Yield-aware Performance-Cost Characterization for Multi-Core SIMT

Seyyed Hasan Mozafari†, Kevin Skadron‡, and Brett H. Meyer†

†Dept. of Electrical and Computer Engineering
McGill University
Montréal, QC, Canada

‡Computer Science Department
University of Virginia
Charlottesville, VA, USA

seyyed.mozafari@mail.mcgill.ca, skadron@cs.virginia.edu, brett.meyer@mcgill.ca

ABSTRACT

Redundancy is now routinely allocated in circuits, microarchitectural structures, or at the system level, to mitigate mounting manufacturing yield losses. In this paper, we explore the effect of core-, lane-, and, a new redundancy technique, shared-lane-sparing on application-specific SIMT cost. The structure of multi-core SIMT systems makes them particularly suitable for applying redundancy at multiple levels of granularity. Just as the performance of thread- or data-parallel applications can be improved with greater core or lane count, respectively, the yield of thread- or data-parallel systems can be improved with core- or lane-sparing. At times, however, applications and systems fall into a third category: they benefit from one or the other category, but not exclusively. In this case, we propose *spare lane sharing*, which reduces the cost of such systems by allowing one of two neighboring cores to make use a redundant lane if necessary. We have evaluated core-, lane-, and shared-lane-sparing applied to multi-core SIMT systems executing 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-lane-sparing outperforms lane-sparing, reducing cost by up to 20%.

1. 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 can operate according to their specification is also increasingly challenging: yield losses are mounting, due to systematic and random defects [2], as well as parametric failure resulting from process variation [3]. To control costs, silicon systems must now be over-provisioned at the circuit level (*e.g.*, guard-bands and noise margins in design rules), microarchitectural level (*e.g.*, using redundant execution units), or system level (*e.g.*, using redundant cores) to ensure that systems with some defects can satisfy specifications and can, therefore, be sold.

We hypothesize that, given different performance and cost targets, different redundancy strategies emerge as most cost-effective. Yield can be improved in a number of 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).

It is clear from industrial adoption that disabling defective cores improves yield by salvaging otherwise defective dice [5][6]; the recovered dice occupy a different (lower cost) market segment, but the fact that they can be sold at all reduces the manufacturing cost of the defect-free dice. However, a question remains: under what circumstances does such coarse-grained redundancy *optimize* yield?

In this paper, we investigate the application of redundancy to improve the yield of multi-core single-instruction, multiple-thread (SIMT) architectures [7]. Application -specific SIMT architectures [8][9] present a unique opportunity for yield improvement, as there are multiple levels of design abstraction at which components are already 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*). Because 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. 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.

We investigate the opportunity to reduce manufacturing costs in the presence of random defects by allocating spare cores, lanes, and shared lanes, in SIMT architecture. We observe, when considering the space of performance-cost trade-offs, that while some design points (low-cost designs, in particular) call for no redundancy, and others for core- or lane-sparing, many benefit the most from shared spare lanes. Our experiments reveal that an SSL can be integrated with negligible area overhead, and only 1.3% timing overhead, while reducing costs significantly, especially for applications that call for systems that mix thread- and data-parallel execution. Our contributions are (1) the introduction and evaluation of *shared spare lanes*, including the development of yield models; and, (2) a characterization of core-, lane-, and shared-lane-sparing in the context of performance-cost optimal, application-specific multi-core SIMT processors.

2. RELATED WORK

Application-specific and reconfigurable single-instruction, multiple-data (SIMD) processors are emerging as a low-cost alternative to general-purpose multi-core systems. For example, Lyuh et al., [10] proposed a reconfigurable processor

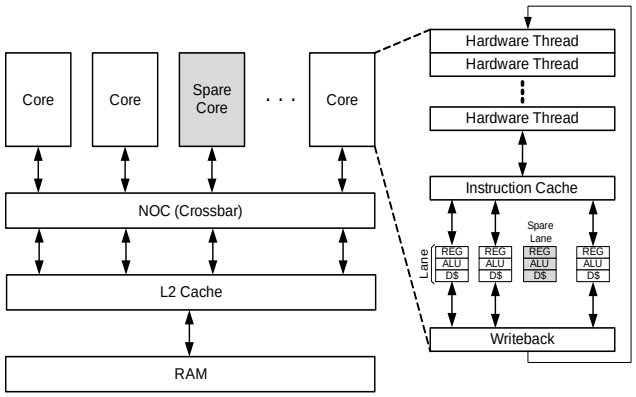


Figure 1: Architecture of SIMT multi-core processors.

using dynamically partitioned SIMD processor for multimedia applications. Shohei et al., [11] proposed a new processor architecture, XC, which can dynamically switch between SIMD and MIMD to get better performance for image processing applications. In this paper, we investigate strategies for improving the yield of application-specific SIMT processors; the application of such techniques to reconfigurable processors is the subject of future work.

Redundancy is a well-studied technique for improving yield and has been previously employed at a variety of levels of design abstraction, e.g.: at the circuit level, by taking advantage of repetitive structures [2][12], or designing multi-purpose functional units [13]; and, at microarchitecture and system level, by allocating spare cores and functional units [14][15]. To best of our knowledge, our work is the first to investigate the effect of shared redundant resources on manufacturing cost reduction for SIMT architectures.

As a consequence of the growing interest in system-level redundancy, a variety of authors have proposed modeling or analysis frameworks for evaluating redundancy strategies in the multi-core era. Several studies have come to the conclusion that as cores proliferate, core-level redundancy is the most cost-effective [14][15][4]: lower-level techniques incur too great an overhead to cover too few possible defects. Shamshiri et al. [16] further 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; we consequently observe that systems that take advantage of data-level parallelism require alternatives to core sparing to achieve cost-effective yield improvement.

3. MULTI-GRANULARITY REDUNDANCY

A multi-core SIMT processor is depicted in Figure 1 [7]. Each SIMT 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. Each core consists of a unified front-end unit, including L1 instruction cache (I$), hardware thread contexts, and instruction decoder, processing elements (lanes), and a back-end write-back unit. Each lane consists of a register file, arithmetic and logic unit (ALU), and L1 data cache (D$). At the system level, cores communicate with L2 caches through a crossbar.

SIMT and array-based architectures differ from vector single-instruction, multiple-data (SIMD) architectures in that vector SIMD architectures expose the SIMD width to the instruction set. As a result, software must operate at the granularity of the vector width, which is often unnatural for both programmers and compilers. SIMT and array-based architectures, on the other hand, present a scalar architecture to each thread, simplifying programming and compilation and allowing the hardware to automatically handle data-parallel execution divergence. SIMT and array-based architectures differ in that array-based PEs execute in lock-step; SIMT adds another level of hierarchy, allowing groups of lockstep lanes to execute in parallel. In this paper, we refer to these collections of lockstep lanes as cores.

Multi-core SIMT systems simultaneously take advantage of thread- and data-level parallelism (TLP and DLP respectively): cores can execute different threads of the same application, or even different applications, and each core can execute the same instructions on multiple data elements at the same time. Different parallel applications have different working set sizes and mixes of TLP and DLP; therefore, different configurations result in optimal performance [17]. 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 granularities both relatively easy and particularly beneficial.

3.1 System Yield

Yield is defined, at the time of manufacturing, as the ratio of working ICs to the total number fabricated [2]. An IC may fail at manufacturing time for many reasons: random defects, such as open- or short-circuits due to particle contamination; systematic defects due to an immature design or fabrication process; or, process variations, which, amongst other things, may affect IC timing.

Redundancy improves yield by substituting an operational component for a defective component at manufacturing time, increasing the salable chip count. However, if no defects are present, we assume that the redundant component is unused. Die cost, therefore, is only reduced if the increase in salable dice overwhelms the increase is system area.

Based on the system architecture in Figure 1, the system yield y_{sys} is the product of the yield of different groups of components, namely the cores, y_{cores}, the L2 caches, y_{L2}, and the crossbar, $y_{crossbar}$:

$$y_{sys} = y_{cores} \times y_{L2} \times y_{crossbar}. \quad (1)$$

Row and column redundancy is standard in SRAM arrays; we assume that the yield of L1 and L2 caches is 1. Furthermore, crossbar yield approaches 1 with the addition of redundant lines [16]. In this case, the yield of the system is equivalent to the yield of the set of cores.

3.2 Core Sparing

One straightforward way to improve system yield in homogeneous multiprocessors is to add spare cores. When a core is defective, a redundant core can be substituted for it. In the context of a crossbar-based system, the only performance penalty is due to the increased size of the crossbar.

As the number of cores in the system increases, the relative overhead of a single redundant core goes down. Systems with many narrow cores (cores with few lanes) therefore 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 the binomial distribution:

$$y_{cores} = \sum_{i=m}^{m+n} \binom{m+n}{i} (1 - y_{core})^{m+n-i} (y_{core})^{i}. \quad (2)$$

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 components have yield y.

3.3 Lane Sparing

The yield of a core is dependent on the yield of its components. In this paper, we divide a core into (a) its lanes, (b) its L1 cache ($y_{L1} = 1$), and (c) everything else (skeleton):

$$y_{core} = y_{lanes} \times y_{L1} \times y_{skeleton}. \quad (3)$$

Just as system-level homogeneity makes core sparing effective, core-level homogeneity makes lane sparing effective. A spare lane can be integrated in a core as depicted in Figure 2. A core without lane sparing is illustrated in white. When spare lane S (gray) is integrated, if one of the four main lanes is defective, the core may continue to function.

We observe from the die photo of AMD Opteron 130nm processor that the elements of a single lane occupy more than 72% of a the core's area. Our assumption is that the remaining area is not easily protected at low overhead: while some units, such as the hardware thread contexts, are repetitive, the structures are relatively small. Redundancy therefore offers little coverage relative to the total area of a core [14].

This is especially true in cores with more than one lane: with each additional lane, the proportion of the core in the skeleton shrinks. Consequently, we do not protect them by redundancy within a given core; instead, failure within these elements can be addressed with core sparing.

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: $y_{lanes} = Binom(y_{lane}, k, l)$.

3.4 Spare Lane Sharing

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, once again illustrated in Figure 2. When the two cores share a spare lane, if either the white core or the orange core have a defective lane, Lane S and

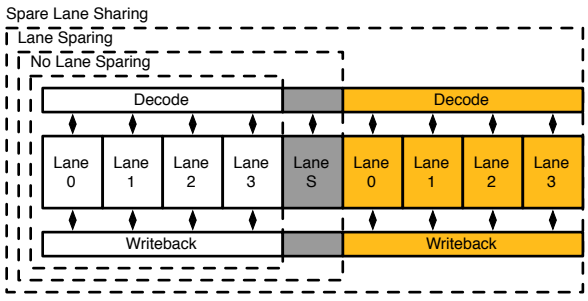


Figure 2: The physical design of SIMT architecture allows two cores to potentially share a single redundant lane.

the corresponding decode and write-back logic can be employed. The shared spare lane improves yield and reduces costs when spare lanes may otherwise be too costly; from a redundancy perspective, an SSL covers twice the lanes (at half the overhead) compared with a dedicated spare lane.

3.4.1 SSL Overhead

Integrating a spare shared lane only requires the addition of multiplexors in the decode and write-back units to direct signals to and from the spare lane. The performance penalty of accessing a redundant resource (e.g., pipeline stage) in general purpose multi-cores is high [18]. However, due to hierarchical design of SIMTs, a shared spare lane can be included in either core with only a small performance loss. To quantify the area and performance overhead of an SSL, we added one between two independent tiles of the FlexGrip GPGPU streaming processor [19]. FlexGrip is very similar in structure to the SIMT architecture in Figure 1. We synthesized the processor using the 65nm TSMC standard library and measured the change in area and clock frequency. We observe that the area overhead (not including the spare lane itself) is less than 0.1% of a single lane; clock frequency falls just 1.3%. Application execution latency is expected to degrade less than this, as the change in clock frequency will be absorbed in part by memory access delay. Quantifying this in multiprocessor simulation is the subject of future work.

3.4.2 SSL Yield

The inclusion of an SSL makes the yield of one core in the pair dependent upon the yield of the other: if one core needs the spare shared lane, it is only available if the other core does not. Consequently, there is no straightforward formula to calculate a single core's yield when both SSL and spare cores are present. Furthermore, the combinatorial space is prohibitively large; simply counting systems that yield or do not is computationally intractable as systems grow. We therefore estimate yield as follows. There are two cases in which both cores that share a spare lane are functional:

1. Each cores' lanes yield ($y_{lanes\_ssl_1}$);
2. One core needs the shared spare lane ($2 \cdot y_{lanes\_ssl_2}$).

When neither core needs the shared spare lane (when sufficient spare lanes are present or no lanes are defective), the yield of the set of lanes (given k lanes and l spares) is:

$$y_{lanes\_ssl_1} = Binom(y_{lane}, k, l) \times Binom(y_{lane}, k, l). \quad (4)$$

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 is both functional (not defective itself) and not needed by the other core. In this case, the yield is:

$$y_{lanes\_ssl_2} = y_{lane} \times \binom{k+l}{l+1} (1 - y_{lane})^{l+1} (y_{lane})^{k-1} \times Binom(y_{lane}, k, l). \quad (5)$$

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,

$$y_{cores} = Binom(y_{core_{ssl}}, m, n)(1 - y_{core}) + \sum_{i=m}^{m+n+1} \binom{m+n}{i-1} (1-y_{core_{ssl}})^{m+n-(i-1)} (y_{core_{ssl}})^{(i-1)} (y_{core}). \quad (6)$$

The effect of the introduced error is, in the end, negligible: we observe that, for the defect densities considered, systems that employ more than one type of redundancy (e.g., spare cores and spare shared lanes) are too expensive to fall on the performance-cost Pareto-optimal front. We hypothesize that this may change under higher defect densities; such an investigation is the subject of future work.

4. EXPERIMENTAL SETUP

To determine what form of redundancy is most beneficial given a particular (a) configuration and (b) application, we evaluated the performance, area, and yield of a wide variety of multi-core SIMT systems executing a diverse benchmarks.

4.1 SIMT Configurations and Benchmarks

We used MV5 [20] to simulate the performance of SIMT configurations, consistent with parameters in the literature [7], including: 1 GHz processor clock frequency, 16 KB instruction and 4 MB L2 cache per core, 300 MHz crossbar, and 300 cycle memory access latency.

We varied the number of cores N, $N \in \{1, 2, 4, 6, \ldots, 20\}$, 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, $S \in \{8, 16, 32, 64\}$ 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 [21], SPLASH-2 [22], and Rodinia [23]. FFT, Filter, HotSpot, LU, MergeSort, ShortestPath, KMeans, SVM have been previously used to evaluate multi-core SIMT performance [7].

4.2 Cost Estimation

The cost of a die C_{die} is a function of the cost of a wafer C_{wafer}, the number of dice per wafer, and die yield, y_{sys} [24]:

$$C_{die} = \frac{C_{wafer}/(Dice/Wafer)}{y_{sys}}. \quad (7)$$

The number of dice per wafer can be estimated as:

$$Dice/Wafer = \frac{\pi \times (Radius_{wafer})^2}{Area_{die}} - \frac{\pi \times Radius_{wafer}}{\sqrt[2]{\frac{Area_{die}}{2}}}. \quad (8)$$

We assume a wafer radius of $150mm$, a wafer yield of 1, and that wafers cost \$3000 each. y_{sys} is calculated as in Section 3. The yield of individual components, such as lanes, is calculated with the negative binomial yield model [25, 26]:

$$y_b = \left(1 + \frac{\lambda_b \times A_b}{\alpha}\right)^{-\alpha}. \quad (9)$$

where (α), (λ_b), and (A_b) are the clustering parameter, defect density, and component area, respectively. Smaller α results in stronger defect clustering and vice versa, a function of the process technology. We assume $\alpha = 4$ and $\lambda_b = 0.025/cm^2$ in $65nm$ manufacturing technology [24].

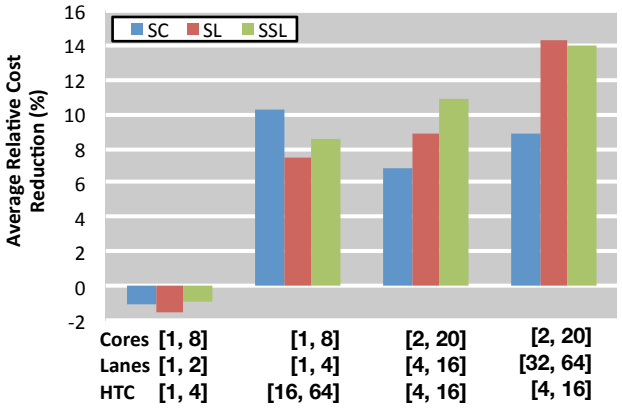


Figure 3: Different sets of configurations benefit the most from different types of redundancy.

To estimate the chip area we measured the area of functional units based on die photo of an AMD Opteron processor. As this processor is fabricated in $130nm$, we used a scaling factor of 0.7 per generation to scale the processor to $65nm$. While the Opteron and SIMT processors are not architecturally the same, the relative size of the functional units is not expected to vary significantly. Moreover, since the core's area is dominated by the integer ALU and multiplier, the relative size of other components is less important.

5. RESULTS

We conducted a variety of experiments to validate our models and evaluate the different redundancy techniques.

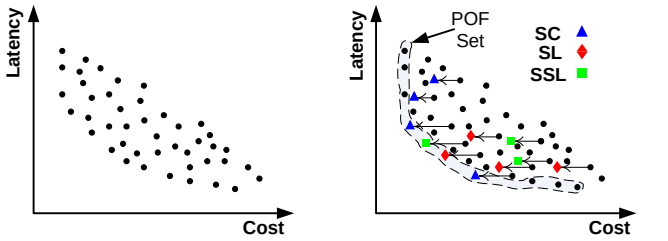
5.1 Model Validation

To validate our yield models (Section 3), we compared our analytical results with statistical simulation using Monte Carlo Simulation (MCS). We considered 140 large configurations (each with more than 50 cores, and at least 16 lanes per each core), divided into seven equal subsets. Each of the first three subsets are allocated a single type of redundancy, a spare core (SC), spare lane (SL), or shared spare lane (SSL). Each of the second three are allocated a pair of redundancy types (SC+SL, SC+SSL, or SL+SSL). The last subset is allocated all three types. As appropriate, we consider n SC, k SL, and zero or one 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. The number of HTC in a configuration is $k + l + s$, where $s = 1$ if an SSL has been integrated.

We simulated random defects, using an unrealistically high defect density ($0.3/cm^2$) to exaggerate any potential error in the models. In each MCS trial, component yield was calculated as described in Section 4.2, and component failure determined by sampling the uniform distribution $\mathcal{U}(0, 1)$. When $X \in \mathcal{U}(0, 1) > y_c$, c is defective. If no redundancy has been allocated to cover this defect, the system in this trial is marked as defective. We performed 100K trials for each configuration. On average over all subsets, the relative difference in yield between MCS and our models is 0.61%, with a 95% confidence interval of 0.76% yield.

5.2 Cost-effectiveness of Redundancy Regimes

We next investigated the general effectiveness of each re-



(a) Baseline systems. (b) Systems with redundancy.

Figure 4: Different types of redundancy is added to configurations to evaluate application-specific cost reduction.

dundancy technique across a wide variety of configurations. For each of 283 configurations (the full design space in Section 4.1, but with D$ held constant—D$ size does not affect yield, only performance), we allocated a single type of redundancy (combinations of redundancy do not appear on the Pareto-optimal front for our selected defect density). As in Section 5.1, we consider n SC, k SL, and zero or one SSL. An SSL is only allocated when there is more than one core in the system. We then calculated the *average relative cost reduction* (ARCR) of defect-tolerant configuration. ARCR is defined as the change in cost relative to the baseline design with no redundancy. Cost is calculated as in Eq. (7).

In Figure 3, we divide 202 configurations, 70% of those evaluated, 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. Designs are omitted to simplify the categories for the purpose of illustrating the differences in cost reduction as a function of configuration types.

The configurations 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 configurations for each type of redundancy.

We observe that when systems are composed of small processors, no redundancy technique reduces manufacturing cost—e.g., lane sparing leads to 1.5% relative cost increase on average. As processors and systems grow, the best redundancy regime varies. When there are few large cores (e.g., more than eight HTC), core sparing reduces the cost nearly 2% and 3% more than SSL and SL, respectively: recall that HTCs cannot be explicitly protected with redundancy; only spare cores can cover failures in HTCs. However, when the number of lanes and HTCs per each core are restricted to less than 16, spare lane sharing is best: the area overhead is amortized across two cores. In this group, the ARCR is 10.8%, 8.8%, and 6.9% for SSL, SL, and SC, respectively. When cores utilize many lanes, lane sparing is the best: SSL is not as effective in the presence of many lanes. In this case, SL reduces ARCR by 14.2%, 0.5% better than SSL.

5.3 General Design Space Characterization

Since performance is an important objective in SIMT processor design, our final experiment (a) identified the set of performance-cost Pareto-optimal designs for each benchmark, and (b) added redundancy to each, to determine which type of redundancy best reduces the cost of application-specific configurations. We performed exhaustive performance simulation using MV5, considering 813 configurations

Table 1: ARCR for Performance-cost POF Configurations

App.	Configurations				ARCR (%)		
	Cores	D$ Size	HTC	Lanes	SC	SL	SSL
Filter	[2, 8]	[8, 32]	[16, 64]	[2, 4]	**11.2**	7.9	9.6
	[4, 16]	[16, 32]	[8, 16]	[16, 32]	7.5	**8.3**	7.2
	[1, 2]	[8, 32]	[2, 8]	[1, 4]	-0.9	-1.3	-0.7
KM*	[6, 8]	[8, 32]	[8, 16]	[1, 2]	**7.3**	4.3	5.8
	[4, 16]	[8, 16]	[2, 4]	[4, 16]	5.3	7.1	**8.3**
	[1, 2]	[16, 32]	[1, 4]	[1, 4]	-1.7	-1.8	-0.9
FFT	[8, 16]	[8, 64]	[8, 16]	[1, 4]	**4.5**	3.1	4.0
	[1, 4]	[8, 16]	[1, 8]	[1, 2]	-0.9	-1.6	-1.2
MS+	[16, 20]	[32, 64]	[8, 16]	[1, 4]	**5.6**	4.3	4.7
	[4, 8]	[8, 32]	[4, 8]	[4, 8]	3.6	3.9	**4.5**
	[1, 2]	[8, 16]	[4, 16]	[1, 4]	-0.6	-1.2	-0.8
SP†	[16]	[16, 64]	[8, 32]	[2, 4]	**7.6**	6.1	6.8
	[8]	[16]	[4, 8]	[4, 8]	7.2	7.1	**8.7**
	[1, 2]	[8, 16]	[1, 4]	[1, 4]	-1.1	-1.4	-0.7
HS‡	[16, 20]	[32, 64]	[8, 16]	[1, 4]	**5.5**	4.2	4.8
	[4, 16]	[8, 32]	[4, 16]	[4, 8]	4.7	5.3	**6.0**
	[1, 2]	[16, 32]	[1, 32]	[1, 2]	-1.3	-1.1	-0.5
LU	[16, 20]	[16, 64]	[16, 32]	[1, 4]	**6.2**	4.7	5.6
	[2, 4]	[16, 32]	[4, 8]	[4, 8]	3.5	4.1	**4.5**
	[1, 2]	[8, 16]	[4, 16]	[1, 2]	-0.7	-1.7	-0.9
SVM	[8]	[8]	[8]	[2]	**1.3**	-0.8	-0.5
	[1, 2]	[8, 64]	[1, 32]	[1, 2]	-1.1	-1.0	-0.7

*: KMeans, +: MergeSort, †: ShortestPath, ‡: HotSpot

Table 2: Redundancy type composition of POF

App.	Type of Redundancy (%)			
	SC	SL	SSL	None
Filter	24	19	0	57
KM	28	0	31	41
FFT	27	0	0	72
MS	22	0	8	70
SP	26	0	6	68
HS	21	0	30	49
LU	20	0	19	61
SVM	5	0	0	95

with no redundancy for each benchmark (Figure 4(a)). We then added redundancy to each configuration, as in Section 5.2, to the point that its manufacturing cost is greater than the cost of the baseline system. Designs utilizing SSL pay a 1.3% execution latency penalty (see Section 3.4.1). From the set of baseline and corresponding defect-tolerant designs, we selected the Pareto-optimal front (POF) (Figure 4(b)). We observe that systems allocated a combination of redundancy types, for our defect density, do not appear on the POF; neither do systems with more than a single redundant unit of a given type.

The results of our experiments are summarized in Tables 1 and 2. Table 1 lists, by benchmark (App.), the range of configurations that benefit most, in terms of ARCR, from each redundancy allocation under consideration: a spare core (SC), spare lane (SL), shared spare lane (SSL), and no redundancy (None). The best ARCR for each configuration group is indicated in bold; when no redundancy type decreases cost, no ARCR is bolded. Table 2 lists, by benchmark, the composition of the POF by redundancy type.

We made the bins in Table 1 as large as possible without

complicating their definition (e.g., by combining two disjoint or partially overlapping sets). Consequently, some POF designs do not fit well into the bins. For example, consider the definition of a system according to the tuple (*Cores*, *D$ KB*, *Lanes*, *HTC*). (1, 32, 1, 1) is in the POF for ShortestPath, but employs no redundancy. However, expanding None to include it would require we also include (2, 32, 4, 4), which belongs to SSL. Each application has, on average, more than 25 POF designs. Three configurations are omitted from Filter, ShortestPath, and MergeSort; seven and five from LU and HotSpot, respectively; and, two and one from FFT and SVM, respectively. In the worst case (LU), incorporation of the outliers results in 0.2% variation in ARCR.

In general, we observe that systems with few cores (1-2 or 4) benefit the most from no redundancy. The cores in these systems often tend to be narrow (1-2 or 4 lanes); consequently, spare cores, lanes, or shared lanes, represent too great an overhead to be absorbed by increases in yield. For configurations with more cores, few lanes, and many HTCs (generally more than eight), SC is the best redundancy regime. As observed in Section 5.2, only a spare core covers defects in the HTCs; these systems do not use enough lanes to benefit from lane-level redundancy. Aside from Filter, applications do not generally benefit from a spare lane. Instead, SSL is used for systems with more lanes (generally more than four) and fewer HTC (usually less than eight).

For Filter, we observe that SC reduces cost up to 25%, with an ARCR of 11.2% in configurations with many HTC (more than 16); SC designs represent 24% of the POF. On the other hand, 19% the POF utilize SL. These configurations use big cores, and SL reduces cost by 1% more than SSL on average. No SSL designs appear in the POF.

Alternatively, consider KMeans. This benchmark tends to use small cores (less than 16 HTC per core), and as a result SSL outperforms other redundancy types. SSL designs reduce cost up to 20%, and by 8.3% on average, more than 1% better than the conventional approach, SL. Furthermore, 31% of POF designs use SSL. No SL designs appear in the POF for KMeans, or any other benchmark besides Filter.

Overall, we observe that when performance-cost Pareto-optimal designs are considered, a shared spare lane is a suitable substitute for a spare lane for most applications. For HotSpot and KMeans, SSL is not only the dominant redundancy technique, representing a greater share of the POF than SC, it also reduces costs about 0.4% more than SC, on average. Moreover, SSL reduces cost more than 0.5% more than SL on average across all benchmarks.

6. CONCLUSION

Designers today must allocate redundancy to combat decreasing manufacturing yields. In this paper, we investigated the application of redundancy allocated at different design granularities in the context of multi-core SIMT systems. In addition to evaluating the traditional approaches of core- and lane-sparing, we proposed spare lane sharing. Under spare lane sharing, the cost of the spare lane is amortized across two cores, making it affordable for a wider range of system configurations. Shared spare lanes can be integrated with negligible area and marginal performance overhead.

We observe that when systems consist of a few small cores, no redundancy reduces cost: the increase in area is too great an overhead to be absorbed by increases in yield. Spare cores increase yield best for applications that require many large, narrow cores: in these systems, cores are not dominated by lanes, but by other resources not explicitly protected by redundancy. Only spare cores can cover their failure.

In general, however, most perform best on systems with a number of wider cores. In this case, shared spare lanes offer the best reduction in cost. Notably, only a single benchmark, Filter, which requires very wide cores, has designs with spare lanes on its performance-cost Pareto-optimal front. In all other cases, shared spare lanes outperform spare lanes, making them a suitable substitute for spare lanes in the context of yield improvement in multi-core SIMT systems.

7. REFERENCES

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

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

[6] R. Stefan, *et al.*, "A 45 nm 8-core enterprise Xeon processor," *J. Solid-State Circuits*, vol. 45, no. 1, 2010.

[7] J. Meng, *et al.*, "Robust SIMD: Dynamically adapted SIMD width and multi-threading depth," in *IPDPS'12*.

[8] Y. Lin, *et al.*, "Soda: A low-power architecture for software radio," in *ISCA*, 2006.

[9] W. Raab, *et al.*, "A 100-gops programmable processor for vehicle vision systems," *Design Test of Computers, IEEE*, vol. 20, no. 1, 2003.

[10] C. Lyuh, *et al.*, "A novel reconfigurable processor using dynamically partitioned simd for multimedia applications," *ETRI*, vol. 31, no. 6, 2009.

[11] N. Shohei, *et al.*, "Performance evaluation of a dynamically switchable SIMD/MIMD processor by using an image recognition applications," *IPSJ T-SLDM*, vol. 3, no. 2, 2010.

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

[13] V. V. Kumar and J. Lach, "Heterogeneous redundancy for fault and defect tolerance with complexity independent area overhead," in *DFT*, 2003.

[14] S. Premkishore, *et al.*, "Exploiting microarchitectural redundancy for defect tolerance," in *ICCD*, 2003.

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

[16] 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.

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

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

[19] K. Andryc, *et al.*, "Flexgrip: A soft gpgpu for fpgas," in *FPT'13*, 2013, pp. 230–237.

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

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

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

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

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

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

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