StageWeb: Interweaving Pipeline Stages into a Wearout and Variation Tolerant CMP Fabric

Shantanu Gupta, Amin Ansari, Shuguang Feng and Scott Mahlke
Advanced Computer Architecture Laboratory
University of Michigan - Ann Arbor, MI
{shangupt, ansary, shoe, mahlke}@umich.edu

Abstract

Manufacture-time process variation and life-time failure projections have become a major industry concern. Consequently, fault tolerance, historically of interest only for mission-critical systems, is now gaining attention in the mainstream computing space. Traditionally reliability issues have been addressed at a coarse granularity, e.g., by disabling faulty cores in chip multiprocessors. However, this is not scalable to higher failure rates. In this paper, we propose StageWeb, a fine-grained wearout and variation tolerance solution, that employs a reconfigurable web of replicated processor pipeline stages to construct dependable many-core chips. The interconnection flexibility of StageWeb simultaneously tackles wearout failures (by isolating broken stages) and process variation (by selectively disabling slower stages). Our experiments show that through its wearout tolerance, a StageWeb chip performs up to 70% more cumulative work than a comparable chip multiprocessor. Further, variation mitigation in StageWeb enables it to scale supply voltage more aggressively, resulting in up to 16% energy savings.

Keywords: permanent faults, process variation, multicore, architecture, reliability

1 Introduction

From the time of its inception, the semiconductor process has witnessed an unhindered growth in transistor integration levels. However, in the forthcoming CMOS technology generations, this aggressive scaling poses critical reliability issues due to the increasing power density and process variation. First, as circuit density grows, each transistor gets smaller, hotter, and more fragile, leading to an overall higher susceptibility of chips to *permanent faults* [6]. These wearout failures, can impact the performance guarantees offered by a semiconductor chip, and limit their useful lifetime. In addition, manufacture-time *process variation* [19, 6], caused by the inability to precisely control the fabrication process at small-feature technologies, introduces significant deviation of circuit parameters (channel length, threshold voltage, wire spacing) from the design specification. Together, these reliability threats, permanent faults and process variation, have broad implications on semiconductor performance and power efficiency.

Resources on a chip multiprocessor (CMP) can be chiefly divided into two categories, computational cores and cache memory arrays. Fortunately, the regular nature of the memory layout makes it amenable to a wide variety of reliability solutions [3], including the well-known techniques such as row/column sparing and error-correcting codes (ECC). Thus, with appropriate protection mechanisms in place for caches, cores become the major source of wearout and process vulnerability on the die.

Wearout-tolerance for individual cores is a challenging problem. At one extreme is the option to disable cores as soon as they develop a fault [1], we refer to this as *core isolation*. Although it is a simple solution and imposes very little overhead, core disabling tends to be wasteful. With the increase in number of failures per chip, systems with core isolation can exhibit rapid throughput degradation, and quickly become useless. The other extreme, when repairing defective cores, is to leverage fine-grained micro-architectural redundancy [20, 21]. Here, the broken micro-architectural structures, such as functional units and reorder buffer entries, are isolated or replaced (with spares). Unfortunately, since a majority of the core logic is non-redundant [16], the fault coverage from these approaches is very limited. Therefore, these solutions also fall short of delivering sustainable throughput in the face of failures.

Process variation is encountered at manufacturing time, and influences almost every chip manufactured from day one. The variations can be systematic (e.g., lithographic lens aberrations) or random (e.g., dopant density fluctuations), and can manifest at different levels – wafer-to-wafer (W2W), die-to-die (D2D) and within-die (WID). Traditionally, D2D has been the most visible form of variation, and was tackled by introducing the notion of speed-binning (chips are partitioned based on their frequency and sold accordingly). However, the increasing levels of WID variations [19, 13] have created newer challenges for today's multicore designs. These parametric deviations can create a wide distribution of operating characteristics for components (like cores) within a chip, resulting in slow parts that work at a low frequency to those that are very fast but leaky (high static power). Figure 1 shows the impact of WID process variation on a 64-core CMP chip, and illustrates how the distribution of core frequencies is expected to widen at future technology nodes. To=

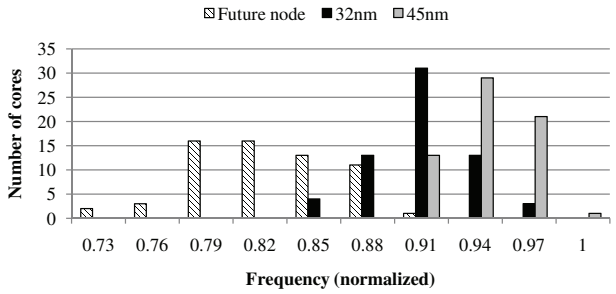


Figure 1: Impact of process variation on a 64-core CMP. The plot shows the distribution of core frequencies at current technology nodes (45nm and 32nm) and a (next-to-arrive) future node. As the technology is scaled, the distribution shifts towards the left (more slower cores) and widens out (more disparity in core frequencies). This is a consequence of large number of cores ending up with slower components, bringing down their operational frequencies.

deal with this challenge, designers in upcoming technology generations may create overly conservative designs or introduce large frequency guard-bands. Both of which are undesirable alternatives for computing efficiently.

To create robust and efficient systems in this landscape of wearout and variation-prone components, small architectural improvements over existing designs might not be sufficient. A rethinking of the architectural fabric from the ground up is needed, with reliability as a primary criteria. As a solution, this paper proposes *StageWeb*, a scalable many-core CMP that can deliver robust and efficient performance in the face of reliability hurdles. The design starts with StageNet (SN) [9], a recent proposal for multicore wearout tolerance. The basic idea of SN is to organize a multicore as a dynamically configurable network of pipeline stages. Logical cores are created at run-time by connecting together one instance of every pipeline stage. The underlying pipeline microarchitecture is designed to be completely decoupled at stage boundaries, providing full flexibility to construct logical cores. In the event of stage failures, the SN architecture initiates recovery by salvaging healthy stages to form logical cores. This ability of SN to isolate failures at a finer granularity (stages rather than cores) makes it less wasteful, and enables a SN chip to tolerate a significantly higher number of failures. Despite its benefits, the original SN proposal has three fundamental limitations that prevent it from meeting the many-core reliability challenge: 1) not scalable, 2) interconnection network is vulnerable to failures, and 3) process variation is not addressed.

StageWeb (SW), a scalable CMP fabric for interweaving wearout and variation-prone stages, eliminates all the aforementioned limitations of SN. The SW system is optimized to determine the best degree of connectivity between pipelines (that can share their resources together), while incurring a modest amount of overhead. A range of interconnection alternatives, and corresponding configuration algorithms, are explored to enable scalable fault-tolerance using SW. The reliability of the interconnection network is also tackled in the SW design through the use of spare crossbars, robust crossbar designs, and intelligent connectivity to give an illusion of redundancy. The underlying interconnection flexibility of SW is further leveraged to mitigate process variation. Using SW, the faster components (pipeline stages) in the fabric can be selectively picked, to form pipelines that can operate at a higher frequency. This ability of SW limits the harmful effects of process variation that intersperse slower components with faster ones throughout a chip.

The contributions of this paper are as follows:

1. SW, a comprehensive solution for the upcoming reliability challenges - permanent faults and process variation.
2. Exploration of robust and scalable interconnection alternatives for building SW chips.
3. Configuration algorithms to a) maximize the SW system throughput in the face of failures, and b) improve the distribution of core frequencies in the presence of process variation.

2 Background

The StageWeb architecture proposed in this paper builds upon StageNet (SN) [9], a solution for permanent fault tolerance in multicores. This section summarizes the design, advantages, and limitations of the SN architecture.

2.1 The StageNet (SN) Architecture

SN enables efficient stage level reconfiguration by means of an adaptable multicore fabric. Its ability to isolate defects and replace broken components at the pipeline stage granularity offers a good trade-off between area investment and reliability benefits. A SN multicore system is designed as a network of pipeline stages, rather than isolated cores. Processor cores within SN are constructed by linking together a set of working stages, where each stage corresponds to a node in the network. A logical core in the SN architecture is also referred to as a *StageNetSlice* (SNS).

SNS is the basic building block for the SN architecture. It consists of a decoupled pipeline microarchitecture that allows reconfiguration at the granularity of stages. As a basis for the SNS design, a simple in-order processor core is used, consisting of five stages: fetch, decode, issue, execute/memory, and write-back [14]. Figure 2 shows a single SNS with all of its microarchitectural additions (shaded structures). The stages are connected using full crossbar switches (to allow any to any communication in SN). The crossbar switches have a fixed channel width (64-bit) and, as a result, the transfer of an instruction from one stage to the next can take a variable number of cycles. These switches replace all direct wire links that exist between the pipeline stages including the bypass network, branch mis-prediction signals and stall signals. The interconnection network itself is bufferless, but the pipeline stages maintain a latch (double buffer) at their input and output, making the network a separate stage and preventing it from impacting critical paths within stages. The decoupling of pipeline stages creates three fundamental challenges for the SNS microarchitecture:

1. Global signals for flush/stall are infeasible
2. Data forwarding is hard to support
3. Performance degradation due to crossbar switches

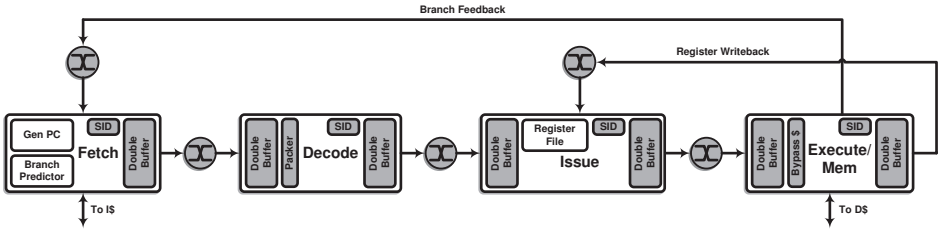


Figure 2: The StageNet Slice (SNS) micro-architecture. The original pipeline is decoupled at the stage boundaries, and narrow-width crossbars (64-bit) replace the direct pipeline-latch connections. The shaded blocks inside the stages highlight the structures added to enable decoupled functionality.

SN provides a solution for each of the challenges listed above. Specifically, it proposes *stream identification* (StreamID) for global control handling, *bypass cache* (Bypass$) for data forwarding, and *macro operations* (MOPs) for performance. With the application of these mechanisms, and some minor ones detailed in [9], the performance of SNS comes within 10% of a baseline in-order pipeline (Figure 3).

2.2 Fault Tolerance using SN

A network of stages can be grouped together, using the full crossbar switches, to form a SN multicore. Figure 4 illustrates a SN multicore created out of four SNSs that share a common crossbar network. The inherent symmetry of SN allows arbitrary formation of a logical SNS by grouping together at least one pipeline stage of each type. For instance, the fetch, issue and execute stage from slice 0 are linked with the decode from slice 1, to construct a working pipeline.

SN relies on a fault detection mechanism to identify broken stages and trigger reconfiguration. The manufacture time failures can be easily identified at the test time and SN can be configured accordingly. However, an active mechanism is required to catch failures in the field. There are two possible solutions for detection of permanent failures: 1) continuous monitoring using sensors [5] or 2) periodic testing for faults [8]. SN can employ either of these or use a hybrid approach. In the presence of failures, SN can easily isolate broken stages by adaptively routing around them. Given a pool of stage resources, a software based configuration manager can divide them into a globally optimal set of logical SNSs. In this way, SN's interconnection flexibility allows it to salvage healthy stages from adjacent cores.

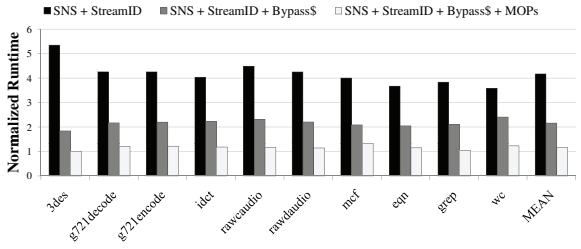


Figure 3: Single thread performance of a SNS normalized to a baseline in-order core. The performance improves by almost a factor of four after applying all proposed micro-architectural modifications.

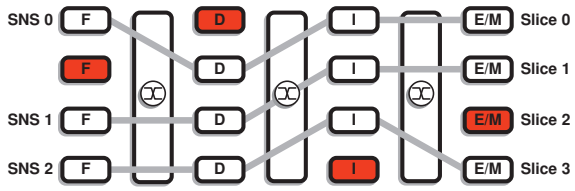


Figure 4: The SN architecture with four slices interconnected to each other. Despite four failed stages (marked by shading), SN is able to salvage three working pipelines, maintaining healthy system throughput. Given a similar fault map, a core-disabling approach for reliability would lose all working resources.

2.3 Limitations of SN

The SN design is an acceptable wearout solution for a small scale multicore system. However, SN is limited in three distinct ways that prevent it from meeting the many-core reliability challenge. First, SN was designed for a CMP with 4-8 cores, and does not scale well to a large number of cores. The crossbar, that was used as the SN interconnect, is notorious for steep growth in area and delay overheads as the number of ports is increased [15], and therefore limits SN scaling. Second, SN focuses primarily on stage failures and does not investigate methods for interconnection fault tolerance. SN's robustness hinges on the link and crossbar reliability. For instance, a SN chip will waste all of its working stages if the shared crossbar between them develops a failure. And finally, the SN design targets only wearout related failures, which constitutes only a part of the reliability challenge. A more immediate concern for the industry today is the accelerating rate of process variation, and its impact on the performance-efficiency of semiconductor products.

3 The StageWeb Architecture

StageWeb (SW) is a scalable architecture for constructing dependable CMPs. SW *interweaves* pipeline stages and interconnection into an adaptive fabric that is capable of withstanding wearout failures as well as mitigating process variation. The interconnect is designed to be flexible such that the system can react to local failures, reconfiguring around them, to maximize the computational potential of the system at all times. Figure 5 shows a graphical abstraction of a large scale CMP employing the SW architecture. The processing area of the chip in this figure consists of a grid of pipeline stages, interconnected using a scalable network of crossbars switches. The pipeline microarchitecture of SW is the same as that of a SNS. Any complete set of pipeline stages (reachable by a common interconnection) can be assembled together to form a logical pipeline.

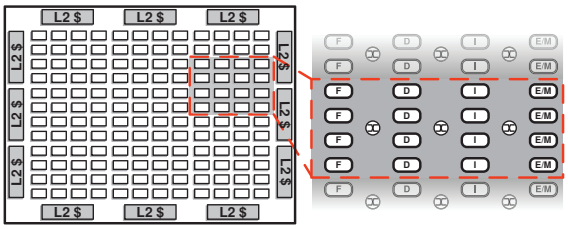


Figure 5: The SW architecture. The pipeline stages are arranged in form of a grid, surrounded by a conventional memory hierarchy. The inset shows a part of the SW fabric. Note that the figure here is an abstract representation and does not specify the actual number of resources.

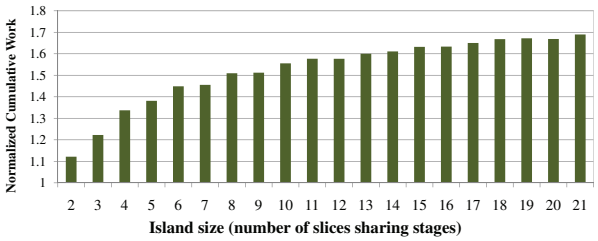


Figure 6: Cumulative work performed by a fixed size SW system with increasing SW island width. The results are normalized to an equally provisioned regular CMP. These results are a theoretical upper bound, as we do not model interconnection failures for this experiment.

The fault-tolerance within SW can be divided into two sub-problems. The first half is to utilize as many working pipeline stages on a chip as possible (interconnect scalability). And the second half is to ensure interconnect reliability. A naive solution for the first problem is to provide a connection between all stages. However, as we will show later in this section, full connectivity is not necessary between all stages on a chip to achieve the bulk of reliability benefits. As a combined solution to both these problems, we explore alternatives for the interconnection network, interconnection reliability and present configuration algorithms for the same. The underlying interconnection infrastructure is also leveraged by SW to mitigate process variation. The insight here is to segregate slower and faster components (stages), allowing more pipelines to operate at a higher frequency.

3.1 Interweaving Range

The reliability advantages of SW stem from the ability of neighboring slices (or pipelines) to share their resources with one another. Thus, a direct approach for scaling the original SN proposal would be to allow full connectivity, i.e. a logical SNS can be formed by combining stages from anywhere on the chip. However, such flexibility is unnecessary, since the bulk of reliability benefits are garnered by sharing amongst small groups of stages. To verify this claim, we conducted an experiment with a fixed number of pipeline resources interwoven (grouped together) at a range of values. Each fully connected group of pipelines is referred to as a *SW island*. Figure 6 shows the cumulative work done by a fixed number of slices interwoven at a range of SW island sizes. The cumulative work metric, as defined in Section 4.3, measures the amount of useful work done by a system in its entire lifetime. Note that the interconnect fabric here is kept fault free for the sake of estimating the upper bound on the throughput offered by SW.

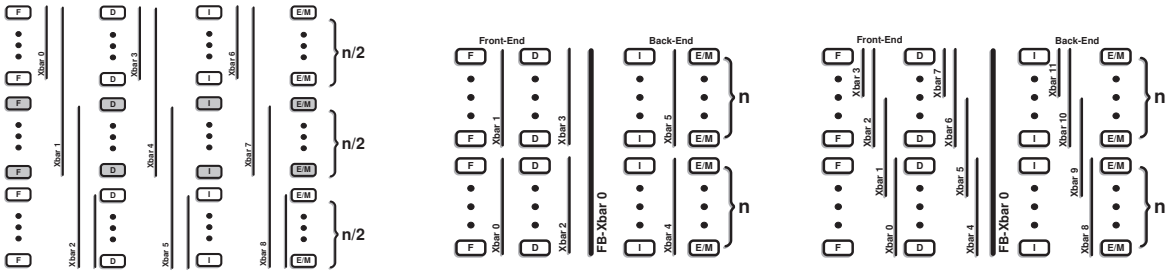
As evident from Figure 6, a significant amount of defect tolerance is accomplished with just a few slices sharing their resources. The reliability returns diminish with the increasing number of pipelines, and beyond 10-12 pipelines, interweaving has a marginal impact. This is because as a SW island spans more and more slices, the variation in time to failure of its components gets smaller and smaller. This factors into the amount of gains that the flexibility of the interconnect can garner in combining working stages, resulting in a diminishing return with an increase in island width. Thus, a two-tier design can be employed for SW by dividing the chip into SW islands, where a full interconnect is provided between stages within the island and no (or perhaps limited) interconnect exists between islands. In this manner, the wiring overhead can be explicitly managed by examining more intelligent system organizations, while garnering near-optimal benefits of the SW architecture.

3.2 Interweaving Candidates

Interweaving a set of 10-12 pipelines together, as seen in Figure 6, achieves a majority of the SW reliability benefits (assuming a failure immune interconnect). However, using a single crossbar switch might not be a practical choice for connecting all 10-12 pipelines together because: 1) the area overhead of crossbars scales quadratically with the number of input/output ports, 2) stage to crossbar wire delay increases with the number of pipelines connected together, at some point this can become the critical path for the design, 3) and lastly, failure of the single crossbar shared by all the pipelines can compromise the usability of all of them. In light of the above reasons, there is a need to explore more intelligent interconnections that can reach a wider set of pipelines while keeping the overheads in check.

Single Crossbars: The simplest interconnection option is to use full crossbars switches that connect n slices. Here, the value of n is bounded by the combined delay of crossbar and interconnection, which should not exceed a single CPU cycle. Note that the interconnection network in SW constitutes a separate stage and does not change the timing of individual pipeline stages.

Overlapping Crossbars: The overlapping crossbar interconnect builds upon the single crossbar design, while enabling a wider number of pipelines to share their resources together. As the name implies, adjacent crossbars overlap half of their territories in this interweaving setup. Figure 7(a) illustrates the deployment of overlapping crossbars over $\frac{3}{2}n$ slices. Unlike the single crossbar interconnect, overlapping crossbars have a fuzzy boundary for the SW islands. The shaded stages in the figure highlight a repetitive interconnection pattern here. Note that these $\frac{n}{2}$ stages can connect to the stages above them using crossbars Xbar 1,4,7, and to the stages below them using crossbars Xbar 2,5,8. Thus, overall these stages have a reach of $\frac{3}{2}n$ slices. The overlapping crossbars have



(a) Overlapping crossbar connections. In this figure, the shaded stages in the middle have a reach of $\frac{3}{2}n$ pipelines.

(b) Combined application of single crossbars in conjunction with front-back crossbars.

(c) Combined application of overlapping crossbars in conjunction front-back crossbars.

Figure 7: Interweaving candidates. The vertical lines here are used as an abstraction for full crossbar switches. The span of the lines represent the pipelines they connect together.

two distinct advantages over the single crossbars: 1) allows up to 50% more slices to share their resources together, and 2) introduces an alternative crossbar link at every stage interface, improving the interconnection robustness.

Single and Front-Back Crossbars: The primary limitation of single crossbars is the interweaving range they can deliver. This value is bounded by the extent of connectivity a single-cycle crossbar can provide. However, if this constraint is relaxed by introducing two-cycle crossbars, twice the number of slices can communicate with one another. Unfortunately, the two cycle latency between every pair of stages can introduce a significant slowdown on the single thread performance of the logical SNSs (~25%). A compromise solution would be to apply two cycle crossbar at a coarser granularity than pipeline stages. One way to accomplish this is by classifying the fetch-decode pair as one block (front-end), and the issue-exmem pair as the other (back-end). The single thread performance loss when using this is about 7%. As per Figure 2, connecting up these two blocks would need one front-end to back-end crossbar, and another in the reverse direction. We call such two-cycle interconnections front-back crossbars. Figure 7(b) shows $2n$ slices divided into front-end and back-end blocks, which are connected by a front-back crossbar (FB-Xbar 0).

Overlapping and Front-Back Crossbars: The single and front-back crossbar combination benefits from the interweaving range it obtains from the front-back crossbar, but, at the expense of single thread performance loss. An alternative is to combine the overlapping crossbars with the front-back crossbars. Figure 7(c) shows this style of interconnect applied over $2n$ slices. In this scenario, $\frac{3}{2}n$ slices can be reached without losing any performance, and the remaining $n/2$ bordering slices can be reached using the front-back crossbars.

3.3 Configuration Algorithms

The faults in a SW chip can manifest as broken stages, crossbar ports or interconnect links. Each of these scenarios demand a reconfiguration of the system such that the defective components are isolated. A good configuration algorithm would guarantee formation of a maximum number of logical pipelines (or SNSs), thus achieving the highest possible system throughput. This section presents three configuration algorithms for handling each type of crossbar deployment, namely, single crossbars, overlapping crossbars and front-back configurations. All four interweaving alternatives discussed in the Section 3.2 can be successfully configured by using a combination of these three algorithms.

Single Crossbar Configuration: The input to this algorithm is the fault map of the entire SW chip, and it is explained here using a simple example. Figure 8 shows a four-wide SW system. The SW islands are formed using the top two and bottom two slices. There are eight defects in this system, four stage failures and four crossbar port/interconnect link failures. The dead stages are marked using a solid shade (F2, D4, I3, E4) and the interconnect as crosses. The stages connected to a dead interconnect are also declared dead, and are lightly shaded (D1, D2, I2). This is to distinguish them from the physically defective stages. For illustration purposes, the backward connections are not shown here and are assumed fault-free.

Given the updated fault-map (with interconnection failures modeled as stage failures), the single crossbar configuration is conducted for one SW island at a time. The first step is to create a list of working stages of each type. The second step groups unique working stages within an island, and sets them aside as a logical SNS. In our example, this results in having only *one* working SNS: F3, D3, I4, E3.

Overlapping Crossbar Configuration: The overlapping crossbars provide additional connectivity for resources from two neighboring SW islands. For the ex-

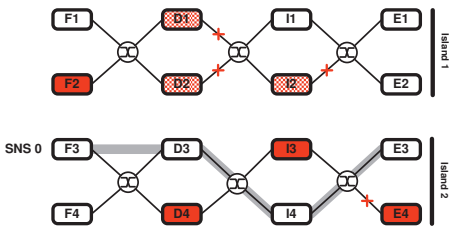


Figure 8: Configuration of SW with single crossbars. The marked stages and interconnections are dead. Island 1 is not able to form any logical SNS, whereas island 2 forms only one logical SNS (SNS 0).

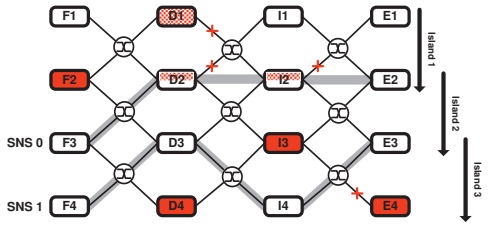


Figure 9: Configuration of SW with overlapping crossbars. The red marked stages and interconnections are dead. The partially marked stages are dead for one island, but are available for use in the other. Island 1 is not able to form any logical SNS, island 2 and 3 form one logical SNS each.

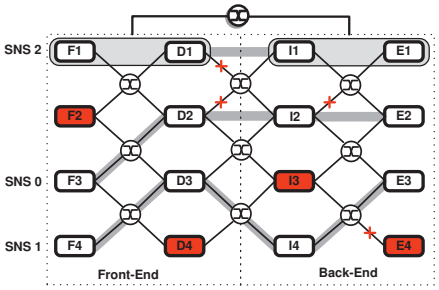


Figure 10: Configuration of SW with overlapping and front-back crossbars. The front-back crossbars add one more logical SNS (SNS 2) over the configuration result of overlapping crossbars, making the total *three*.

planation of this algorithm, we will use the same SW example from before. The addition of overlapping crossbars makes the tally for the number of logical SN islands three (see Figure 9). Also, note the change in shading used for stages D2 and I2. The top half of these stages are lightly shaded, and the bottom half is clear. This is to denote that these stages are dead for use in island 1, but are available for use in island 2.

Given the fault-map, and the proper abstraction of interconnection faults as stage faults, the single crossbar configuration algorithm is used to form logical SNSs for one island at a time. This process is started at one end of the SW fabric, and is swept across the entire SW. When this process is started at the top of the fabric, working stages from the top of the pile within each island are given preference to form logical SNSs. This heuristic helps in keeping more resources free when the succeeding islands are configured. Figure 9 illustrates this logical progression from island 1 to island 3 in our example. The steps for each island configuration are detailed below, and result in a total of *two* logical SNSs.

1. **Island 1:** Free working stages: fetch {F1}, decode {}, issue {I1}, execute/memory {E1,E2}.
   Logical SNSs: *none.*
2. **Island 2:** Free working stages: fetch {F3}, decode {D2, D3}, issue {I2}, execute/memory {E2, E3}.
   Logical SNSs: F3, D2, I2, E2.
3. **Island 3:** Free working stages: fetch {F4}, decode {D3}, issue {I4}, execute/memory {E3}.
   Logical SNSs: F4, D3, I4, E3.

Front-Back Crossbar Configuration: The front-back crossbars are only used to connect the front-end (fetch-decode pair) with the back-end (issue-execute/memory pair). This requires their use to be in conjunction with some other crossbar configuration (see Section 3.2). Henceforth, we will refer to this *other crossbar configuration* as the first-level interconnect. Nevertheless, the configuration algorithm for front-back crossbars is independent of the choice made for the first-level interconnection. The running example from the previous algorithms will again be employed in this section (see Figure 10). In our example (Figure 10) front-back crossbars are assumed to be fault-free. The front-back algorithm can be divided into three phases:

1. *First-level Interconnect:* For this example, we employ overlapping crossbars as the first-level interconnection. This results in forming two logical SNSs: F3, D2, I2, E2 and F4, D3, I4, E3.
2. *Front-back Bundling:* In this step, the resources remaining in the SW fabric are individually bundled up in the front-end and the back-end. Figure 10 example forms one front-end bundle (F1, D1) and one back-end bundles (I1, E1).
3. *Front-back Integration:* The last phase in the configuration is to combine pairs of front-end and back-end bundles and form logical SNSs. Figure 10 forms one logical SNS using the front-back crossbars: F1, D1, I1, E1.

The configuration algorithms discussed in this section can cover all possible interweaving candidates discussed in Section 3.2. It is noteworthy that the algorithms presented here are not optimal (in specific, the latter two), and are based on heuristics. This was done in order to keep their run-times linear ($O(n)$), thereby, minimizing the overheads from manufacture-time and in-field reconfiguration.

3.4 Interconnection Reliability

Interconnection reliability can be divided into link reliability and crossbar reliability. The link reliability is accounted for, to a certain extent, by the interconnection alternatives which introduce redundancy. Further, they are not as vulnerable to wearout and variation as logic. For crossbar reliability, SW can use three alternatives:

1. *Simple Crossbar:* This is the simplest scenario with a single crossbar switch used at each interconnection spot. No redundancy is maintained in this case.
2. *Simple Crossbar with spare(s):* In this set-up, one spare is maintained for every crossbar in the system. The cold spare corresponding to a crossbar switch is only brought into use when the latter develops a certain number of port failures.
3. *Fault-Tolerant Crossbar (no spares):* The most expensive alternative is to deploy one-sided fault-tolerant (FT) crossbars [24] that nearly eliminate the chances of crossbar failures. Note that in a FT crossbar, multiple paths exist from a given input port to the output port. This is unlike a regular crossbar that have a unique path for every input-output pair. However, FT crossbars tend to be two/three times the size of a regular crossbar.

3.5 Variation Tolerance

Process variation introduces slower circuit components throughout a chip. This presence of slower components results in a wide distribution of operational frequencies for different structures on the die. For instance, in a conventional CMP, the slowest structure within each core would determine the best frequency achievable by that core. Similarly, in the case of SW, this impact can be observed at the granularity of pipeline stages, a few of which will be much slower than others. However, unlike a conventional CMP, SW can selectively salvage faster pipeline stages from the grid of resources and construct logical pipelines that can operate at a higher frequency. This will result in an improved distribution of core frequencies as compared to a traditional CMP with isolated cores.

The configuration methodology of SW in the presence of process variation builds upon the algorithms discussed earlier. The key observation is that for a given frequency target (and fixed supply voltage), pipeline stages can be marked functional or non-functional. Once this level of abstraction is reached, the non-functional stages can be treated in the same manner as broken stages were earlier in this section. Given a SW chip with a wide variation in pipeline stage frequencies, the algorithm proceeds as follows. It start with the highest possible frequency, and marks the working stages in the grid. The standard configuration algorithm is used to form logical pipelines. The frequency is now reduced by a unit step, and the process is repeated. This is continued until the configuration is defined for the lowest operational frequency of the system. At this point, the number of cores functional at each frequency point can be tabulated.

Apart from enhancing the performance, the improvement in core frequencies using SW can also be translated into energy savings relative to a conventional CMP. The insight here is that given a system utilization level (fraction of cores occupied), SW can form the fastest cores from its pool of stages and meet the frequency target at a lower operational voltage than a CMP. Since the CMP lacks the flexibility to combine faster stages across its cores, it will be forced to run at a higher voltage to meet the same frequency target. This difference in voltage translates to (quadratic) dynamic power savings and (cubic) static power savings [7]. As both systems operate at the same frequency, these power savings map directly to energy savings.

4 Evaluation

4.1 Methodology

Microarchitectural Simulation: The microarchitectural simulator for the SW evaluation was developed using the Liberty Simulation Environment (LSE) [23]. Two flavors of the microarchitectural simulator were implemented in sufficient detail to provide cycle accurate results for single thread performance. The first simulator models a five stage pipeline, which is used as the baseline. The second simulator models the decoupled SNS pipeline microarchitecture with all its enhancements (see Section 2.1). Table 1 lists the parameters for the core and the memory hierarchy used for the simulations. These parameters and the baseline microarchitecture pipeline stages are modeled after the OR1200 processor [14], an open source RISC microprocessor.

Table 1: Architectural parameters.

Pipeline	4-stage in-order OR1200 RISC [14]
Frequency	400 MHz
Area	$1mm^2$ (90nm process)
Branch predictor	Global, 16-bit history, gshare predictor BTB size - 2KB
L1 I$, D$	4-way, 16 KB, 1 cycle hit latency
L2 $	8-way, 64 KB (per core), 5 cycle hit latency
Memory	40 cycle hit latency

Wearout and Process Variation Modeling: For the wearout failures, the mean-time-to-failure (MTTF) was calculated for the various stages and crossbars in the system using the empirical models from [21]. The entire core was qualified to have a MTTF of 10 years. These wearout models heavily depend on the module (stages and crossbar) temperatures that were generated using HotSpot [10]. A customized floorplan was created for StageWeb to account for the lateral heat transfer on the die. Finally, the calculated MTTFs are used as the mean of the Weibull distributions for generating time to failures (TTF) for each module (stage/crossbar) in the system. The stages are considered dead as a whole when a fault occurs, whereas, the crossbar failures are modeled at the crossbar-port granularity.

Process variation was modeled using VARIUS [19]. Given a chip's floorplan, and σ/μ for a technology process, VARIUS can be used to obtain the spread of operational frequencies for all structures on the die. In our experiments, we use σ/μ of 0.25, as a representative value for technologies beyond $32nm$.

Area, Power and Timing: Industry standard CAD tools with a library characterized for a 90nm process are used for estimating the area, power and timing for all design blocks. A Verilog description for the OR1200 microprocessor was obtained from [14]. All other design blocks, SNS enhancements, and crossbar configurations were hand-coded in Verilog. The area for the interconnection links between stages and crossbars (interweavings) was estimated using the same methodology as in [12] with intermediate wiring-pitch at 90nm taken from the ITRS road map [11]. The power consumption for all structures was computed using Synopsys Power Compiler. For the power saving experiments, we assume that dynamic power scales quadratically with supply voltage, and linearly with frequency [17]. The synthesis tool chain (used for area) was also employed to find the target frequency for the design. The interconnection link delay between stages and crossbars was estimated using the intermediate wiring-delay from the ITRS road map [11].

CMP Simulations: A thorough simulation infrastructure was developed to simulate a variable-size regular CMP system and SW system. This infrastructure integrates all components of our evaluation methodology and SW design: single thread performance, wearout modeling, interweaving alternatives, configuration algorithms

and crossbar models. To obtain statistically significant results, 1000 Monte-Carlo runs were conducted for every lifetime reliability experiment.

For lifetime reliability experiments, the stages/crossbars fail as they reach their respective time-to-failures (TTFs). The system gets reconfigured over its lifetime whenever a failure is introduced. The instantaneous throughput of the system is computed for each new configuration using the number of logical SNSs. This way, we can obtain the chip's throughput over its lifetime.

4.2 StageWeb Design Space

For the latest generation Intel Core 2 processors, about 60% die area is occupied by the processing cores. With that estimate, in order to accommodate 64 OR1200 RISC cores (our baseline in-order core) we assume a $100mm^2$ die (a typical value for multicore parts). We use this die area as the basis for constructing various SW chip configurations. There are a total of twelve SW configurations that we evaluate, distinguished by their choice of interweaving candidates (single, single with front-back, overlap, overlap with front-back) and the crossbars (no spare, with spare, fault-tolerant). Table 2 shows the twelve configurations that form the SW design space. The cap on the processing area guarantees an area-neutral comparison in our results. In the base CMP case, the entire processing area can be devoted to the cores, giving it a full 64 cores.

Table 2: Design space for SW. The rows span the different interconnection types (F/B denotes front-back), and the columns span the crossbar type: crossbar w/o (without) sp (spares), crossbar w/ sp and fault-tolerant (FT) crossbar. Each cell in the table mentions the number of pipeline slices, in each SW configuration, given the overall chip area budget ($100mm^2$).

	Xbar (w/o sp)	Xbar (w/ sp)	FT Xbar
Single Xbar	56	55	54
Single + F/B Xbar	55	53	52
Overlap Xbar	55	53	52
Overlap + F/B Xbar	54	51	50

The interconnection (crossbar + link) delay acts as a limiting factor while connecting a single crossbar to a group of slices. As per our timing analysis, the maximum number of slices that can be connected using a single crossbar is 6. This is for the 90nm technology node and a single-cycle crossbar. A two-cycle crossbar (that is used as the Front-Back crossbar) can connect up to 12 slices together.

4.3 Cumulative Work

The lifetime reliability experiments, as discussed in the evaluation methodology, track the system throughput over its lifetime. The cumulative work, used in this section, is defined as the total work a system can accomplish during its entire lifetime, while operating at its peak throughput. In simpler terms, one can think of this as the total number of instructions committed by a CMP during its lifetime. This metric is same as the one used in [9]. All results shown in this section are for 1000 iteration Monte-Carlo simulations.

Figure 11 shows the cumulative work results for all

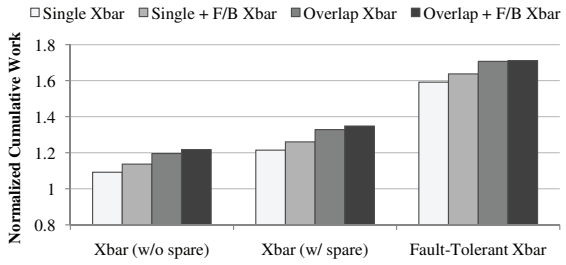


Figure 11: Cumulative work performed by the twelve SW configuration normalized to a CMP system. The cumulative work improves with the richer choices for interweaving, as well as with the more resilient crossbars.

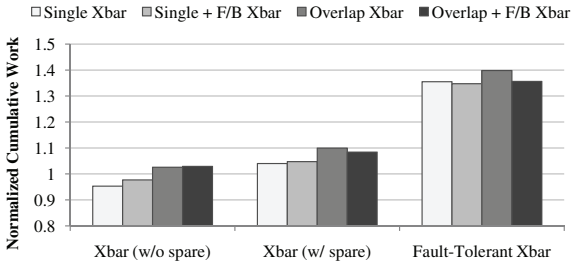


Figure 12: Cumulative work performed by the twelve SW configuration normalized to a CMP system (*area-neutral study*). The cumulative work improves with more resilient crossbar choice. However, richer interweaving does not map directly to better results. In the best case, a SW system achieves 40% more cumulative work relative to the CMP system.

twelve SW configurations, normalized to what is achievable using a 64 core traditional CMP. The results categorically improve with increasing interweaving richness, and better crossbar reliability. The biggest gains are achieved when transitioning from the regular crossbar to the fault-tolerant crossbar. This is due to the ability of the fault-tolerant crossbar to effectively use its internal fine-grained cross-point redundancy [24], while maintaining fault-free performance. When using the fault-tolerant crossbars, SW system can deliver up to 70% more cumulative work (overlapping with front-back configuration) over a regular CMP.

The same set of experiments were repeated in an area-neutral fashion for the twelve SW configurations (using the data from Table 2). Figure 12 shows the cumulative work results for the same. The trend of improving benefits while transitioning to a more reliable crossbar remains true here as well. However, the choice of the best interweaving candidate is not as obvious as before. Since the area of each interconnection alternative is factored-in, the choice to use a richer interconnect has to be made at the cost of losing computational resources (pipelines). For instance, the (fault-tolerant) overlapping crossbar configuration (column 11) fares better than the (fault-tolerant) overlapping with front-back crossbar configuration (column 12). The best result in this plot (fault-tolerant overlapping crossbar) achieves 40% more cumulative work than the baseline CMP.

4.4 Throughput Behavior

The cumulative work done by the system is a useful metric, but is insufficient in showing the quality of system's behavior during its lifetime. For this purpose, we conducted an experiment to track the system throughput over its lifetime (Figure 13), as wearout failures occur. Three systems configurations are compared head-to-head: SW's best configuration *fault-tolerant overlapping crossbars*, area-neutral version of *fault-tolerant overlapping crossbars*, and the baseline CMP. As evident from Figure 13, the throughput for the SW system exhibits a very graceful degradation with the progression of time. At the beginning of life, the CMP system has an edge over the SW system. This is due to the higher number of pipeline resources a CMP system initially possesses. However, the SW catches up soon enough into the lifetime, and maintains its advantage for the remaining lifetime.

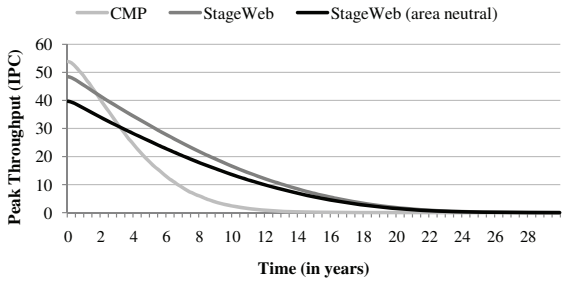


Figure 13: This chart shows the throughput over the lifetime for the best SW configurations and the baseline CMP. The throughput for the SW system degrades much more gradually, and in the best case (around the 8 year mark), SW delivers 4X the throughput of CMP.

4.5 Variation Mitigation

In addition to wearout tolerance, the interconnection flexibility of SW can also be leveraged to mitigate process variation. As discussed in Section 4.5, the basic idea is to group together faster pipeline stages to form pipelines that can run at higher frequencies. This way, the slower resources are isolated, reducing their overall performance impact. Figure 14 shows the distribution of core frequencies for a regular CMP system and a SW CMP with overlapping configuration. In this experiment, both systems contain 64 cores each, and process variation is injected with $\sigma/\mu = 0.25$. The results confirm that the distribution of core frequencies in a SW CMP are considerably better than that of a conventional CMP. The mean increase in the core frequencies is 7%. It is noteworthy that the slowest cores in both systems operate at the same frequency (0.73). This is true by construction, since even in a SW CMP, some logical pipeline has to absorb the slowest stage and operate at that frequency.

4.6 Power Saving

The better distribution of frequencies, as discussed in Section 4.5, can also translate into power/energy savings. For a given system utilization, SW can scale down the supply voltage (reducing power quadratically) and still provide the same level of performance as a baseline CMP.

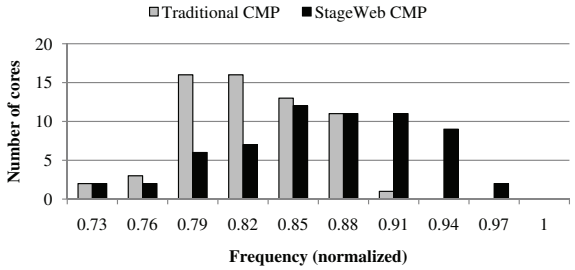


Figure 14: The distribution of core frequencies in 64-core CMP and StageWeb chips. Facing the same level of process variation, SW enables a noticeable improvement in the frequency distribution.

Note that a single global supply voltage is assumed in all our experiments. This is a commonly accepted practice as multiple supply sources introduce significant noise. Figure 15 shows the power savings obtained at different levels of system utilization (fraction of cores occupied) when using SW. Each bar is normalized to the CMP power at that utilization level. The results range from 16% power saving at 12.5% utilization to a small loss in power at 100% utilization. When the utilization is low, more opportunity exists for SW to gather faster stages, and switch off the slowest ones. But, at full utilization, everything (including the slowest stage) has to be switched on, requiring the global supply voltage to be scaled back to its original level. Most commercial servers have time-varying utilization [2] (segments of high and low utilization), and can be expected to create many opportunities where SW saves power. Since this power is saved without any accompanying loss in performance (frequency), it translates directly to energy savings.

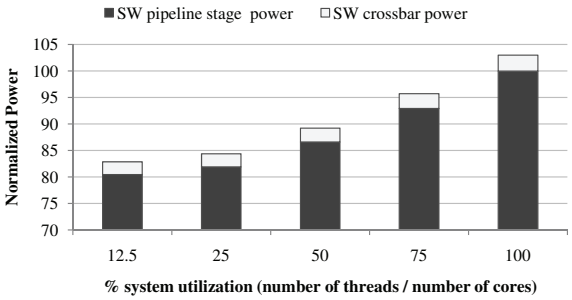


Figure 15: Power saving using SW relative to a CMP at different system utilization levels.

5 Related Work

High-end server systems (such as Tandem NonStop and IBM zSeries [4]) designed with reliability as a first-order design constraint have been around for decades but have typically relied on coarse grain replication to provide a high degree of reliability However, dual and triple modular redundant systems incur significant overheads in terms of area and power, and cannot tolerate a high failure rate. Configurable Isolation [1], and Architectural Core Salvaging [16] are more recent proposals for multiprocessor fault tolerance. Configurable Isolation disables the broken cores for tolerating faults, whereas Core Salvaging enables use of a broken core in 20-30% of the cases by

re-scheduling a thread to a fully functional core right before a failure is encountered. Unfortunately, these recent proposals are also designed to work at small failure rates.

A newer category of techniques use stage-level reconfiguration for reliability. StageNet (SN) [9] groups together a small set of pipelines stages with a simple crossbar interconnect. By enabling reconfiguration at the granularity of a pipeline stage, SN can tolerate many more failures. Romanescu et al. [18] also propose a multicore architecture, Core Cannibalization Architecture (CCA), that exploits stage level reconfigurability. However, CCA allows only a subset of pipelines to lend their stages to other broken pipelines, thereby avoiding full interconnection.

The prior research efforts on tolerating process variation have mostly relied on using fine-grained VLSI techniques such as adaptive body biasing / adaptive supply voltage [22], voltage interpolation [13], and frequency tuning. Although effective, all such solutions can have high overheads, and their feasibility has not been established in mass productions. SW stays clear of any such dependence on circuit techniques, and mitigates process variation with a fixed global supply voltage and frequency.

6 Conclusion

With the looming reliability challenge in the future technology generations, mitigating process variation and tolerating in-field silicon defects will become necessities in future computing systems. In this paper, we propose a scalable alternative to the tiled CMP design, named StageWeb (SW). SW fades out the inter-core boundaries and applies a scalable interconnection between all the pipeline stages of the CMP. This allows it to salvage healthy stages from different parts of the chip to create working pipelines. In our proposal, the flexibility of SW is further enhanced by exploring a range of interconnection alternatives and the corresponding configuration algorithms. In addition to tolerating failures, the flexibility of SW is also used to create more power-efficient pipelines, by assembling faster stages and scaling down the supply voltage. The best interconnection configuration for the SW architecture was shown to achieve 70% more cumulative work over a regular CMP containing equal number of cores. Even in an area-neutral study, SW system delivered 40% more cumulative work than a regular CMP. And lastly, in low system utilization phases, its variation mitigation capabilities enable SW to achieve up to 16% energy savings.

7 Acknowledgements

We thank the anonymous referees for their valuable comments and suggestions. The authors acknowledge the support of the Gigascale Systems Research Center, one of five research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation program. This research was also supported by National Science Foundation grants CCF-0916689 and ARM Limited.

References

[1] N. Aggarwal, P. Ranganathan, N. P. Jouppi, and J. E. Smith. Configurable isolation: building high availability systems with commodity multi-core processors. In *Proc. of the 34th Annual International Symposium on Computer Architecture*, pages 470–481, 2007.

[2] A. Andrzejak, M. Arlitt, and J. Rolia. Bounding the resource savings of utility computing models, Dec. 2002. HP Laboratories, http://www.hpl.hp.com/techreports/2002/HPL-2002-339.html.

[3] A. Ansari, S. Gupta, S. Feng, and S. Mahlke. Zerehcache: Armoring cache architectures in high defect density technologies. In *Proc. of the 42nd Annual International Symposium on Microarchitecture*, 2009.

[4] W. Bartlett and L. Spainhower. Commercial fault tolerance: A tale of two systems. *IEEE Transactions on Dependable and Secure Computing*, 1(1):87–96, 2004.

[5] J. Blome, S. Feng, S. Gupta, and S. Mahlke. Self-calibrating online wearout detection. In *Proc. of the 40th Annual International Symposium on Microarchitecture*, pages 109–120, 2007.

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

[7] D. Ernst, N. S. Kim, S. Das, S. Pant, T. Pham, R. Rao, C. Ziesler, D. Blaauw, T. Austin, and T. Mudge. Razor: A low-power pipeline based on circuit-level timing speculation. In *Proc. of the 36th Annual International Symposium on Microarchitecture*, pages 7–18, 2003.

[8] S. Gupta, A. Ansari, S. Feng, and S. Mahlke. Adaptive online testing for efficient hard fault detection. In *Proc. of the 2009 International Conference on Computer Design*, 2009.

[9] S. Gupta, S. Feng, A. Ansari, J. Blome, and S. Mahlke. The stagenet fabric for constructing resilient multicore systems. In *Proc. of the 41st Annual International Symposium on Microarchitecture*, pages 141–151, 2008.

[10] W. Huang, M. R. Stan, K. Skadron, K. Sankaranarayanan, and S. Ghosh. Hotspot: A compact thermal modeling method for cmos vlsi systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 14(5):501–513, May 2006.

[11] ITRS. International technology roadmap for semiconductors 2008, 2008. http://www.itrs.net/.

[12] R. Kumar, N. Jouppi, and D. Tullsen. Conjoined-core chip multiprocessing. In *Proc. of the 37th Annual International Symposium on Microarchitecture*, pages 195–206, 2004.

[13] X. Liang, R. Canal, G.-Y. Wei, and D. Brooks. Replacing 6t srams with 3t1d drams in the l1 data cache to combat process variability. *IEEE Micro*, 28(1):60–68, 2008.

[14] OpenCores. OpenRISC 1200, 2006. http://www.opencores.org/projects.cgi/web/ or1k/openrisc_1200.

[15] L.-S. Peh and W. Dally. A delay model and speculative architecture for pipelined routers. In *Proc. of the 7th International Symposium on High-Performance Computer Architecture*, pages 255–266, Jan. 2001.

[16] M. D. Powell, A. Biswas, S. Gupta, and S. S. Mukherjee. Architectural core salvaging in a multi-core processor for hard-error tolerance. In *Proc. of the 36th Annual International Symposium on Computer Architecture*, page To Appear, June 2009.

[17] J. Rabaey, A. Chandrakasan, and B. Nikolic. *Digital Integrated Circuits, 2nd Edition*. Prentice Hall, 2003.

[18] B. F. Romanescu and D. J. Sorin. Core cannibalization architecture: Improving lifetime chip performance for multicore processor in the presence of hard faults. In *Proc. of the 17th International Conference on Parallel Architectures and Compilation Techniques*, 2008.

[19] S. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas. Varius: A model of process variation and resulting timing errors for microarchitects. In *IEEE Transactions on Semiconductor Manufacturing*, pages 3–13, Feb. 2008.

[20] P. Shivakumar, S. Keckler, C. Moore, and D. Burger. Exploiting microarchitectural redundancy for defect tolerance. In *Proc. of the 2003 International Conference on Computer Design*, page 481, Oct. 2003.

[21] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. Exploiting structural duplication for lifetime reliability enhancement. In *Proc. of the 32nd Annual International Symposium on Computer Architecture*, pages 520–531, June 2005.

[22] A. Tiwari and J. Torrellas. Facelift: Hiding and slowing down aging in multicores. In *Proc. of the 41st Annual International Symposium on Microarchitecture*, pages 129–140, Dec. 2008.

[23] M. Vachharajani, N. Vachharajani, D. A. Penry, J. A. Blome, S. Malik, and D. I. August. The liberty simulation environment: A deliberate approach to high-level system modeling. *ACM Transactions on Computer Systems*, 24(3):211–249, 2006.

[24] K. Wang and C.-K. Wu. Design and implementation of fault-tolerant and cost effective crossbar switches for multiprocessor systems. *IEE Proceedings on Computers and Digital Techniques*, 146(1):50–56, Jan. 1999.