Configurable Isolation: Building High Availability Systems with Commodity Multi-Core Processors

Nidhi Aggarwal
Computer Sciences Dept,
University of Wisconsin-Madison
naggarwal@wisc.edu

Parthasarathy Ranganathan
Hewlett Packard Labs,
Palo Alto, California
partha.ranganathan@hp.com

Norman P. Jouppi
Hewlett Packard Labs,
Palo Alto, California
norm.jouppi@hp.com

James E. Smith
Dept. of Electrical and Computer Engineering,
University of Wisconsin-Madison
jes@ece.wisc.edu

ABSTRACT
High availability is an increasingly important requirement for enterprise systems, often valued more than performance. Systems designed for high availability typically use redundant hardware for error detection and continued uptime in the event of a failure. Chip multiprocessors with an abundance of identical resources like cores, cache and interconnection networks would appear to be ideal building blocks for implementing high availability solutions on chip. However, doing so poses significant challenges with respect to error containment and faulty component replacement. Increasing silicon and transient fault rates with future technology scaling exacerbate the problem. This paper proposes a novel, cost-effective, architecture for high availability systems built from future multi-core processors. We propose a new chip multiprocessor architecture that provides *configurable isolation* for fault containment and component retirement, based upon cost-effective modifications to commodity designs. The design is evaluated for a state-of-the-art industrial fault model and the proposed architecture is shown to provide effective fault isolation and graceful degradation even when the failure rate is high.

Categories and Subject Descriptors
C.4 [**Performance of Systems**]: Reliability, Availability and Serviceability of Systems

General Terms
Performance, Design, Reliability

Keywords
High Availability, Fault Isolation, Chip Multiprocessors

1. INTRODUCTION
High availability is a critical feature of enterprise computer systems, especially given the close ties between information technology infrastructure and overall business processes (e.g., e-commerce, business-to-business and customer relationship portals). For example, in a 2001 survey, a quarter of the respondents estimated the costs from server outage to be more than $250,000 per hour, and 8% estimated them as more than $1 million per hour [10]. Others have similarly estimated downtime costs as high as $6 million per hour for availability-critical systems [26]. Consequently, high availability is emerging as the top requirement for enterprise systems and it is often valued more than performance [15].

Current systems offering very high availability, such as the IBM z-series [11] and the HP NonStop systems [4], provide coverage for both permanent and transient hardware faults through a combination of redundant processor hardware and error correcting codes in memories. Redundant processor hardware, employing dual (or triple) modular redundancy – DMR (TMR) – can be applied at different granularities. In current systems this ranges from duplicated pipelines on the same die as in the IBM z-series to mirroring complete processors as in the HP NonStop systems. Redundant hardware not only detects the presence of faults, thereby preventing costly errors and system failure, it also allows applications to continue executing, without downtime, until faulty component(s) can be replaced.

The impending widespread usage of chip multiprocessors (CMPs) – containing multiple processor cores, caches, and interconnection networks – would appear to be an excellent match for high availability systems incorporating redundant hardware. Using a CMP as the basic building block for future high availability systems has the benefit that the entire high availability solution can be implemented on a single chip (e.g., an 8-core CMP can be configured as a cluster of four high availability processors with DMR). However, architecting these systems poses significant challenges with respect to error containment and core-level reconfiguration. For example, in a chip architecture that shares a memory controller across multiple cores, a single fault in the memory controller can result in correlated, undetected errors, and system failure. Or, even if the fault is detected, none of the cores will be able to perform useful computation, requiring the replacement of the entire multi-core chip (including all of the functioning as well as non-functioning components).

It is our objective to study the architecture of high availability systems employing commodity CMPs, in particular:

- We propose a new chip multiprocessor architecture that provides low-level isolation for fault containment and reconfiguration through cost-effective modifications to

ISCA'07, June 9–13, 2007, San Diego, California, USA.

commodity designs. Specifically, the proposed architecture introduces a minimal amount of hardware support for dynamic repartitioning of CMP hardware into multiple fault zones. At the system level, the fault zones are leveraged to provide fault detection and reuse of system resources when a single on-chip component suffers a fault. We also demonstrate a new optimization that dynamically re-assigns the power budget of failed CMP components to the remaining components. The re-allocated power enables voltage/frequency up-scaling of remaining cores so that performance loss due to failed component(s) is mitigated.

- We perform an exhaustive simulation of the system design space using detailed fault models obtained from high availability system vendors. Results show that the proposed architecture is superior to alternative approaches and provides graceful performance degradation while using general-purpose commodity components. We also perform a detailed sensitivity analysis for various failure rates and show that benefits of the proposed approach become increasingly important with future technologies.

2. MOTIVATION AND BACKGROUND

2.1 Future Technology: A Two-Edged Sword

Continuing trends toward aggressive scaling and greater transistor densities enable CMPs with increasing numbers of cores, but with reduced reliability. It is the combination of high density CMPs and reduced reliability that motivates the research reported here.

From an architecture perspective, increased transistor densities are leading virtually all the major companies (Intel, AMD, Sun, and IBM) toward CMPs. Scaling of CMPs to more cores allows greater computational capability and system integration at the chip level, enabling cost and performance benefits from reduced component counts and enhanced resource sharing. The Sun Niagara chip, for example, includes eight cores, with a shared L2 cache, and integrated memory controllers and I/O interfaces.

With respect to reliability, the International Technology Roadmap for Semiconductors has predicted significant reliability problems for future systems, increasing at a pace that has not been seen in the past [36]. Several studies have shown dramatic increases in the number of hardware errors with technology scaling. Permanent or intermittent hardware faults, caused by defects in the silicon and wear out over time, lead to "hard errors". Transient faults which change random bits due to electrical noise or external radiation (e.g., alpha radiation from impurities or gamma radiation from outside) lead to "soft errors" [8]. For example, Srinivasan et. al [29] showed a three-fold increase in processor wear out related faults when scaling from 180nm to 65nm. Similarly, Borkar [5] estimates a 100-fold increase in transient faults when scaling from 180nm to 16nm; Shivakumar et. al [24] predict an even higher *nine-orders-of-magnitude* increase in logic circuits' transient fault rates from 1992 to 2011. The increased likelihood of soft errors now means that we can potentially have multi-bit fault modes at the memory [9]. Even more importantly, with the typical doubling of logic state bits every generation, transient faults that cause arbitrary logic errors cannot be ignored any longer [5]. This means that more sophisticated techniques to detect soft errors, including redundant execution to verify logic at every level, are required even for low-end systems.

2.2 CMP Based High Availability Systems

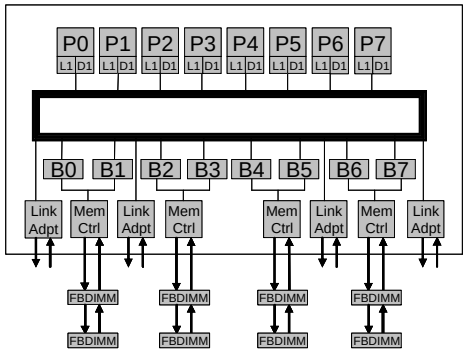
The multiple cores in a CMP provide the basic ingredients for building highly available systems that mitigate the effects of reduced transistor level reliability. To realize this potential, however, there are a number of architecture and engineering obstacles that must be overcome. First, high availability CMP-based system architectures that deal with future technology fault modes must be developed. These architectures should not only enable fault detection, but they should also provide fault isolation at relatively small granularities, so that an entire chip is not disabled by a single fault. Developing high availability system architectures is probably not enough, however. There is also a practical engineering issue: at least in the near term, relatively few systems will employ high availability DMR/TMR solutions. Consequently, high availability enhancements to CMP architectures must be inexpensive and non-intrusive with respect to other CMP features.

In the past, when the basic system building blocks were individual processors, memory controllers, and cache memory SRAMs, system designers could achieve good fault isolation by combining these chip-level building blocks into redundant configurations at the board level. And, when necessary, small amounts of "glue logic" could be incorporated. In these systems, fault isolation to a single chip (or socket) was adequate. For example, the NonStop Advanced Architecture implements process pairs and fault containment boundaries at the socket level. With CMP-based approaches, however, socket level isolation is no longer an attractive solution (nor is the use of off-chip glue logic) especially for small systems. The challenge is to design CMPs that use on-chip mechanisms to enable either conventional, non-redundant systems or high availability systems with good fault isolation characteristics. Therefore, in our research we are focusing on techniques that will enable "off-the-shelf" CMPs to be configured, with relatively little added on-chip hardware and complexity, into high availability, redundant systems.

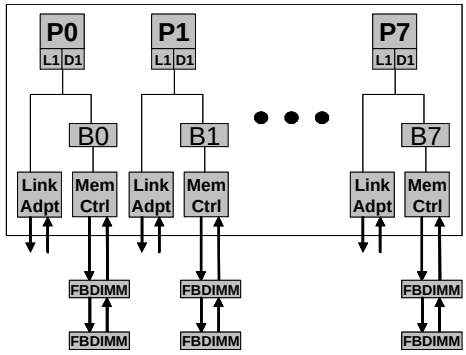
Figure 1(a) shows a conventional CMP architecture with 8 cores (P0...P7) each with private L1 caches, an 8-way banked shared L2 cache, 4 memory controllers, and coherent links (such as Hypertransport) to other sockets or I/O hubs. In this architecture, a bidirectional ring connects the processors and cache banks. (Our techniques also apply to more complex 2-D arrangements such as meshes). Although a "dance hall" layout is shown for simplicity (with all the cores on one side of the ring and the shared cache banks on the other), the proposed techniques apply equally well to interleaved layouts. This design with fully shared resources is the conventional baseline used in the remainder of this paper; it is also referred to as the *shared design.*

As the number of cores in a CMP increase geometrically with lithographic scaling, a failure in one part of the conventional organization affects larger and larger amounts of computational capability. For example, if all L2 cache banks are shared, and addresses are low-order interleaved among the banks, a transient fault in the cache controller state machine can lead to erroneous

coherence state. Note that ECC on a coherence bit does not help in this case because the fault is in the cache controller logic, before the coherence bits are set. Such a fault affects the reliability of an entire socket. Similarly, a fault in a memory controller or anywhere in the ring interconnect affects all the cores. This is true even if programs are being run in a Dual Modular Redundant (DMR) or Triple Modular Redundant (TMR) configuration. For cost-effective high availability systems, we want architectures where the effects of faults are isolated to much smaller areas.



(a) Baseline conventional system architecture



(b) A design with full isolation

Figure 1. CMP architecture designs.

One alternative to the conventional architecture is a design with independent computers fabricated on the same die (Figure 1(b)). Each computer has its own memory controller and I/O connections. This architecture has a number of disadvantages, however. Hard partitioning of cache resources (inhibiting any sharing) significantly reduces overall system performance and is not been used in proposed CMP designs [53, 54, 55]. Similarly, by partitioning chip interfaces and pins, off-chip bandwidth is inefficiently used. These performance inefficiencies make such a design unattractive for high volume applications where performance is the objective rather than very high availability. This design provides full isolation, however, and is referred to as *fully isolated* or *private* design in the remainder of the paper.

In this paper, we explore the tradeoff between full isolation and full sharing. We propose non-intrusive enhancements to commodity CMP architectures, which provide configurable isolation for providing varying levels of availability.

2.3 Related Work

Gold et al. [13] classify error handling according to how far outside the processor core errors can propagate. The categories are (1) core-level containment, (2) cache-level containment, and (3) memory-level containment. Our work focuses on memory-level containment where processors and memory are grouped into a fault zone (or fault containment boundary). Consequently only operations that cause device accesses (I/O) require detection and recovery. The closest related approach is the HP NonStop Advanced Architecture (NSAA) [4]. However, as discussed earlier, the NSAA approach implements process pairs and supports fault containment boundaries at the socket level. In contrast, we consider the implications of memory-level fault containment at the chip multiprocessor level. The SafetyNet [28] and ReVive [18] projects both consider the challenges of, and solutions for, global checkpointing at the memory-level. These solutions are orthogonal to our work, and their methods can be used in conjunction with the fault detection and isolation architecture that we propose.

At the core-containment level, the IBM z-series processor [26] uses an aggressive custom design employing redundant pipelines to create separate fault zones and detect failures. Academic work has proposed variants of such integrated checking at the processor level [2, 19, 21, 27]. Several studies have also evaluated core-level fault detection and containment by running redundant processes, either on a separate thread [22, 23, 33] or on a separate core [14, 17, 32]. Solutions using core-containment methods rely on other fault tolerant methods for the rest of the system (ECC, RAID-M, etc). At the cache-containment level, the original Tandem NonStop architecture [3] uses a lock-stepped process pair's results, compared at the front-side bus. The TRUSS project [12] proposes an alternate form of cache-level fault containment for shared-memory multiprocessors by performing error detection and checkpointing at the granularity of cache coherence events. These approaches typically require custom changes to the processor or cache controller. Moreover, they assume fine grained and aggressive fault containment at every level of the system. In contrast, memory-level fault containment approaches like ours are typically less expensive, by virtue of performing fault tolerance at a coarser level. Our work is also different in its approach to leveraging commodity multi-core processors with little additional on-chip support for providing high-levels of availability and fault containment.

Russ Joseph discusses an approach to "salvage" processors in multi-core architectures using a virtualization layer to emulate lost functionality [16]. Other studies [6, 25, 31] have explored the design of redundancy and self-repair at the hardware level to make up for lost functionality in the event of a fault. These methods need additional hardware overhead at every structure where faults occur, and furthermore, they do not address fault isolation and issues regarding the effects of single faults on multiple cores.

2.4 Paper Overview

The rest of the paper is organized as follows. Section 3 discusses the proposed architecture and the tradeoffs in detail. Sections 4 and 5 discuss evaluation methodology and results. In

section 6, we discuss future work and other implications of the proposed research and conclude the paper.

3. PROPOSED ARCHITECTURE

A block diagram of the proposed CMP organization is in Figure 2. In the simplest arrangement, the resources of the CMP are split into two groups represented by two colors, say for example, red and green (in a black and white print, these appear as black and gray in Figure 2). The CMP is configured so that the colored domains are units of fault containment. Any failure in a color-shared component affects computation only on the cores mapped to that color. To ensure that a failure in one color domain does not affect all the other colored domains, we propose "*configurable isolation*" for interconnect, caches and memory controllers. We define configurable isolation to be a set of *techniques that provide optional logical fault isolation for shared components.*

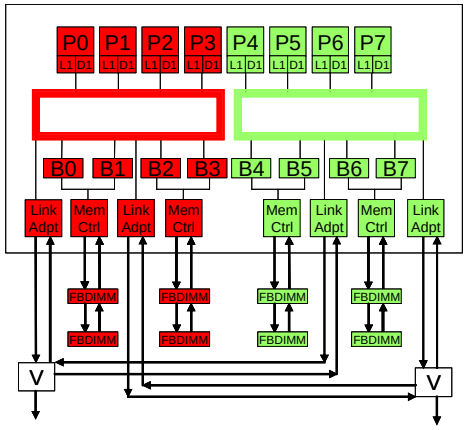


Figure 2. Proposed architecture with configurable isolation split into two domains.

The proposed architecture can be used in a number of configurations. Below, we discuss a NonStop-like DMR configuration. We choose this configuration as an illustrative example of both the benefits of our approach – (1) fault isolation without losing the benefits of sharing and (2) graceful degradation via reconfiguration in the event of a hard fault. Furthermore, fault recovery is more difficult in a DMR configuration as compared to TMR. Resources from two colors are used to run a DMR process pair, with computations in the red domain replicated in the green domain when higher availability is required. The microprocessor is dynamically reconfigured to the higher-availability configuration by setting a small number of control points in the design. This allows systems to support "high availability on demand."

Note that this capability only requires small changes to the ring and bank addressing, while the rest of the CMP is unchanged. The ring interconnect in Figure 2 has been logically re-routed via configuration cross links to create two independent rings. Physically, the ring is expected to form the central spine of the chip, so the cross links should be less than a millimeter long and their activation requires the insertion of a multiplexer at the input of a ring interface incoming data port. Thus, cross links and input multiplexers are a small additional fixed cost in terms of area and power, which does not significantly increase the cost of the design for system configurations where higher availability is not an objective. The cross-links are also expected to be shorter than the ring segments between cores, so the cross-connects should operate at least as fast as a core-to-core or bank-to-bank ring segment. Because the cross links and input multiplexers are shared and they can now form a single point of failure, they need to be implemented using self checked logic to satisfy stringent availability requirements. Self checked logic in these components allows dynamic reconfiguration and reassignment of colors after failures. If these components are not self checked then the color assignment is static and can not be changed.

When the inter-core interconnect is partitioned, interleaving among L2 cache banks uses one fewer address bit, thereby interleaving references among half the banks, and all references are kept within the same color. Therefore, the L2 cache size available in a color is inversely proportional to the number of colors.

In high availability configurations, self checked voters compare the output of the redundant execution to detect errors. In our proposed architecture these voters can be implemented in a number of ways. For highest availability, voters can be implemented in I/O hubs connected to a red and green link adapter, similar to the hardware voters in the Nonstop Advanced Architecture [4]. For lower-cost lower availability solutions, the voter can be implemented in hypervisors that communicate between the colored partitions through I/O [7].

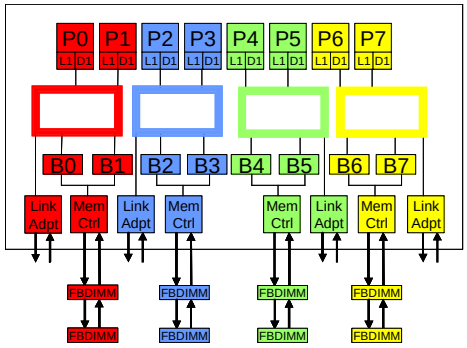
The proposed architecture is developed from a system perspective. Similar to the NonStop system, physical memory is partitioned between the logical processors using virtual to physical memory mapping. Redundant TLBs are used for fault tolerance. Further, OS support for statically partitioned TLB entries provides complete memory fault isolation.

When a fault is detected in the proposed system, reconfiguration takes place. Cores that fail in one color domain are deleted (retired from use), but the remaining cores are still usable. If a failure in a cache bank cannot be corrected by line sparing (e.g., a logic failure in the bank controller), the other bank sharing its memory controller can be reconfigured to cache all lines serviced by its memory controller. This requires the provision of an extra bit in the bank cache tags and a mode bit in the cache bank. Similarly, if a memory controller in a color domain fails, the domain can be reconfigured to use a single memory controller. This requires caching all lines in the cache banks associated with the failed memory controller in the remaining controller's banks. Doing so adds one bit in the cache tags and a second mode bit. Given the large number of bits in a cache line (more than 600 bits for a 64B cache line with ECC plus previously required tag bits), providing two more bits to enable reconfiguration is a very modest overhead [1, 20]. Overall, the number of extra bits required in a bank to enable caching of lines from any other bank is log_2 *(number of banks)*.

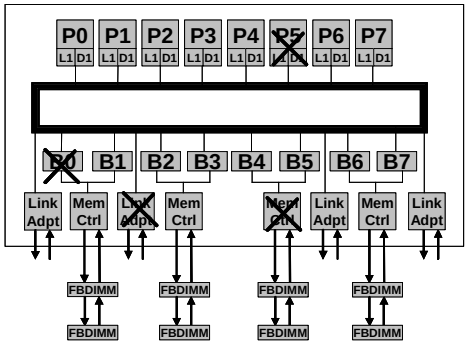
Reconfiguration may be limited by the topology of the system. For example, due to the placement of the ring cross-connect, if two cores fail in one color domain while all the cores are fault-free in the other domain, it may not be possible to reassign a core from one color to another. Thus in the 8-core 2-color system, if two red cores fail, only two DMR process pairs can be

supported by the CMP. However, the number of colors is not restricted to two and can be increased to provide more redundancy or smaller granularity of fault containment. For example, three colors can be used to enable a Triple Modular Redundant (TMR) configuration. Furthermore, the number of colors need not be static and can be changed as the system needs evolve.

Figure 3(a) shows a configuration with four colors. This configuration requires four ring cross connects and uses two fewer address bits for interleaving cache bank addresses. This configuration runs processes in TMR using triples of three different colors. For example, one process can be run on red, green, and blue, simultaneously another TMR process can be run on green, blue, and yellow and DMR processing can be supported on red and yellow to balance the load among colors.



(a) Design with 3 reconfigurable cross links supporting four fault isolation domains for use in TMR.



(b) Design illustrating use of reconfiguration to isolate specific faulty components.

Figure 3. More architectures with configurable isolation and reconfiguration.

It should be noted that the reconfiguration and fault isolation proposed in our architecture can be used independently of DMR/TMR-coloring to provide repartitioning of resources in the event of a hard fault. This is illustrated in Figure 3(b) for applications with more modest availability requirements. If there is a core fault (P5 in the figure) then system software isolates the faulty core and continues functioning with the remaining cores. Furthermore, the proposed architecture has the ability to continue functioning in the event of hard faults in cache banks, memory controllers and interconnect. For example, if there is a fault in a cache bank (B0 in the figure) then all the lines in that bank can be cached in the bank that shares the memory controller with the faulty bank (in this case B1). In case of a memory controller fault, lines cached by both the banks (B4 and B5) connected with the memory controller can be cached by two other banks connected to a fault free memory controller (B6 and B7). Similarly, link adapter and interconnect failures can be tolerated by isolation of the faulty components and reconfiguration to use the remaining fault free components.

3.1 Example: A NonStop-like DMR system

In this section we briefly explain how the proposed architecture can be deployed in a high availability configuration similar to a NonStop DMR system. The NonStop software stack includes the NonStop kernel and critical software implemented as process pairs. Using the NonStop terminology, a single logical processor is implemented with two hardware processing elements (PEs). So, an eight core CMP (eight PEs) implements four logical processors. A logical processor runs a redundant process pair. In a DMR configuration of the proposed architecture the OS schedules each process of a process pair to a different color. Each PE of a logical processor has the same virtual-to-physical mapping and takes page faults and interrupts at the same points in the instruction stream – supported by hardware in the voter logic and the OS. The cache and TLB state across the two colors can be different, but each PE writes to memory so that on any output operation (say to disk) the data pulled from either of the colors should be the same. Consequently, result comparison is done in the I/O system. The output operations of each PE in a logical processor are compared by a fully self-checked voter in the I/O hub. Therefore, in an eight core CMP, four voters compare the output operations from the four logical processors.

Coverage for soft errors is provided for all the circuitry on chip because the voting is done at the I/O level. Voting at the I/O level does not lead to miscomparison if a transient fault does not cause an error in the I/O output. In case of a miscomparison, execution is restarted with the help of software supported checkpoints on both PEs in a logical processor. If the error persists, then the two cores (PEs) are halted and the process pair is restarted from a checkpoint on remaining, fault-free cores.

In a DMR configuration a voter miscompare can be ambiguous regarding which PE is at fault. However, as pointed out in the NonStop description [4] some hard faults, for example, interconnect faults and functional unit faults, are self identifying. Heuristics similar to a probation vector used by NonStop in a DMR configuration can also help disambiguate voter miscomparisons. A bit in the probation vector is set when a core suffers a number of correctable errors. In case of an ambiguous miscomparison the core that has its probation vector set is assumed to be the one with the fault. Furthermore, localized component diagnostic tests can identify a faulty component. Once the faulty hardware structure is diagnosed, the chip is reconfigured (similar to hardware reconfiguration example explained above) to allow further operation. In such a case, the proposed architecture provides effective fault isolation and allows continued operation with fault free components. However, if the error can not be identified then the logical processor is halted and the OS will invoke failover to other logical processors and the application can continue to run.

4. EVALUATION METHODOLOGY

During the lifetime of a fault tolerant system, as hard faults occur and are detected, system reconfiguration degrades the system's computing capacity. Consequently, we perform lifetime simulations targeted at measuring computing capability as a function of time. We compare three fault tolerant architectures: 1) ***shared*** -- a completely shared system similar to proposed CMPs (Figure 1a), 2) ***fully isolated*** -- a completely private system with full isolation (Figure 1b), and 3) ***configurable isolation*** – the proposed architecture with reconfiguration and configurable isolation (Figure 2). All three are assumed to be in a DMR configuration.

4.1 Workloads

Because the proposed architecture does not contain any modification to the cores, the most important workload characteristic is the size of the working set and its effect on cache behavior. We tried to cover a spectrum of workloads and considered the following factors while choosing the benchmarks for simulated workloads – 1) the combined working set size of the workload, 2) stability of the benchmark – the chosen benchmarks typically grow quickly to their target size and stay there, and 3) the ability of the benchmarks in the workload to exercise the L2 cache. Consequently, we constructed three different workloads from the SPEC benchmarks. The workloads have large, mixed, and small memory requirements as measured by memory footprint [35] and are listed in Table 2. The large and small memory workloads contain benchmarks all having large and small memory requirements, respectively. The mixed workload has benchmarks with large, medium, and small memory footprints. When simulated, each benchmark is duplicated to mimic the DMR configuration and is run to completion.

Table 1: Benchmarks studied and their memory footprints.

High Memory	gap (192), apsi (191), swim (191), mcf (190)
Mixed Memory	applu (181), perlbmk (146), fma3d (103), crafty (2.0)
Low Memory	vortex (72), equake (49), facerec (16), mesa (9.4)

Over the course of a simulation run, as cores become unusable (either because of a core fault or a fault in a component that is required by the core), benchmarks are dropped from the workloads, reflecting the loss of computing capability. This is done only for our evaluation; in a deployed configuration, a program would not be dropped but would be failed-over to one of the other cores. At any point, the dropped benchmark is the one having the median-sized footprint. For example, when evaluating the small memory workload after two core failures, the workload becomes equake, vortex, and mesa. The goal is to maintain the overall memory characteristic of the workload.

4.2 Fault and Failure Modeling

The fault-models are based on state of the art technology and are derived from detailed (confidential) models from processor vendors and were calibrated by HP-internal fault analysis experiments. The fault data includes failures in time (FIT) rates and distributions for hard and soft errors per component. The fault data has been de-rated both at the chip level and system level for various architecture optimizations that help mitigate the effect of faults. For example, the fault model assumes that 90% of cache related errors are dynamically correctable through chip-kill, ECC, cache line sparing, etc. The model also uses separate detailed distribution models for hard and soft errors. For hard errors the distribution is Weibull with a decreasing hazard rate. For soft errors the distribution is exponential. (This is in contrast to some prior work which assumes that all failure distributions are poisson or log normal [30, 31].)

The system is divided into five different fault zones (which also represent the granularity of reconfigurations). These are: core and L1 cache, L2 circuitry, L2 banks, memory controller circuitry, and link controller. To prune the simulation space, we limit the study to hard faults that require system reconfiguration – for example we do not model the performance impact from faults that lead to minor degradations such as cache line deletions. Consequently, reconfiguration is done only at full component granularity.

On the shared system, any hard fault leads to system failure. This means that after a failure, the system throughput goes to zero for all workloads. On the fully isolated system, any single fault leads to the loss of throughput from a DMR process pair. For example, even a fault in the bank associated with a core would lead to that core being unusable. On a system with configurable isolation, a reconfigurable fault (for example, in a memory controller) leads to loss of performance over all the process pairs, but not the loss of a workload. Only when a core fails is a benchmark (both copies) dropped from the workload. The entire system becomes unusable in the configurable isolated architecture only when the penultimate component of any type fails (for example, 7^{th} core, 3^{rd} memory controller, 7^{th} cache bank in a system similar to the baseline).

The simulation model assumes that in all the cases soft errors are detected by DMR execution and corrected through checkpointing or re-execution; but given that this consumes relatively little time, this is not shown in the results. Note that in the fully shared case, given the lack of fault isolation, it is not guaranteed that all soft errors will be detected; however, we assume that such cases are rare and do not penalize this configuration.

4.3 Simulation Methodology

In order to make a large number of very long time scale simulations feasible we use a two phase simulation methodology. First, we use a full system simulator to exhaustively simulate the possible system configurations (using more than one machine year) and compute the throughput of all configurations, subject to policies described below. Second, we perform Monte Carlo simulations using a detailed component-level fault model. With Monte Carlo simulation we simulate fault injection in a total of 10,000 systems with each run comprising 100,000 simulated hours (approximately 11 simulated years).

For the simulated configurations and failure rates, the system typically becomes unusable in a fault tolerant configuration in 100,000 hours. At every fault instance the system is reconfigured to map out the faulty component. The Monte Carlo simulator then looks up the aggregate throughput of the new configuration (from the first phase detailed simulation results), and assigns it as the system throughput until the next fault and subsequent reconfiguration occur. This methodology is similar to the one used by Srinivasan et al. [30]. The key difference is that we model the lifetime of a complete system

along with its transitions through the configuration space and model the potential throughput on a yearly scale. We include the overheads of coloring to the performance of the proposed configurable isolation architecture. In the DMR configuration each color has access to only half the cache and so we model the performance assuming two L2 caches; each half the size of the cache in the shared configuration.

All simulations were done using a full system x86/x86-64 simulator (based on AMD SimNow™) [37] that can boot an unmodified Windows or Linux OS and execute complex application programs. The simulator has been validated by comparing to real hardware. The simulator guest runs a 64-bit Ubuntu Linux distribution with 2.6.15 kernel. The SPEC benchmarks were compiled directly in the simulated machine with gcc/g77 version 4.0, at -O3 optimization level. To evaluate only workload execution, we restore a snapshot of the system taken just after booting when the machine is idle (except for standard OS housekeeping tasks) and directly invoke the execution of the benchmark from a Linux shell. The timing simulation begins just after the execution command is typed in the OS console. Falcon et al. provide more details about the simulator [37].

Table 2: Simulation parameters.

Instruction set	x86_64
Core clock	2200MHz
Bus clock	550MHz
Issue width	4
2-level branch predictor (g-share)	16384 entries
L1 icache size	32kB
L1 icache line_size	64B
L1 icache associativity	2-way
L1 dcache size	32kB
L1 dcache line_size	64B
L1 dcache associativity	2-way
L1 itlb entries	32
L1 itlb associativity	8
L1 dtlb entries	32
L1 dtlb associativity	8
L2 cache size	2MB (initial)
L2 cache line_size	64B
L2 cache associativity	4-way
L2 dtlb entries	512
L2 dtlb associativity	4
Coherency protocol	MOESI, shared bus
Main memory access	220 cycles

We used a SMARTS-like [34] functional timing sampling of 200:1, where for every 3s (6.6B instructions) of simulation there is a warming phase of 15ms (33M instructions), a detailed timing phase of 15ms (33M instructions), and a fast functional emulation phase of 2.7s (6,534M instructions). The long warming phase is necessary to ensure that large L2 cache structures contain meaningful data for the timing phase.

We use a timing model with a memory hierarchy roughly similar to that supported by an AMD Opteron 280 processor. The only exception is that the L2 caches are smaller than currently found in real systems. As explained in Section 2, the changes to support cache reconfiguration lead to smaller caches in the DMR configuration. In order to avoid giving an unfair advantage to configurable isolation, we used smaller caches that match the working set requirements of the benchmarks, thereby exposing the difference in performance when the cache size is halved due to coloring. Our results hold qualitatively for larger cache sizes which we validated by simulating several configurations with an initial 8 MB L2 cache. The detailed simulator configuration is in Table 2.

5. RESULTS

To evaluate the proposed architecture, in Section 5.1 we consider an application of configurable isolation for systems that provide full fault coverage using DMR for both hard fault and transient faults. Section 5.2 considers the applicability of the proposed architecture in an environment where graceful degradation is more important: a server farm with lower-availability systems that provide coverage only for hard faults. Section 5.3 presents results showing sensitivity to future trends and per-component fault variations. Section 5.4 presents results showing improvements with dynamic re-provisioning of power.

5.1 High availability Systems with Soft Error and Hard Error Coverage

Figures 4(a), (b), and (c) summarize the baseline results for the small, mixed, and large workloads respectively. All these experiments employ averaged results from 10,000-run Monte Carlo simulations of an individual system. Each system is a server with 8 cores, 8 L2 banks and 4 memory controllers and similar to the baseline. Each graph has three curves representing (1) a baseline system using traditional full-resource sharing, (2) a system with full isolation, and (3) the proposed system with configurable isolation. In each of the graphs, the y-axis is the mean cumulative performance normalized to the baseline performance of the shared configuration with no faults. The x-axis is time, measured in years.

Referring to Figure 4, as expected, the shared system performs the worst, with a dramatic degradation (30-35%) in average performance during the first two years, and a degradation of close to 50% by the end of five years. The fully-isolated configuration is much more resilient to failures and provides more gradual performance degradation. Over five years, the net performance loss is only 10-15%. The results for the large memory workload (Figure 4c) are particularly interesting. Here, the completely isolated configurations, by virtue of having private caches, initially under-perform the shared configuration. However, compared to the shared system, the fully-isolated system becomes performance competitive at around 2 years (at the crossover of the curves in Figure 4c).

The configurable isolation system consistently achieves the best performance across all the workloads. Figure 4 illustrates that it achieves effectively 100% fault detection without compromising benefits from sharing. Compared to the fully isolated configuration that underperforms the fully shared for a full two years, the proposed architecture breaks even in the middle of the first year. With configurable isolation, resources can still be shared within a given fault zone. Additionally, the ability to dynamically repartition the resources leads to the best graceful degradation across all three workloads.

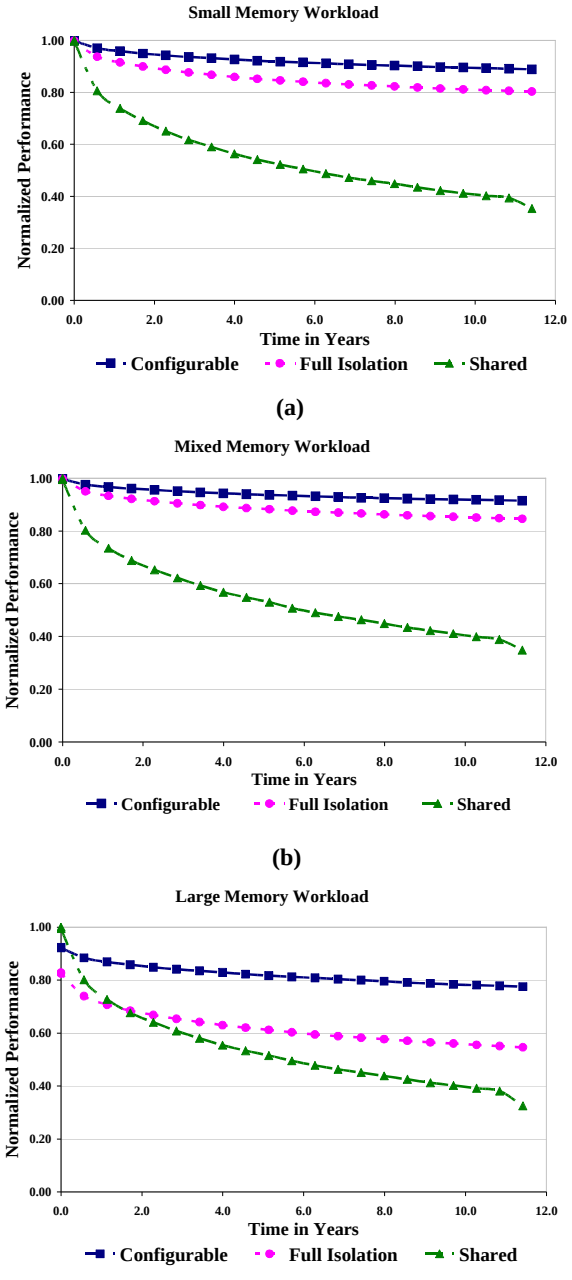


Figure 4. Benefits from our architecture. *The results from a Monte-Carlo simulation of faults for normalized performance over an 11-year period of time for three architectures – a baseline conventional solution with full sharing, our proposed solution, with configurable isolation, and a solution with full isolation for a) small memory workload, b) mixed memory workload and c) large memory workload.*

It is interesting to consider the reconfiguration benefits as compared with full isolation. In the case where one component fails, for example a bank controller, the entire core associated with that bank is taken out of commission. In contrast, in the configurable isolation case, the core can be reconfigured to use other banks in the same color. As discussed earlier, these benefits are obtained with very little area overhead in conventional commodity systems and provide the option of being disabled if unneeded.

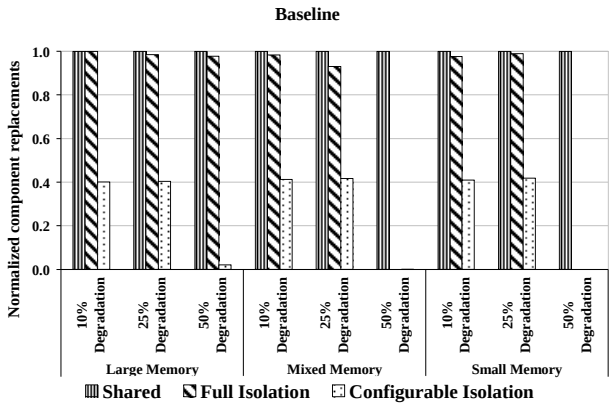


Figure 5. Number of component replacements as a function of performance. *The figure shows the normalized number of component replacements for the three architectures (mixed workload), compared assuming components are replaced (a) when performance dips below 90%,(b) when performance dips below 75%, and (c) when performance dips below 50%.*

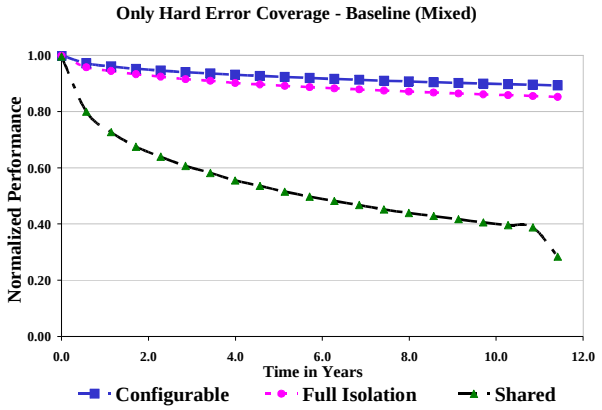
Figure 5 provides an alternate view of the benefits of configurable isolation. For each of the three approaches, the number of component replacements is shown. It is assumed that the system continues to stay operational until the performance dips below a certain threshold after which the entire multi-core component is replaced (and the performance is re-initialized to that of the no-fault configuration). The simulation then continues for the remainder of the 100,000 hours. Note that there might be multiple replacements in a single Monte Carlo run simulating 100,000 hours. We consider three cases where the performance threshold is set to (a) 90% (b) 75%, and (c) 50% of initial performance. The total number of replacements (across 10000 Monte Carlo runs) for fully isolated and configurable isolated system is normalized with respect to the total number of replacements for a fully shared system. In a fully shared system every fault leads to system replacement because the performance drops to 0. These results show that the architecture with configurable isolation dramatically reduces the need to replace components across all three workloads irrespective of the performance thresholds.

5.2 Systems with Hard Error Protection

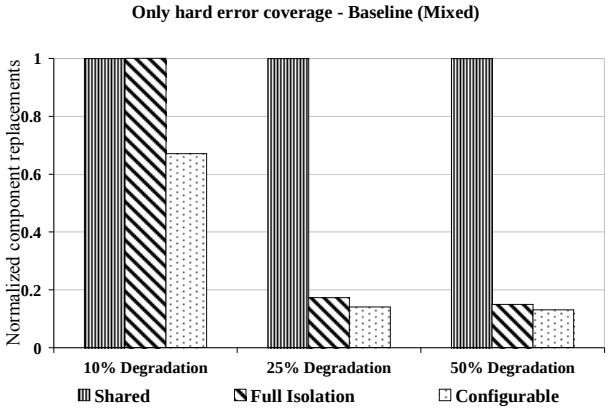
Thus far, we have assumed that configurable isolation is used in conjunction with redundant processes to detect soft errors. Although there are several ways to reconfigure after a core failure, as discussed earlier, in our experiments we conservatively assume that in configurations with an odd number of cores, one core is unused to make sure that there are as many cores as the total number of processes in a DMR configuration. This can potentially affect the slope of the performance degradation curves.

Additionally, there may be many system environments, such as farms of web servers, where coverage for soft errors through redundant processes may be more expensive than the acceptable alternative of just restarting the system when the software

detects an error and halts. In these cases, rather than being able to guarantee fault isolation across redundant processes in a dual modular configuration, the biggest advantage from configurable isolation is the ability to isolate a fault to a specific component, allowing continued use of the rest of the system.



(a) Performance over time



(b) Normalized component replacements

Figure 6. Evaluating a server-farm with only hard-error coverage

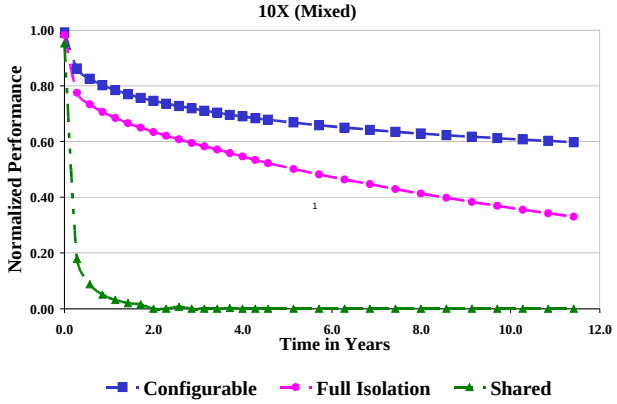
Therefore, we performed experiments on a simulated server farm with each core running a single process. Figure 6 summarizes the results[1]. As shown in Figure 6(a), performance degrades much more gracefully with respect to hard faults. The average performance of the configurable isolation architecture degrades by less than 10% over 10 years. In contrast, the fully shared configuration degrades by almost 60% over the same time period. Also, the difference in performance between the fully isolated and configurable isolated architecture is now smaller as compared to that in Figure 4. In the present configuration, any single fault in a fully isolated architecture leads to throughput loss of a single process, unlike the DMR case where the loss is in pairs. Comparison of the number of component replacements (Figures 5 and 6(b)) show similar trends.

[1] Unless the trends for the "small" and "large" workloads are qualitatively different, we present time-variation results just for the "mixed workload" in the interests of space.

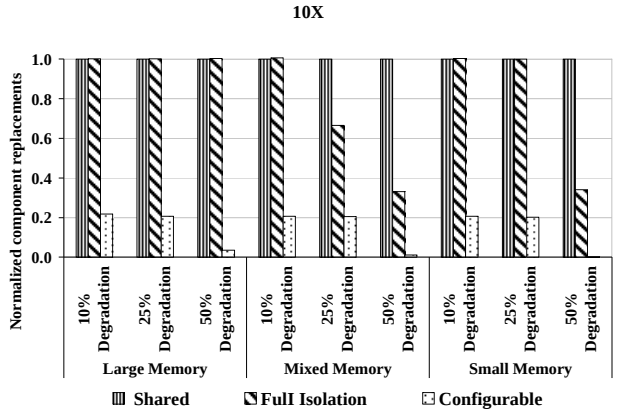
5.3 Sensitivity Study

5.3.1.1 Sensitivity to overall fault rate

As discussed in Section 3, our fault model is based on proprietary industry data. This data, while based on actual measurements and deployment experience, is indicative of current and past technologies. Several studies [6, 29, 24] have hypothesized that the fault rates in the future are likely to be significantly higher. To study higher fault rates, we performed some additional evaluations. Specifically, we increased the industry FIT rates by a factor of ten while keeping the same fault distributions.



(a) Performance over time



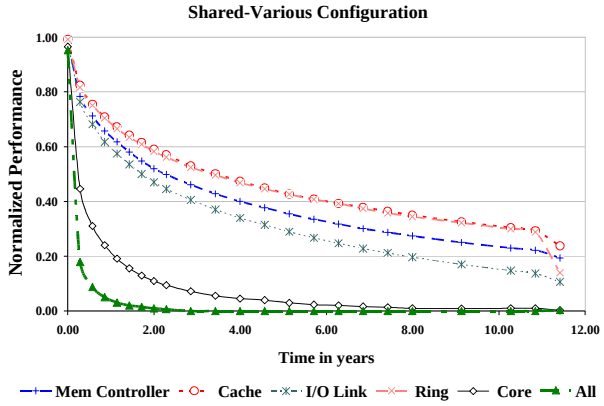
(b) Normalized component replacements

Figure 7. Evaluation with a "future" fault model. *We change the default fault model (that is based on current industry data) to reflect predicted future trends in the number of faults.*

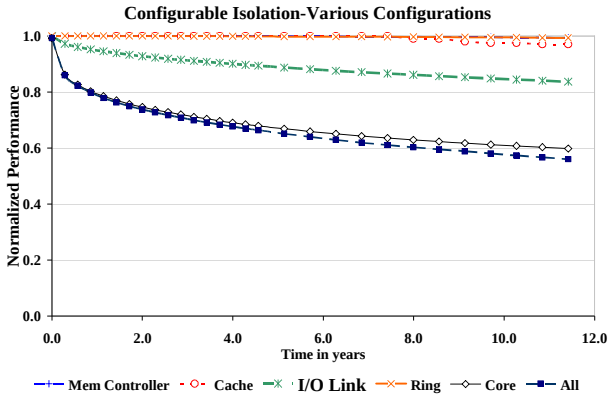
Figure 7 summarizes the results for the DMR-configurations. As shown in the figure, increased fault rates more strongly motivate the need for fault isolation and faulty-component retirement. In particular, the fully-shared system fails completely in less than one year. In contrast, configurable isolation continues to provide 60% of the original performance even after 10 years. In the same time frame, the fully isolated solution provides only 40% of the original performance. The component replacements data show similar trends.

5.3.1.2 *Sensitivity to specific component faults*

Our results thus far have focused on performance degradation over time with an integrated fault model that includes the possibility of failure in any CMP component. However, different system components contribute disproportionately to performance degradation for two reasons. First, the fault rates and distributions differ across components, and second, the failures of different components have varying impact on aggregate system performance. To understand the relative importance of specific components to the system lifetime, we present results on performance degradation with five variations of the fault model that specifically isolate (1) core faults, (2) cache faults, (3) interconnection faults, (4) memory controller faults and (5) I/O link faults.



(a) Shared design



(b) Configurable isolation

Figure 8. Isolating the impact from faults at a component level. *It is assumed that only one component fails at a given time; the results are for the futuristic fault model for the DMR system.*

Figure 8 summarizes results for the fully-shared and configurable isolated configurations with the DMR-based system (the fully-isolated configuration is similar to the configurable isolated configuration). To better illustrate the trends, we assume the more aggressive future fault rates. We plot five curves representing performance degradation when a given component type fails, and for reference, also plot the overall performance degradation. For the fully-shared configuration, as expected, all faults are equally critical. However, for the configurable isolated configuration, the maximum performance impact is from core faults. This reflects the characteristics of the proposed architecture where reconfiguration can address some of the performance degradation for all faults except those in a core; a core fault results in the loss of throughput from a process pair (the next section discusses a novel solution to address this). Our results also underscore the importance of techniques proposed in prior work that focus on intra processor redundancy to help mitigate the effect of faults in a core [31].

5.4 Power Re-provisioning to Offset Performance Loss in Degraded Configurations

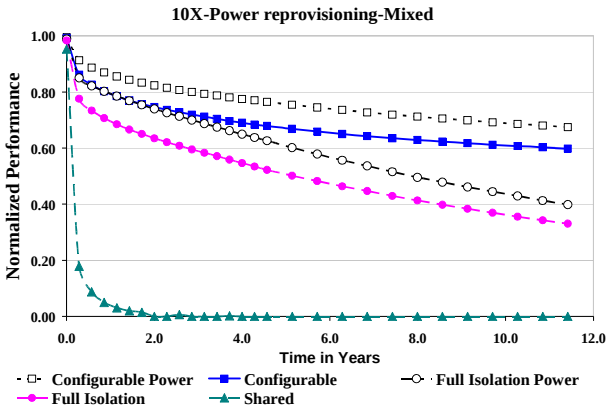
The proposed architecture enables dynamic core partitioning to isolate faults and reuse system resources, albeit with degraded performance. One way to offset this performance loss is to dynamically re-provision the power budget by re-assigning power allotments of the faulty components to the remaining fault-free components.

Specifically, given that the cores are the dominant component of power consumption (and, as seen in Figure 8, they also have the most impact on performance degradation), we focus on core failures. When a core fails, we assume that the power budget of that core can be re-allocated dynamically to increase the clock frequency of the remaining cores. There are two issues to be considered. First, the supply voltage needs to be increased to enable the high frequency[2]. Given that power is proportional to the product of the frequency and the voltage squared, the increased frequency is likely to be limited to the cube root of power available from the faulty core. Second, the thermal packaging of the processor must be able to support the additional localized heat generation. Both these problems can be addressed, however, through judicious design. In particular, for our experiments, we assume that when two cores are retired, the remaining processors' frequency can be scaled up by $8/6^{(1/3)} \approx 10\%$, and when four are retired, the remaining processors' frequency can be increased by $8/4^{(1/3)} \approx 25\%$. When six cores are retired, we conservatively cap the maximum upward frequency scaling to only 25% to avoid thermal overload (although our calculations show that the remaining processors' frequency can be scaled up by 50%).

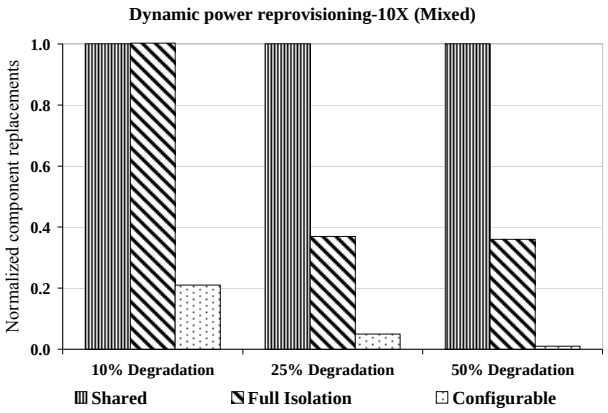
Figure 9 summarizes the results for dynamic power re-provisioning. We focus on the DMR-based system for the future fault model from section 4.3. Overall, we find that re-allocating power and frequency scaling to use the extra power budget does improve system performance by around 10%. Also, as compared to the results from Figure 7 (b), we find that the two configurations that support isolation now degrade much

[2] Recent proposals [38] for multi-core processors have suggested capping power by dynamic power management across multiple cores. This would imply that one could potentially do frequency scaling in the cores without having to worry about increasing the voltage or hitting thermal caps, up to certain limits. In this paper, we assume, conservatively, that this option does not exist, but if it did, one could expect higher performance benefits from the proposed optimization.

more gracefully. For example, at the 25% performance threshold, the number of component replacements declines from 20% to 5%.



(a) Performance over time



(b) Normalized component replacements

Figure 9: Improved performance from dynamically re-provisioning the power budget during failures.

6. SUMMARY AND CONCLUSIONS

Integration of more cores and other components on CMPs allows greater computational capacity and system functionality at the chip level, enabling cost and performance benefits. These same trends, however, pose problems by requiring fault containment and faulty-component replacement at the CMP level, leading to higher costs and increased downtime. This problem is exacerbated by the expected increased fault rates in future technologies.

In this paper, we addressed the conflict between the benefits from system integration and resource sharing and the need for fault isolation and graceful degradation. Specifically, we proposed a novel CMP architecture that is designed to support *configurable isolation*, or optional logical isolation to shared components. We achieved this through intelligent support for reconfiguration that allows processor resources to be *reallocated and partitioned dynamically*, but with very little area overhead.

Using a two-phase simulation methodology, we showed the benefits of the proposed architecture. Fault isolation in the proposed architecture can deal with failures from both permanent faults and transient faults, while still getting most of the benefits from sharing. Additionally, for permanent faults, the reconfiguration support in our architecture allows for graceful degradation even in the face of failure of individual CMP components. The reconfiguration also enables other optimizations including a novel approach that dynamically reprovisions the power budget to further reduce degradation from faults. Our results show that, even for a futuristic fault model, the performance degradation from our architecture is less than 30% over a 10-year period, in contrast to more than 80% with conventional architectures; moreover, the incremental area overhead to enable this solution in a commodity processor is less than 1%.

Our work also opens up an interesting design space of policies - for reconfiguration, fault color distribution, workload-to-partition assignment, and power re-provisioning. Additionally, configurable isolation has implications beyond faults, to performance and security isolation. Furthermore, this architecture can enable tradeoffs between availability and performance, which is a useful characteristic in future utility computing environments. As part of ongoing and future work, we plan to evaluate these in more detail.

In the near future, not all general purpose systems will require such a high degree of protection from faults. Therefore, we designed our architecture to be configurable and evaluated the proposed architecture as it would be deployed in a high availability configuration. Although using these techniques can lead to 100% overhead they provide effectively 100% detection of faults. Furthermore, the 100% overhead stays constant as the fault rates increase in future generations, making this solution viable for continued technology scaling. As fault rates continue to increase in the future, we believe approaches like configurable isolation, needing only small and non-intrusive changes to commodity architectures to enable high availability, are likely to be an integral part of future systems, including possibly in general-purpose desktops and laptops.

7. ACKNOWLEDGMENTS

We are grateful for the feedback of Luiz Barroso and our reviewers. Paolo Faraboschi's help with the simulator and the input from Dave Garcia and George Krejci were invaluable. We would also like to thank Prasun Agarwal, Wendy Bartlett, Daniel Ortega, Kewal Saluja, John Sontag, Bill Tian and Shyam Thoziyoor for their input.

8. REFERENCES

[1] Albonesi, D.H. Selective Cache Ways: On-Demand Cache Resource Allocation. Journal of Instruction-Level Parallelism, Vol. 2, 2000.

[2] Austin, T. M. DIVA: A reliable substrate for deep submicron microarchitecture design. In Proc. of the 32nd Intl. Symposium on Microarchitecture, November 1999.

[3] Bartlett, W. and Ball, B. Tandem's Approach to Fault Tolerance. Tandem Systems Rev., vol. 4, no. 1, Feb. 1998, pp. 84-95.

[4] Bernick, D., Bruckert, B., Vigna, P. D., Garcia, D., Jardine, R., Klecka, J., and Smullen, J. NonStop® Advanced Architecture. Conf. on Dependable Systems and Networks, 2005, 12–21.

[5] Borkar, S. Challenges in Reliable System Design in the Presence of Transistor Variability and Degradation. IEEE Micro, vol. 25, no. 6, Nov.-Dec. 2005, pp. 10-16.

[6] Bower, F. et al. Tolerating hard faults in microprocessor array structures. In proceedings of the 2004 International Conference on Dependable Systems and Networks, 2004.

[7] Bressoud, T. C. and Schneider, F. B. Hypervisor-based fault tolerance. ACM Trans. Computer Systems 14, 1 (Feb. 1996), 80-107.

[8] Constantinescu, C. Trends and challenges in VLSI circuit reliability. IEEE Micro, 23(4):14–19, 2003.

[9] Dell, T.J. A White paper on the benefit of chipkill-correct ECC for PC Server Main Memory, IBM white paper, http://www-03.ibm.com/servers/eserver/pseries/campaigns/chipkill.pdf.

[10] Eagle Rock Alliance Ltd. Online survey results: 2001 cost of downtime. http://contingencyplanningresearch.com/2001.Survey.pdf, Aug. 2001.

[11] Fair, M.L., Conklin, C.R., Swaney, S. B., Meaney, P. J., Clarke, W. J., Alves, L. C., Modi, I. N., Freier, F. , Fischer, W. ,and Weber, N. E. Reliability, Availability, and Serviceability (RAS) of the IBM eServer z990. IBM Journal of Research and Development, Nov, 2004.

[12] Gold, B. T. et al. TRUSS: a reliable, scalable server architecture. IEEE Micro, Nov-Dec 2005.

[13] Gold, B. T., Smolens, J. C., Falsafi, B. and Hoe, J. C. The Granularity of Soft-Error Containment in Shared Memory Multiprocessors, Proceedings of The Workshop on Silicon Errors in Logic - System Effects (SELSE), 2006

[14] Gomaa, M. et al. Transient-fault recovery for chip multiprocessors. In Proceedings of the 30th International Symposium on Computer Architecture, June 2003.

[15] Hennessy, J. The Future of Systems Research. IEEE Computer, vol. 32, no. 8, Aug. 1999, pp. 27-33.

[16] Joseph, R. Exploring Core Salvage Techniques for Multi-core Architectures. Workshop on High Performance Computing Reliability Issues, 2005

[17] Mukherjee, S. S. et al. Detailed design and evaluation of redundant multithreading alternatives. In Proceedings of the 29th International Symposium on Computer Architecture, May 2002, 99–110.

[18] Nakano, J. et al. ReViveI/O: Efficient handling of I/O in highly-available rollback-recovery servers. In HPCA, 2006.

[19] Qureshi, M. K. et al. Microarchitecture-based introspection: A technique for transient-fault tolerance in microprocessors. In Proc. of 32nd Intl. Symp. on Comp. Arch. (ISCA-32), June 2005.

[20] Ranganathan, P., Adve, S., and Jouppi, N. P. Reconfigurable Cache and their Application to Media Processing, Proceedings of the 27th International Symposium on Computer Architecture (ISCA-27), June 2000.

[21] Ray, J. et al. Dual use of superscalar datapath for transient-fault detection and recovery. In Proceedings of the 34th International Symposium on Microarchitecture, December 2001.

[22] Reinhardt, S. K. and Mukherjee, S. S. Transient fault detection via simultaneous multithreading. In Proceedings of the 27th International Symposium on Computer Architecture, June 2000.

[23] Rotenberg, E.. AR-SMT: A microarchitectural approach to fault tolerance in microprocessors. In Proceedings of the 29th International Symposium on Fault-Tolerant Computing, June 1999.

[24] Shivakumar, P. et al. Modeling the effect of technology trends on the soft error rate of combinational logic. In Proceedings of the International Conference on Dependable Systems and Networks, June 2002, 389–398.

[25] Shivakumar, P. Keckler, S. W., Moore, C. R., and Burger, D. Exploiting Microarchitectural Redundancy for Defect Tolerance. The 21st International Conference on Computer Design (ICCD), October, 2003

[26] Slegel, T.J. et al. IBM's S/390 G5 Microprocessor Design. IEEE Micro, vol. 19, no. 2, Mar./Apr. 1999, pp. 12-23

[27] Smolens, J. C. et al. Efficient resource sharing in concurrent error detecting superscalar microarchitectures. In Proc. of 37th IEEE/ACM Intl. Symp. on Microarch. (MICRO 37), December 2004.

[28] Sorin, D. J. et al. SafetyNet: improving the availability of shared memory multiprocessors with global checkpoint/recovery. In Proc. of 29th Intl. Symp. on Comp. Arch. (ISCA-29), June 2002.

[29] Srinivasan, J., Adve, S.V., Bose, P., Rivers, J.A. The Impact of Technology Scaling on Lifetime Reliability. Proceedings of International Conference on Dependable Systems and Networks (DSN '04) June 2004.

[30] Srinivasan, J., Adve, S. V., Bose, P., and Rivers, J. A. The Case for Lifetime Reliability-Aware Microprocessors. Proceedings of 31st International Symposium on Computer Architecture (ISCA '04) June 2004.

[31] Srinivasan, J., Adve, S. V., Bose, P., and Rivers, J. A. Exploiting Structural Duplication for Lifetime Reliability Enhancement. In Proceedings of the 32nd International Symposium on Computer Architecture (ISCA'05), June 2005.

[32] Sundaramoorthy, K. et al. Slipstream processors: Improving both performance and fault tolerance. In ASPLOS, October 2000.

[33] Vijaykumar, T. N. et al. Transient-fault recovery using simultaneous multithreading. In Proceedings of the 29th International Symposium on Computer Architecture, May 2002.

[34] Wunderlich, R. E., Wenisch, T. F., Falsafi, B., and Hoe, J. C. 2003. SMARTS: accelerating microarchitecture simulation via rigorous statistical sampling. In Proceedings of the 30th Annual international Symposium on Computer Architecture, June 2003.

[35] SPEC Benchmark Suite. http://www.spec.org and http://www.spec.org/cpu/analysis/memory/

[36] International Technology Roadmap for Semiconductors. http://www.itrs.net/

[37] Falcon, A. Faraboschi, P., and Ortega, D. Combining Simulation and Virtualization through Dynamic Sampling. ISPASS-2007.

[38] Foxton Technology, http://www.intel.com/technology/magazine/computing/foxton-technology-0905.htm

[39] Barroso, L. A., Gharachorloo, K., McNamara, R., Nowatzyk, A., Qadeer, S., Sano, B., Smith, S., Stets, R., and Verghese, B. Piranha: A scalable architecture based on single-chip multiprocessing. In Proceedings of the 27th International Symposium on Computer Architecture, June 2000.

[40] Kongetira, P., Aingaran, K., and Olukotun, K. Niagara: A 32-way multithreaded SPARC processor. IEEE Micro, 25(2):21–29, 2005.

[41] Tendler, J. M., Dodson, J. S., Fields Jr., J. S., Le, H., and Sinharoy, B. IBM Power4 system microarchitecture. IBM Journal of Research and Development, 46(1):5–26, 2002.