extend the yield model we previously developed [11]. We calculate yield based on the type of redundancy that is utilized in the system (None, SC, SL, 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 performance-cost optimal solutions [11].

For multi-core SIMT, yield is

$$y_{sys} = y_{cores} \times y_{L2} \times y_{crossbar} \times (y_{RC_{skel}} \times y_{RC_{L1}})^p, \tag{3}$$

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:

$$y_{cores} = Binom(y_{core}, m, n) = \sum_{i=m}^{m+n} \binom{m+n}{i} (1 - y_{core})^{m+n-i} (y_{core})^i, \tag{4}$$

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:

$$y_{core} = y_{lanes} \times y_{L1} \times y_{skel}, \tag{5}$$

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. (5)), 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 [11]. The yield of each of these groups is:

$$y_{3core} = Binom(y_{lane}, 2k, l) \times y_{L1}^2 \times y_{skel}^2. \tag{6}$$

Note that the yield of each RCore's $L1$ and $skel$ are accounted for in the system yield formula Eq. (3).

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

$$E[P] = \sum_{C_i \in \{S\}} Perf(C_i) \times Prob(C_i), \tag{7}$$

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 billion 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 excluded 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.

3) Estimating $E[P]$: To control the computational cost of design evaluation, we estimate $E[P]$ with $\hat{E}[P] \leq E[P]$ 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,

$$\hat{C_i} = \begin{cases} C_i & \sum_{j=1}^{i} Prob(C_j) \leq Th \\ Null & otherwise \end{cases} \tag{8}$$

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

$$\hat{E}[P] = \sum_{\hat{C}_i \in \{\hat{S}\}} Perf(\hat{C}_i) \times Prob(\hat{C_i}), \tag{9}$$

when $Perf(Null) = 0$.

We observe that $\hat{E}[P] \leq E[P]$ and that as $Th \to 1$, $\hat{E}[P] \to E[P]$. 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 future work.

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 SL and SSL), and where one core in the system is defective (when considering SC). These cases constitute more than 94% of the dice population for our example system whether considering SL, SSL, or SC, reduce the required simulation by 90%. This significantly reduces the number of systems for which we must calculate $Prob$.

4) 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

$$p_{C_i} = p_{cores} \times p_{crossbar} \times p_{L2\$} \tag{10}$$

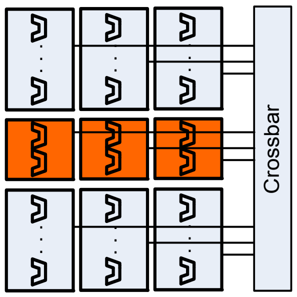

Fig. 2: Case study system.

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 IV-B); $p_{L2\$}$ 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

$$\begin{aligned} p_{cores} = \binom{n_{mc}}{n_{dmc}} \times p_{core'}^{n_{dmc}}(n_{ml}, n_{dml}) \times \\ p_{core}^{(n_{mc}-n_{dmc})}(n_{ml}) \times \binom{n_{rc}}{n_{drc}} \times p_{core'}^{n_{drc}}(n_{rl}, n_{drl}) \\ \times p_{core}^{(n_{rc}-n_{drc})}(n_{rl}), \end{aligned} \tag{11}$$

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

$$p_{core}(n_l) = y_{lane}^{n_l} \times y_{skel} \times y_{L1\$}, \tag{12}$$

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

$$p_{core'}(n_l, n_{dl}) = \binom{n_l}{n_{dl}} \times y_{lane}^{(n_l-n_{dl})} \times (1 - y_{lane})^{n_{dl}} \times y_{skel}. \tag{13}$$

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

IV. 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 GForce 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) [27];

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 2). We have previously observed that many redundant lanes are not helpful in terms of cost reduction in the context of SIMT processors [11], we did not consider more than two lanes per each RCore.

A. SIMT Configurations and Benchmarks

We used MV5 [24] to simulate the performance of each SIMT configuration we considered. We assume a number of architectural parameters consistent with the literature [28], 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 selected benchmarks from some suites: Minebench [29], SPLASH-2 [30], and Rodinia [31]. 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), has been previously used in the literature to evaluate multi-core SIMT performance [24].

B. Cost Estimation

We assume the cost of \$3000, radius $150mm$, and yield of 1 for each wafer, and adopt the negative binomial yield model to calculate the yield of individual components. This model has three parameters: defect density (λ_b), the clustering parameter (α), and block area (A_b) [32], [33]:

$$y_b = \left(1 + \frac{\lambda_b \times A_b}{\alpha}\right)^{-\alpha}. \tag{14}$$

In $65nm$ manufacturing technology, we assume $\lambda_b = 0.025/cm^2$ and $\alpha = 4$ [26]. Moreover, we consider the yield values of D\$ and crossbar 1, since inexpensive redundancy can increase yield dramatically [34].

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 [23]. FlexGrip is a configurable GPUPU based on the NVIDIA G80 architecture targeting FPGA. 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 [35] memory models to estimate their areas. The area measurement of different sub-components of the FlexGrip GPGPU is reported in Table I.

TABLE I: 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

C. Performance Degradation with Hot Redundancy

Adding redundancy often increases critical path delay and consequently ultimately reduces system performance. 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 [36].

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 for the corresponding core. Memory access latency can, in some cases, hide the reduction in clock frequency. Calculating this performance degradation considering delays in memory interactions is the subject of future work.

V. Results

A. 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 IV-B.

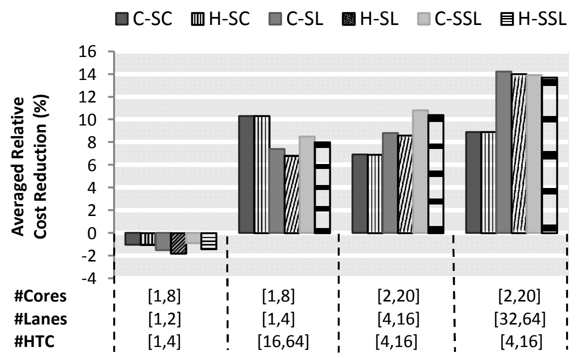

Fig. 3: Comparison of cold- and hot-sparing under different sets of configurations and redundancy techniques.

In Figure 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 (C-) or hot (H-) spare cores, we observe that there is no difference 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 C-SL or C-SSL, their ARCR are at most 1.5% better than H-SL and H-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.

B. E[P]/C Improvement of Hot Sparing

In Figure 4, we illustrate the normalized *expected performance per cost ($E[P]/C$)* for different types of redundancy (bar graphs), normalized to the 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 [29], [30]: 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 [31], [29]: 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%.

VI. 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.

Acknowledgements

This research was made possible with funding from Fonds de recherche Nature et technologies du Québec (FRQNT), and CAD tools from CMC Microsystems.

References

[1] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," *IEEE MICRO*, vol. 25, no. 6, 2005.

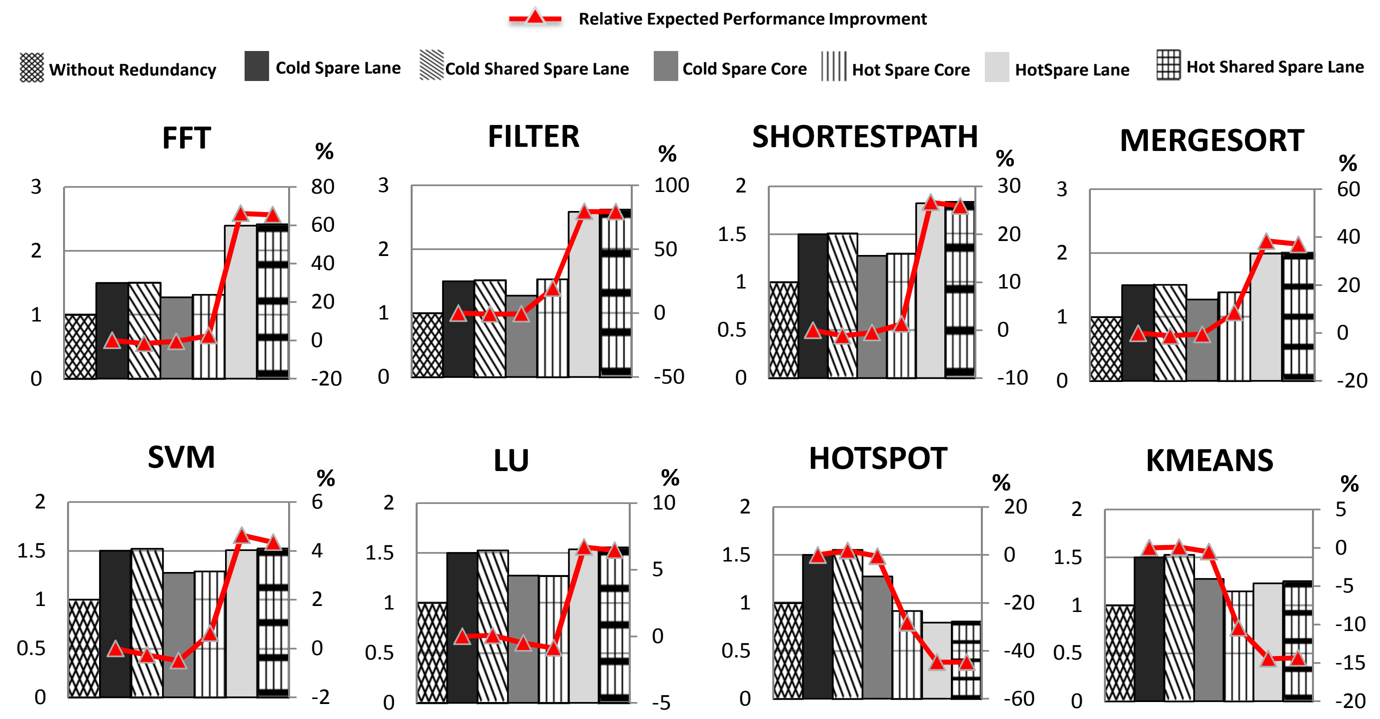

Fig. 4: Normalized $E[P]/Cost$ (bars, left) and relative $E[P]$ improvement (lines, right) and over different benchmarks.

[2] I. Koren and Z. Koren, "Defect tolerance in VLSI circuits: Techniques and yield analysis," *Proceedings of the IEEE*, vol. 86, no. 9, 1998.

[3] E. Humenay, *et al.*, "Impact of process variations on multicore performance symmetry," in *DATE*, 2007.

[4] Y. Markovsky and J. Wawrzynek, "On the opportunity to improve system yield with multi-core architectures," in *DFM&Y*, 2007.

[5] M. Mirza-Aghatabar *et al.*, "Theory of redundancy for logic circuits to maximize yield/area," in *ISQED*, 2012.

[6] Y. Gao *et al.*, "Trading off area, yield and performance via hybrid redundancy in multi-core architectures," in *VTS*, 2013.

[7] P. Shivakumar *et al.*, "Exploiting microarchitectural redundancy for defect tolerance," in *Computer Design*, 2003.

[8] J. A. Kahle, *et al.*, "Introduction to the Cell multiprocessor," *IBM Journal of Research and Development*, vol. 49, 2005.

[9] R. Stefan, *et al.*, "A 45 nm 8-core enterprise Xeon processor," *JSSC*, vol. 45, no. 1, 2010.

[10] A. Durytskyy *et al.*, "Improving GPU robustness by making use of faulty parts," in *ICCD*, 2011.

[11] S. H. Mozafari, *et al.*, "Yield-aware performance-cost characterization for multi-core SIMT," in *GLSVLSI*, 2015.

[12] M. Gschwind, *et al.*, "Synergistic processing in Cell's multicore architecture," *Micro, IEEE*, vol. 26, no. 2, pp. 10–24, March 2006.

[13] S. Vangal, *et al.*, "An 80-tile 1.28 TFLOPS Network-on-Chip in 65nm CMOS," in *ISSCC*, 2007.

[14] D. Pham, *et al.*, "The design and implementation of a first-generation Cell processor," in *ISSCC*, 2005.

[15] "International technology roadmap for semiconductors," http://www.itrs.net/Links/2013ITRS/Home2013.htm, accessed: 2015-01-30.

[16] F. Hatori, *et al.*, "Introducing redundancy in field programmable gate arrays," in *CICC*, 1993.

[17] S. Makar, *et al.*, "Testing of Vega2, a chip multi-processor with spare processors." in *ITC*, 2007.

[18] E. Schuchman and T. N. Vijaykumar, "Rescue: a microarchitecture for testability and defect tolerance," in *ISCA*, 2005.

[19] J. Srinivasan, *et al.*, "Exploiting structural duplication for lifetime reliability enhancement," in *ISCA*, 2005.

[20] D. Cheng and S. Gupta, "Maximizing yield per area of highly parallel CMPs using hardware redundancy," *TCAD*, vol. 33, no. 10, 2014.

[21] B. F. Romanescu and D. J. Sorin, "Core cannibalization architecture: Improving lifetime chip performance for multicore processors in the presence of hard faults," in *PACT*, 2008.

[22] S. Gupta, *et al.*, "StageNet: Reconfigurable fabric for constructing dependable CMPs," *IEEE Transaction of Computer.*, vol. 60, no. 1, 2011.

[23] K. Andryc, *et al.*, "FlexGrip: A soft GPGPU for FPGAs," in *FPT*, 2013.

[24] J. Meng and K. Skadron, "Avoiding cache thrashing due to private data placement in last-level cache for many-core scaling," in *ICCD*, 2007.

[25] Y. Gao *et al.*, "A new paradigm for trading off yield, area and performance to enhance performance per wafer," in *DATE*, 2013.

[26] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 5th ed. Morgan Kaufmann Publishers Inc., 2012.

[27] "GForce GTX-260 graphic card," http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-260/specifications, accessed: 2015-04-25.

[28] J. Meng, *et al.*, "Exploiting the forgiving nature of applications for scalable parallel execution," in *IPDPS*, 2010.

[29] R. Narayanan, *et al.*, "Minebench: A benchmark suite for data mining workloads," in *IISWC*, 2006.

[30] S. C. Woo, *et al.*, "The SPLASH-2 programs: Characterization and methodological considerations," in *ISCA*, 1995.

[31] S. Che, *et al.*, "A performance study of general purpose applications on graphics processors using CUDA," *JPDC, Elsevier*, vol. 68, no. 10, 2008.

[32] J. Cunningham, "The use and evaluation of yield models in integrated circuit manufacturing," *TSM*, vol. 3, no. 2, 1990.

[33] J. De Sousa and V. Agrawal, "Reducing the complexity of defect level modeling using the clustering effect," in *DATE*, 2000.

[34] S. Shamshiri and K.-T. Cheng, "Modeling yield, cost, and quality of a spare-enhanced multicore chip," *IEEE Tran. Comp.*, vol. 60, no. 9, 2011.

[35] "CACTI tools," www.hpl.hp.com/research/cacti, accessed: 2015-04-20.

[36] G. Passas, *et al.*, "Crossbar NoCs are scalable beyond 100 nodes," *TCAD*, vol. 31, no. 4, 2012.