Isolation in Commodity Multicore Processors

Nidhi Aggarwal, University of Wisconsin-Madison
Parthasarathy Ranganathan and Norman P. Jouppi, Hewlett-Packard Laboratories
James E. Smith, University of Wisconsin-Madison

Resource sharing in modern chip multiprocessors (multicores) provides many cost and performance benefits. However, component sharing also creates drawbacks for fault, performance, and security isolation. Thus, integration of components on a multicore chip should also be accompanied by features that help isolate effects of faults, destructive performance interference, and security breaches.

Technology scaling and power trends have led to the widespread emergence of chip multiprocessors (CMPs) as the predominant hardware paradigm.[1] Multiple cores are being integrated on a single chip and made available for general-purpose computing. Intel and AMD manufacture dual-core processors and, more recently, quad-core processors. From a system viewpoint, CMPs provide higher levels of integration, typically including multiple processing cores, caches, memory controllers, and even some I/O processing—all in a single socket. The Sun Niagara processor, for example, includes eight cores, a shared second-level cache, and integrated memory controllers and I/O interfaces. The IBM Power5 dual-core processor has an on-chip memory controller.

Trends toward multiple cores will likely continue. Indeed, at a recent Intel Developer Forum, the company announced an aggressive roadmap of multicore processors with on-chip integration, including an 80-core prototype chip. AMD and other processor vendors have similar roadmaps. Further research in the academic community focuses on processors with a much larger number of cores,[2] as well as interesting variations in the design of multicore chips to include asymmetric and conjoined multicore processors.[3]

This scaling to include more cores allows for greater computational capability and system integration at the chip level. In turn, this enables cost benefits from reduced component count. Additionally, enhanced resource sharing leads to better performance. On-chip components can now be easily shared to improve resource utilization, such as core sharing via hyperthreading, shared caches, and I/O interfaces. However, the same features of multicore processors that offer benefits can also present drawbacks. In particular, the increased levels of consolidation and integration lead to important isolation concerns—for performance, security, and fault tolerance.

Fault tolerance is an area of major concern. This is a particularly important issue given that recent studies have shown dramatic increases in the number of hardware errors when scaling technology to smaller feature sizes.[4] Developers have encountered two main kinds of errors. First, defects in the silicon cause permanent or intermittent hardware faults, resulting in wear out over time and leading to *hard errors*. Second, electrical noise or external radiation can cause transient faults when, for example, alpha radiation from impurities or gamma radiation from outside changes random bits, leading to *soft errors*.

With CMPs, the fault-tolerance problem is compounded because a fault in any single component can lead to the failure of the entire chip. The *failure in time* (FIT) of cores, caches, memory, or I/O components combines to provide a high FIT for the CMP. Future CMP

0018-9162/07/$25.00 © 2007 IEEE

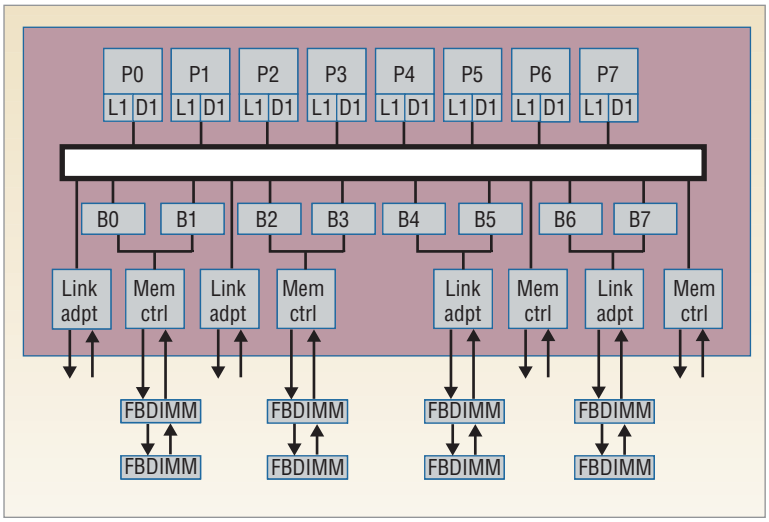

Figure 1. Conventional chip multiprocessor architecture. This CMP architecture has eight cores, P0 ... P7, each with private L1 caches; an eight-way banked, shared L2 cache, B0 … B7; four memory controllers; and coherent links to other sockets or I/O hubs; FBD, IMM = fully buffered dual in-line memory module, Link adpt = link adapter, and Mem ctrl = memory controller.

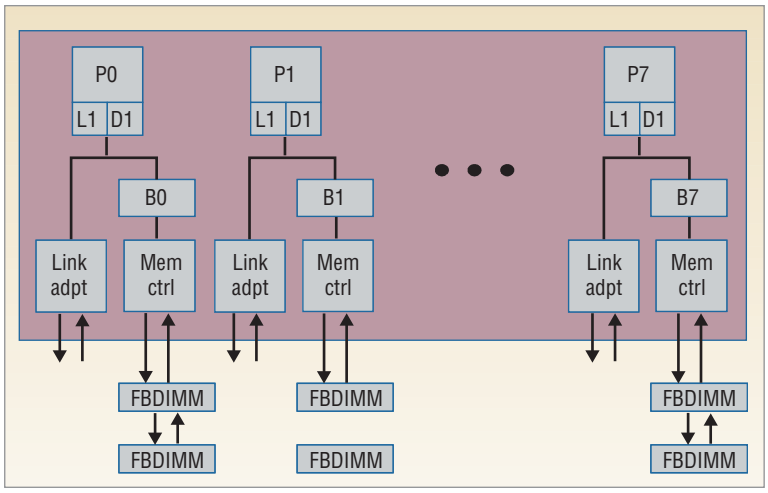

Figure 2. Static isolation. Independent computers are fabricated on the same die, and each computer has its own memory controller and I/O connections.

designs must offer the capability to isolate the faulty components and map them out so that the chip can be used with the remaining fault-free components.[4]

Figure 1 shows a conventional CMP architecture with eight cores, P0 … P7, each with private L1 caches, an eight-way banked, shared L2 cache, four memory controllers, and coherent links—such as Hypertransport—to other CMP sockets or I/O hubs. In this architecture, a bidirectional ring connects the processors and cache banks, but other configurations with more complex 2D arrangements, such as meshes and interleaved layouts, are possible.

As the number of cores in a CMP increases geometrically with lithographic scaling, a failure in one part of the conventional organization affects increasingly larger amounts of computational capability. For example, if the system shares all L2 cache banks and employs low-order address interleaving among the banks, a transient fault in the cache controller state machine can lead to an erroneous coherence state. Using error detecting codes on a coherence bit does not help in this case because the fault lies in the cache controller logic, before the coherence bits are set. Such a fault affects an entire chip's availability. Similarly, a fault in a memory controller, or anywhere in the ring interconnect, affects all the cores.

As the "Examining Current Commodity CMPs for Fault Isolation" sidebar describes, in the past when individual processors, memory controllers, and cache memory SRAMs provided the basic system building blocks, system designers could achieve good fault isolation by combining these chip-level components into redundant configurations at the board level. When necessary, designers can incorporate small amounts of application-specific integrated circuit (ASIC) glue logic. For example, the HP NonStop Advanced Architecture implements process pairs and fault-containment boundaries at the socket level.

With multicore approaches, however, socket-level isolation is no longer an attractive solution, and neither is using off-chip glue logic, especially for small systems. With growing numbers of cores at the socket level, implementing redundant configurations using different parts of a single multicore processor has become increasingly desirable. However, the lack of fault isolation in current multicore processors makes this impossible.

CHALLENGES

Static isolation, in which independent computers are fabricated on the same die, is a very straightforward approach to providing isolation in multicore processors. As Figure 2 shows, each computer has its own memory controller and I/O connections. This architecture has several disadvantages, however. The static partitioning of cache resources—which inhibit any interprocessor sharing—significantly reduces overall system performance and has not been used in proposed CMP designs. Similarly, static partitioning of chip interfaces and pins uses off-chip bandwidth inefficiently, making such a design unattractive for high-volume applications in which performance rather than high availability is the objective. Therefore, this offers a poor design choice for

balancing the tradeoffs between isolation and the benefits from shared resources.

The challenge therefore is to design techniques for configuring "off-the-shelf" CMPs with relatively little added on-chip hardware and complexity into high-availability, redundant systems. This can enable configuring the levels of sharing dynamically, allowing isolation to be selectively turned on when needed.

CONFIGURABLE ISOLATION

We propose CMP implementations with *configurable isolation*—a set of techniques for dynamically config-

Examining Current Commodity CMPs for Fault Isolation

We analyzed the reliability and availability features of five commodity multicore architectures from key vendors: IBM's Power5,[1,2] AMD's Opteron,[3] Sun's Niagara,[4] and Intel's Xeon[5] and Montecito.[6] Figures A shows the five commodity CMP architectures.

AMD's Opteron 64-bit microprocessor has an on-chip memory controller and three HyperTransport links. The links connect an Opteron to other Opteron processors without additional chips. The Opteron has error-correcting codes (ECC) and protects large storage arrays like caches and memory. Hardware scrubbers are implemented for the L1 data cache, L2 cache tags, and DRAM, which supports chip kill ECC.

Sun Niagara is a CMP of multithreaded cores that supports 32 threads, with four threads per core. All the cores share a single floating-point unit. The memory system consists of an on-chip crossbar, L2 cache, and memory controllers. Each L2 bank connects to one memory controller. Niagara also supports ECC, chip kill, and memory scrubbing to protect against errors in the storage arrays. In addition, the chip has extensive support for per-thread trap detection and notification.

In the Northbridge, Intel Xeon-based 64-bit multiprocessors have multiple cores sharing a single external memory controller. The Xeon also supports ECC, parity, and scrubbing to protect the storage arrays.

The IBM Power5 is a dual-core, two-way SMT processor with an on-chip memory controller. Power5-based multiprocessors have extensive error checking and also include reliability features such as support for CPU sparing, chip kill, ECC, and parity for the memory hierarchy.

Intel Montecito is an Itanium-based dual-core and dual-threaded processor. Montecito provides parity protection for register files in addition to the ECC, scrubbing, and parity protection in the memory hierarchy. It also supports steering logic to isolate hard errors in the L3 cache lines.

Key Components

We divided each CMP into different components and then characterized whether they satisfied key requirements for fault tolerance: fault isolation, fault detection, and online repair. These three requirements are typically satisfied by employing redundancy.

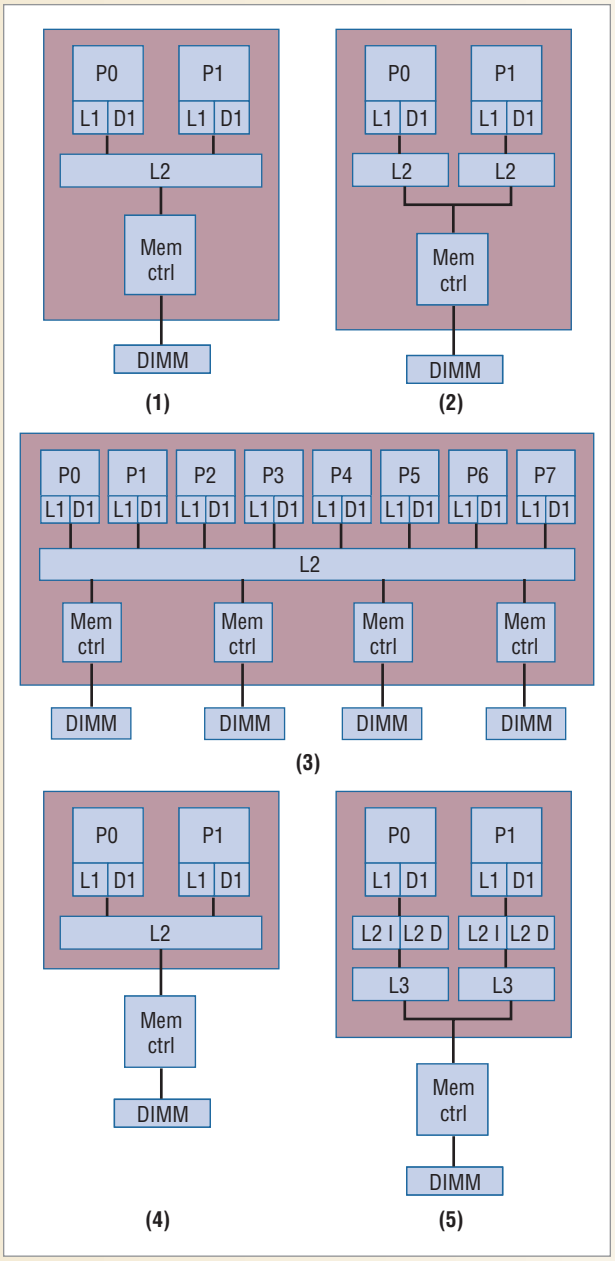

Figure A. High-level architectural block diagrams of the various commodity processors studied. Diagrams 1-5 correspondingly approximate the IBM Power5, AMD Opteron, Sun Niagara, Intel Xeon, and Intel Montecito.

Continued on the next page

Table A. IBM zSeries.

Component	Redundancy	Fault isolation	Fault detection	Online repair
Core	8 spare processors (in 4 books)	Processors checkstops on failure	Mirrored pipeline, ECC, and parity with retry in register files	Dynamic core sparing, checkpoint at each instruction boundary, concurrent book add, checkpoint transplant to spare processor, separate register file for checkpoint
Cache	Active redundant L2 cache, redundant L2 rings	Special uncorrectable error codes	L1 - parity protected, L2 - ECC protected, XOR checking of control signals from L2 chips	Capability to add new cache at L2 ring interface, retry
Memory	Spare DRAM chips, redundant main storage controller, redundant store protect keys	Isolation of erroneous DRAM chip and store key	TMR for store keys, ECC, memory scrubbing, extra ECC code space, special uncorrectable error (UE) codes to indicate error source	Concurrent book add, chip kill, ECC
I/O	Redundant memory bus adapters (per book), I/O resources shared across all partitions	Single memory bus adapter clockstep design	Parity protection, command reject request, forced hang, special UE tag for known uncorrectable data	Operation retries, concurrent book add
System data and control buses	Redundant buses	Independent buses, immediate checkstop on control bus UE, regeneration of ECC across interfaces	Parity protected tag bit for uncorrectable data	Call for repair

Cores

Inside the core, currently transient fault detection is mainly restricted to the register file via parity or ECC. Montecito provides an exception with its built-in lockstep support and internal soft-error-checking capabilities. Opteron, Xeon, and Niagara have no fault isolation, so an error originating in any core can propagate to all other cores through the shared system components. Power5 and Montecito provide some degree of isolation for cores in different logical or electrical partitions, respectively.

In summary, all the commodity CMP architectures are vulnerable to soft errors, except Montecito in its lockstep configuration.

Caches

Most architectures are resilient to errors in the cache array and provide ECC or parity checking at all cache levels. However, Opteron and Xeon cannot tolerate errors that are not correctable by ECC alone, such as multibit errors. Niagara, Power5, and Montecito have more redundancy and fault isolation and can tolerate important classes of multibit errors. These CMPs usually share at least one level of the cache hierarchy, either across cores or contexts. However, none of the commodity CMPs can tolerate errors in the associated cache control circuitry.

Memory

Memory is perhaps the most fault-tolerant resource in commodity CMP systems. All the conditions for fault tolerance are satisfied in the memory arrays. This also reflects that historically memory is a system's most error-prone component.

All the architectures have sophisticated techniques like chip kill, background scrubbing, and DIMM-sparing to tolerate failures. However, there is no tolerance to failures in memory access control circuitry. A failure in any memory controller or anywhere in the interconnect would affect all the cores. For example, in a design like the Xeon, an error in one memory controller in the shared Northbridge memory controller hub can affect multiple cores. On the other hand, in Opteron the failure of an on-chip memory controller can potentially be isolated to the cores in that chip.

Summary

Overall, we find that existing transient fault detection is limited to storage arrays such as register files, cache, and memory arrays. The lack of system-level fault isolation poses the biggest problem. Shared components do not have adequate fault isolation because a fault in one shared component can affect all cores on the chip. This

Table B. HP NonStop.

Component	Redundancy	Fault isolation	Fault detection	Online repair
Core	Dual or triple modular redundancy	Isolated to a CPU	Compare results of I/O	Reintegration of new processing element, dedicated reintegration link
Cache	Dual or triple modular redundancy	Isolated to cache	Compare results of I/O	Replace processor slice
Memory	Dual or triple modular redundancy	Isolated to memory, no shared memory	Compare results of I/O, symmetric handling of interrupts for memory coherence across replicas	Replace processor slice
I/O	Dual redundant SAN	Independent fabrics	Self-checked circuits	Online replacement of logical synchronization units
System data and control buses	Redundant buses	Independent buses	CRC checksums	Replace processor slice

is true even if the system is running programs in a dual-modular redundant (DMR) or triple-modular redundant (TMR) configuration.

Comparing High-End, High-Availability Systems

We also examined two state-of-the-art systems, the IBM zSeries, shown in Table A, and the HP NonStop, shown in Table B. Enterprise-class applications that demand continuous availability use both of these systems.

The NonStop systems are DMR or TMR fault-tolerant servers built from standard HP four-way SMP Itanium server processor modules, memory boards, and power infrastructure. The processors communicate with each other and with shared I/O adapters through the ServerNet system area network (SAN). Each row of processors in a dual-mode or triple-mode redundant configuration forms one logical processor, which is made up of processor elements, one from each of the slices. The logical processor is the self-checked member of the cluster. Each processor element is a microprocessor running its own instruction stream and has a portion of the slice memory dedicated to its use. There are no synchronized clocks among the slices. The system compares all outputs from the servers at the I/O operation level (both IPC and device I/O) for 100 percent detection of faults. The voters themselves are self-checked.

The IBM zSeries servers incorporate extensive reliability, availability, and serviceability features to prevent both hard and soft errors. The zSeries pipeline is duplicated and each instruction checked before committing its results to an architected state. The servers have extensive redundancy in all components, including processors, L2 rings, L2 cache, and memory bus adapters. Most of the redundant components can be deployed dynamically, with no downtime, using techniques like Concurrent Book Add and Dynamic CPU Sparing. Wendy Bartlett and Lisa Spainhower provide an excellent discussion of the evolution of the NonStop and zSeries systems.[7]

References

1. D.C. Bossen et al., "Fault-Tolerant Design of the IBM Pseries 690 System Using Power4 Processor Technology," *IBM J. Research and Development*, vol. 46, no. 1, 2002, pp. 77-86.
2. "IBM Power5 Processor-Based Servers: A Highly Available Design for Business-Critical Applications," IBM white paper; www.ibm.com/systems/p/hardware/whitepapers/power5_ras.html.
3. C.N. Keltcher et al., "The AMD Opteron Processor for Multiprocessor Servers," *IEEE Micro*, vol. 23, no. 2, 2003, pp. 66-76.
4. P. Kongetira, K. Aingaran, and K. Olukotun, "Niagara: A 32-Way Multithreaded Sparc Processor," *IEEE Micro*, vol. 25, no. 2, 2005, pp. 21-29.
5. "Reliability, Availability, and Serviceability for the Always-on Enterprise," Intel white paper; www.intel.com/business/bss/products/server/ras.pdf.
6. C. McNairy and R. Bhatia, "Montecito: A Dual-Core, Dual-Thread Itanium Processor," *IEEE Micro*, vol. 25, no. 2, 2005, pp. 10-20.
7. W. Bartlett and L. Spainhower, "Commercial Fault Tolerance: A Tale of Two Systems," *IEEE Trans. Dependable and Secure Computing*, vol. 1, no. 1, 2004, pp. 87-96.

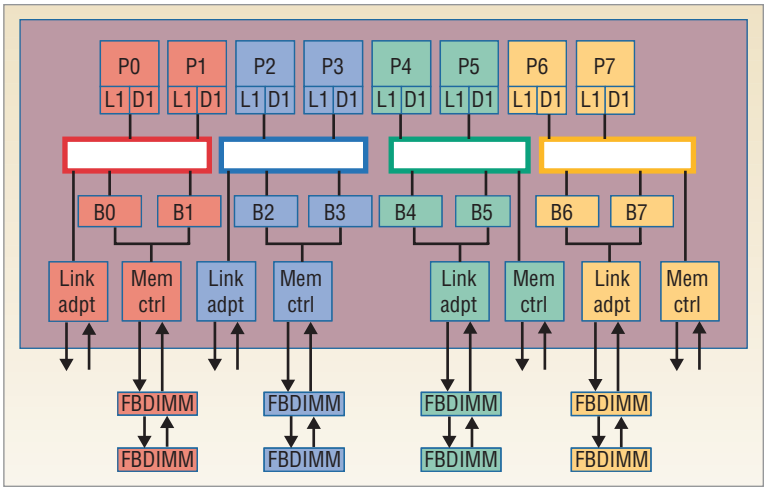

Figure 3. Configurable isolation. The introduction of low-cost configurable isolation at the interconnect, caches, and memory-controller levels provides a set of techniques for configuring the system with different isolation levels by controlling resource sharing.

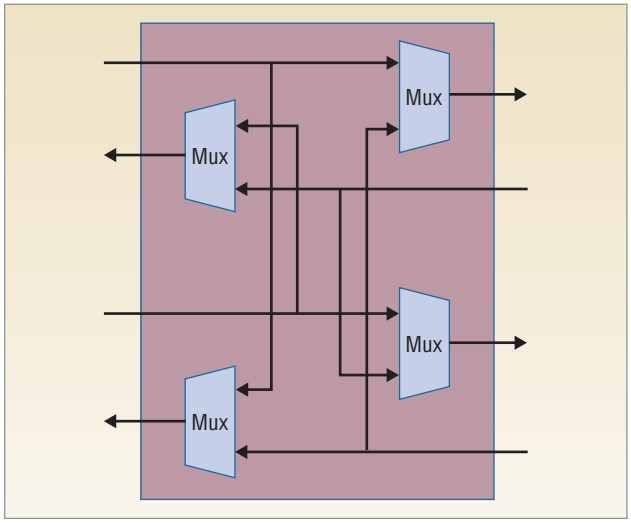

Figure 4. Ring configuration units. Physically, the ring forms the chip's central spine, so the cross-links should be less than a millimeter long. Their activation requires inserting a multiplexer at the input of a ring interface incoming data port.

uring the system with different isolation levels by controlling resource sharing. Figure 3 shows one such system.

The key difference between the system in Figure 3 and the one in Figure 1—the baseline architecture—is the introduction of low-cost configurable isolation at the interconnect, caches, and memory controller levels. For example, the ring interconnect in Figure 1 has been cut apart and reconfigured to create multiple logically independent rings using configuration crosslinks similar to the ring configuration units (RCU) shown in Figure 4. Physically, the ring is expected to form the chip's central spine, so the crosslinks should be less than a millimeter long, and their activation requires inserting a multiplexer at the input of a ring interface incoming data port. The crosslinks and input multiplexers introduce a small additional fixed cost in terms of area and power, which does not significantly increase the design's cost for system configurations in which higher availability is not an objective.

As shown, an RCU can be implemented using multiplexers. Under software configuration control, the multiplexers can pass signals through to create a larger ring, or they can divide the larger ring into separate segments. The crosslinks are also expected to be shorter than the ring segments between cores, so the crossconnects should operate at least as fast as core-to-core or bank-to-bank ring segments.

Because the cross-links and input multiplexers are shared and can form a single point of failure, they must be implemented using self-checked logic if the design requires stringent fault tolerance. At the cache level, providing configurable isolation requires small changes to the ring and bank addressing. When the system software partitions the intercore interconnect, fewer address bits are required for interleaving among L2 cache banks within a single domain. Therefore, the L2 cache size available in a domain is inversely proportional to the number of domains.

Providing reconfiguration capabilities for cache banks and memory controllers requires the addition of two mode bits and extra tag bits. The first mode bit and one extra tag bit enable caching lines from the bank connected to the same memory controller. Another mode and two tag bits can enable caching lines from banks connected to a different memory controller. Overall, the number of extra bits required in a bank to enable caching of lines from any other bank in the system is $\log_2$ (number of banks).

This architecture offers the advantage that the system can be partitioned into separate *domains* on the fly, starting, for example, with Figure 1, and using configurable isolation to separate faulty domains from working domains. If there is a core fault, system software can isolate it within its domain and continue functioning with cores in the remaining, working domains. Further, the proposed architecture can continue functioning in the event of faults in cache banks, memory controllers, and the interconnection network. For example, if a fault occurs in a cache bank—say, B0 in Figure 1—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. Should a memory controller fault occur, lines cached by both the B4 and B5 banks, 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 isolating the faulty components and reconfiguring the system to use the remaining fault-free components.

The architecture in Figure 3 offers another advantage. Because system software can now divide the multicore processor into separate isolated domains, the separate domains can execute redundant copies of the same software to check for soft or transient errors. For example, Figure 5 shows how the system can be configured into two domains. The system employs resources from two domains to run dual-modular redundant (DMR) process pairs, with computations in the one domain (red in the figure) replicated in the second (green) domain when higher availability is required. In this design, self-checked voters compare the output of the redundant execution to detect errors.

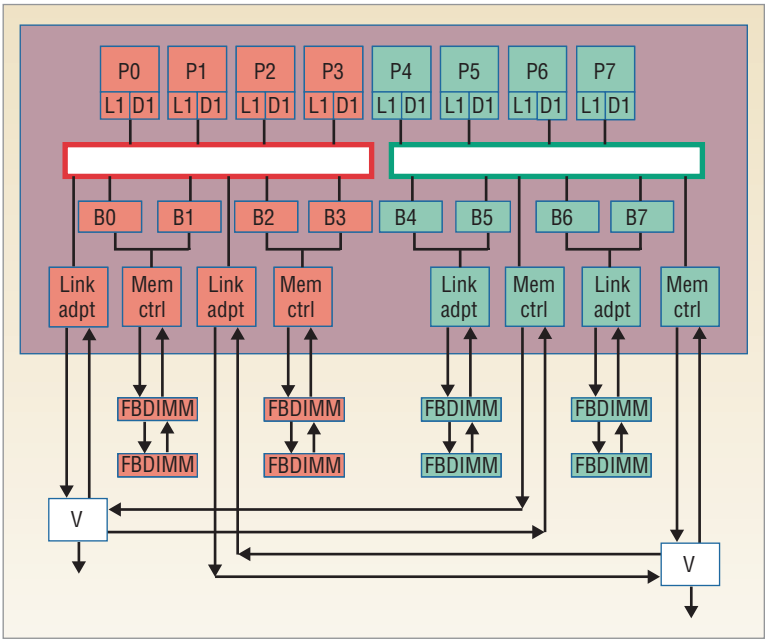

Figure 5. Dual fault domains. The system can be configured into two domains to run a dual-modular redundant process pair, with computations in the one domain (red) replicated in a second (green) domain when the system requires higher availability.

For highest availability, voters can be implemented in I/O hubs connected to adapters from the redundant domains, similar to the hardware voters in the Nonstop Advanced Architecture.[5] For lower-cost, lower-availability solutions, hypervisors that communicate between the redundant domains through I/O can implement the voter.[6] Similarly, we could start with Figure 3 and use three isolated domains to enable a triple modular redundant (TMR) configuration. Further, the number of domains need not be static if the RCUs are self-checked, and they can be changed as system needs evolve.

BENEFITS

Configurable isolation in a CMP lets reconfiguration map out the faulty component and provides graceful performance degradation. We evaluated the impact of hard faults and subsequent reconfiguration on the system's computing capacity over its lifetime by comparing three architectures:

- *Shared.* A completely shared system similar to proposed CMPs, as shown in Figure 1.
- *Static isolation.* A completely private system with full isolation, as shown in Figure 2.
- *Configurable isolation.* Our proposed architecture, with reconfiguration and configurable isolation, as shown in Figure 3.

Because the configurable-isolation architecture does not contain any modification to the cores, the size of the working set and its effect on cache behavior is the most important workload characteristic. Using SPEC benchmarks, we constructed three workloads with large, mixed, and small memory footprints. Over the course of a simulation run, as cores become unusable due to hard faults, benchmarks drop from the workloads, reflecting the loss of computing capability.

The fault model is based on state-of-the-art technology and derived from detailed and confidential microprocessor vendor models. We used HP-internal fault-analysis experiments to calibrate the fault model. The fault data includes FIT rates and distributions for hard and soft errors per component. We modeled five different regions that represent the granularity of reconfigurations: core and L1 cache, L2 circuitry, L2 banks, memory controller circuitry, and link controller.

On the shared system, any hard fault leads to system failure. This means that after a failure, such a system's throughput drops to zero for all workloads. On the statically isolated system, any single fault leads only to the loss of throughput from the benchmark mapped to that private system. For example, even a fault in the bank associated with a core leads to that core being unusable. On a configurable isolated system, a fault—in a memory controller, for example—leads to loss of performance from the banks connected to the memory controller, but not the loss of a workload. Only when a core fails does a benchmark drop from the workload. Thus, the entire system becomes unusable in the configurable isolated architecture only when the last component of any type fails.

To make evaluation feasible, we used a two-phase methodology to simulate the performance of different processor configurations for various fault-arrival

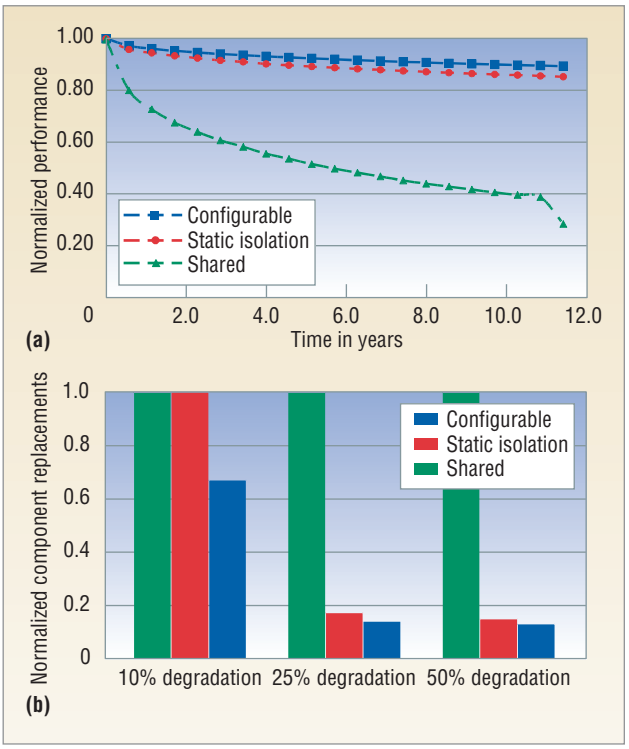

Figure 6. Evaluating the benefits of reconfiguration. Normalized performance from Monte Carlo hard-fault simulation over an 11-year period generated the results shown for three architectures—a baseline conventional system with full sharing; the proposed system, with configurable isolation; and a system with static isolation: (a) Performance over time and (b) normalized component replacements.

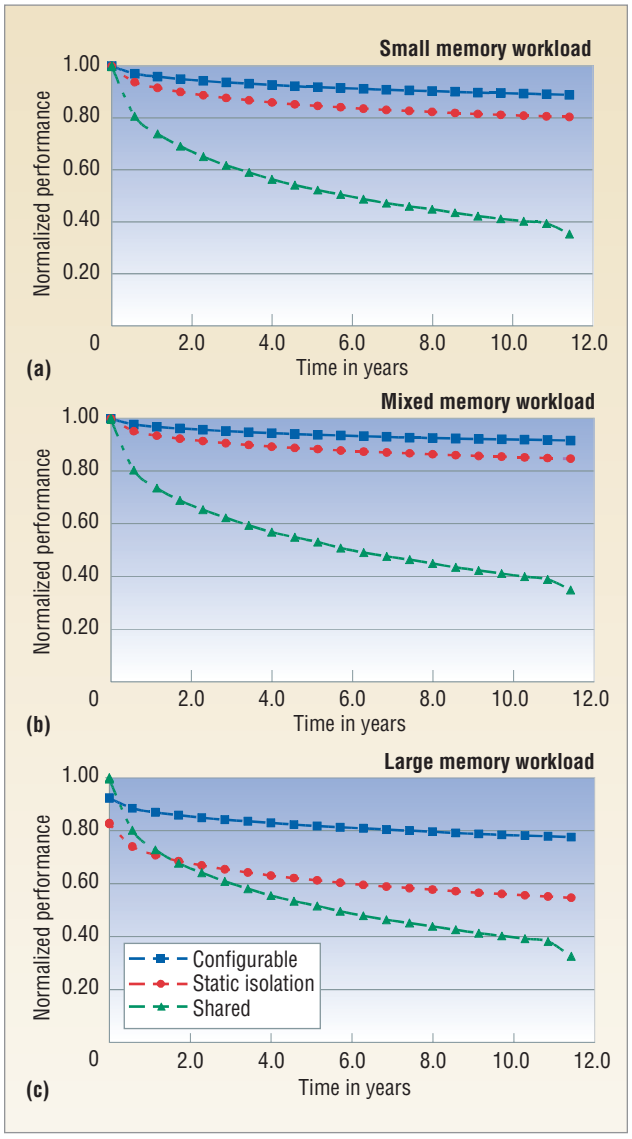

Figure 7. Three dual-modular redundant (DMR) configuration architectures: shared, statically isolated, and configurable isolation. The benefits of configurable isolation for providing graceful performance degradation in the event of hard faults (for DMR systems with transient fault protection) are shown across three memory workloads: (a) small, (b), mixed, and (c) large.

scenarios. First, using more than one machine-year, we ran a full-system simulator to exhaustively simulate the possible system configurations and compute the throughput of all configurations, subject to specific policies. Second, we performed a Monte Carlo simulation using a detailed component-level fault model. By running the Monte Carlo simulation for 10,000 runs, we simulated fault injection in a total of 10,000 systems, with each run comprising 100,000 simulated hours—approximately 11 years, as Figure 6 shows.

All simulations were done using a full system x86/x86-64 simulator based on AMD SimNow, which can boot an unmodified Windows or Linux OS and execute complex application programs. We used a timing model with a memory hierarchy similar to that supported by an AMD Opteron 280 processor, except with smaller L2 cache sizes to match the workloads' working set.[1]

As Figure 6a shows, performance degrades more gracefully with respect to hard faults in a system with configurable isolation. The average performance of the configurable isolation architecture degrades by less than 10 percent over 10 years. In contrast, the fully shared configuration degrades by almost 60 percent over the same period.

Figure 6b provides an alternate view of configurable isolation's benefits, showing the number of component replacements for each of the three approaches. We assume that the system continues to stay operational until the performance dips below a certain threshold, after which the entire multicore component must be replaced and the performance reinitialized to that of the no-fault configuration. The simulation then continues for the remainder of the 100,000 hours with the new system.

We consider three cases in which the performance threshold is set to 90, 75, and 50 percent of initial performance. The total number of replacements across

Other Benefits from Isolation

In addition to fault isolation that enables more graceful degradation of performance in the presence of faults, isolation offers other benefits.

Power Reprovisioning

Isolation can lead to optimizations that are otherwise impossible. For example, with suitable fault-isolation support, the power budget could be dynamically reprovisioned by reassigning the power allotments of faulty components to the remaining fault-free components. Figure B shows the results assuming a future fault-model when a failed core's power budget can be reallocated dynamically to increase the clock frequency of the remaining cores, leading to improved system performance.

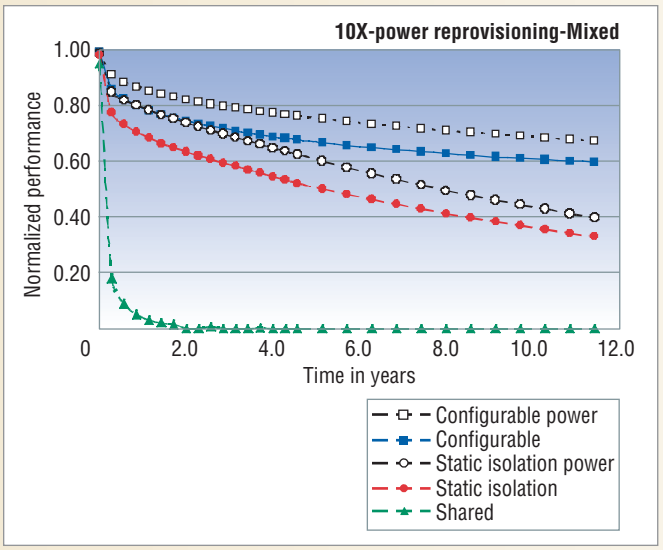

Figure B. Dynamic power reallocation with a future fault-model. System performance can be improved when a failed core's power budget is reallocated dynamically to increase the clock frequency of the remaining cores.

Trust and Performance

Other kinds of isolation include trust and performance. Consider, for example, scenarios in which workloads of different priorities compete for shared resources or have destructive interference—such as a background virus scanner running in parallel with an interactive user application. Support for isolation can ensure that the system partitions resources for the two workloads dynamically to avoid conflict. Similarly, in environments where the same computing platform hosts multiple workloads with different service-level agreements, configurable isolation can be used to partition resources to end users based on priority.

Recent studies describe the need for and benefits of performance isolation in a CMP.[1,2] Kyle Nesbit and colleagues[2] propose a virtual private machine system that allocates a set of CMP resources—processors, bandwidth, and memory resources—to individual tasks. Virtual private machines isolate performance for coscheduled tasks in a CMP and ensure that the performance does not vary significantly regardless of the load placed on the system by other tasks.

Even if performance is not an issue, from a security and trust viewpoint, isolation could still prove useful to avoid malicious attacks from one user or application affecting other users or applications hosted on the same multicore. Dong Woo and Hsien-Hsin Lee[3] suggest active monitoring to identify denial-of-service attacks and point out the challenges in differentiating attack scenarios from normal heavy-usage cases.

References

1. R. Iyer, "CQoS: A Framework for Enabling QoS in Shared Caches of CMP Platforms," *Proc. Int'l Conf. Supercomputing* (ICS 04), ACM Press, 2004, pp. 257-266.
2. K.J. Nesbit, J. Laudon, and J.E. Smith, "Virtual Private Caches," *Proc. Int'l Symp. Computer Architecture* (ISCA 07), IEEE CS Press, 2007, in press.
3. D.H. Woo and H.H. Lee, "Analyzing Performance Vulnerability Due to Resource Denial-of-Service Attack on Chip Multiprocessors," *Proc. Workshop Chip Multiprocessor Memory Systems and Interconnects*, 2007.

10,000 Monte Carlo runs for a statically isolated and configurable isolated system is normalized with respect to the total number of replacements for a fully shared system. In such a system, every fault leads to system replacement because the performance drops to zero. These results show that the architecture with configurable isolation dramatically reduces the need to replace components irrespective of performance thresholds.

Figure 7 presents results for the three architectures when used in a DMR configuration. In this configuration, a core failure would lead to loss of throughput from both copies of the benchmark. Since memory footprint affects performance significantly in this configuration, we present results for the large, mixed, and small memory workloads in Figures 7a, b, and c, respectively.

As expected, the shared system performs worst, with a dramatic degradation in average performance of 30 to 35 percent during the first two years, and degradation close to 50 percent by the end of five years. The statically isolated configuration is more resilient to failures and pro-

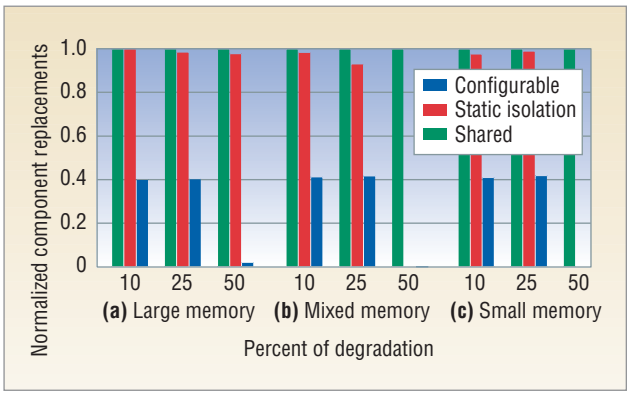

Figure 8. Number of normalized component replacements as a function of performance. When comparing the three architectures, assume components are replaced (a) when performance dips below 90 percent, (b) when performance dips below 75 percent, and (c) when performance dips below 50 percent.

vides more gradual performance degradation. Over five years, the net performance loss is only 10 to 15 percent.

The results for the large memory workload in Figure 7c are particularly interesting. Here, the isolated configurations (statically isolated and configurable isolated), by virtue of having private caches, initially underperform the shared configuration. However, compared to the fully shared system, the statically isolated system becomes performance competitive at around two years, the crossover point in the curves in Figure 7c.

Figure 8 presents the results for the number of component replacements required for the three architectures across all three workloads. The configurable isolation system consistently achieves the best performance across all workloads. With configurable isolation, resources can still be shared within a given fault domain. Additionally, dynamically repartitioning the resources leads to the most graceful degradation across all three workloads.

Several enhancements to the configurable isolation architecture provide additional benefits. For example, we assumed a single process per core. Overloading processes on remaining cores in a given working domain can potentially mitigate some of the performance degradation from losing a core in that domain. Similarly, when remapping fault domains, we assume arbitrary remapping of the fault domains and assignment of processes to cores. More advanced policies, aware of workload requirements and latency effects, could improve performance further. For example, prior work on heterogeneous multicore architectures demonstrates significant benefits from intelligently mapping workloads to available hardware resources.[2]

Configurable isolation also offers the ability to dynamically reconfigure the system's availability guarantees. The approach we propose lets the system be configured to a spectrum of choices, from no-fault isolation to multiple smaller domains. For example, in utility-computing environments, a server can be provisioned as a payroll server with high levels of availability turned on, then it can be redeployed later as a Web server with lower availability levels. Isolation in CMP environments also has benefits beyond availability; the sidebar on "Other Benefits from Isolation" discusses some of these.

Multiple cores will provide unprecedented compute power on a single chip. However, integration of several components on a chip must be accompanied by features that enable isolation from fault effects, destructive performance interference, and security breaches. These features must ideally be low cost in terms of power and area and not impact the performance of the system adversely.

Here, we focus on isolation from faults. Future processors will be increasingly susceptible to hardware errors. The impact of errors on a conventional CMP with extensive sharing will likely be severe because the shared resources lack system-level fault isolation. Much of the recent architecture research in fault-tolerant systems has focused on tolerating errors originating in the core, such as DIVA,[7] AR-SMT,[8] chip-level redundantly threaded processor with recovery (CRTR),[9] dynamic reliability management,[10] total reliability using scalable servers,[11] and several others that use the extra cores or contexts available in a CMP. Other system-level recovery solutions for SMPs, such as NonStop[5] and zSeries,[12] handle errors in the interconnection network and the cache coherence protocol, but they do not deal with the lack of fault isolation in CMPs.

For reliability at the system level, all components of the chip must be protected and faults must be isolated to smaller fault domains than the entire socket. Our design requires minimal hardware changes and retains the commodity economics and performance advantages of current CMPs. Further, we believe that there are exciting research opportunities in the area of enabling low-cost isolation features in CMPs that can enable them to be used as building blocks for high-performance, dependable, and secure systems.

References

1. This article is based on an earlier work: N. Aggarwal et al., "Configurable Isolation: Building High Availability Systems with Commodity Multicore Processors," *Proc. Int'l Symp. Computer Architecture* (ISCA 07), ACM Press, 2007; http://doi.acm.org/10.1145/1250662.1250720.
2. D. Patterson, "Recovery-Oriented Computing: A New Research Agenda for a New Century," keynote address, *Proc. Int'l Symp. High-Performance Computer Architecture* (HPCA 02), 2002; http://roc.cs.berkeley.edu/talks/pdf/HPCAkeynote.pdf.

3. R. Kumar et al., "Heterogeneous Chip Multiprocessors," *Computer*, Nov. 2005, pp. 32-38.
4. S. Borkar, "Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation," *IEEE Micro*, Nov. 2005, pp. 10-16.
5. D. Bernick et al., "NonStop Advanced Architecture," *Proc. Int'l Conf. Dependable Systems and Networks* (DSN), IEEE CS Press, 2005, pp. 12-21.
6. T.C. Bressoud and F.B. Schneider, "Hypervisor-Based Fault Tolerance," *ACM Trans. Computer Systems*, Feb. 1996, pp. 80-107.
7. T.M. Austin, "DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design," *Proc. Int'l Symp. Microarchitecture* (MICRO), IEEE CS Press, 1999, pp. 196-207.
8. E. Rotenberg, "AR-SMT: A Microarchitectural Approach to Fault Tolerance in Microprocessors," *Proc. Int'l Symp. Fault-Tolerant Computing*, IEEE CS Press, 1999, pp. 84-91.
9. M. Gomaa et al., "Transient-Fault Recovery for Chip Multiprocessors," *Proc. Int'l Symp. Computer Architecture* (ISCA), IEEE CS Press, 2003, pp. 98-109.
10. J. Srinivasan et al., "The Case for Lifetime Reliability-Aware Microprocessors," *Proc. Int'l Symp. Computer Architecture* (ISCA 04), IEEE CS Press, 2004, pp. 276-287.
11. J.C. Smolens et al., "Fingerprinting: Bounding Soft-Error Detection Latency and Bandwidth," *Proc. Int'l Conf. Architecture Support for Programming Languages and Operating Systems* (ASPLOS), ACM Press, 2004, pp. 224-234.
12. M.L. Fair et al., "Reliability, Availability, and Serviceability (RAS) of the IBM eServer z990," *IBM J. Research and Development*, Nov. 2004, pp. 519-534.

Nidhi Aggarwal *is a doctoral student in the Computer Sciences Department at the University of Wisconsin-Madison. Her research interests include high-performance and high-availability systems, virtual machines, and memory system design. Aggarwal received an MS from the Department of Electrical and Computer Engineering at the University of Wisconsin-Madison. Contact her at naggarwal@wisc.edu.*

Parthasarathy Ranganathan *is a principal research scientist at Hewlett-Packard Laboratories. His research interests include low-power design, system architecture, and parallel computing. Ranganathan received a PhD in electrical and computer engineering from Rice University. Contact him at partha.ranganathan@hp.com.*

Norman P. Jouppi *is a Fellow and Director of the Advanced Architecture Lab at Hewlett-Packard Laboratories. His research interests include highly parallel systems, high-performance networking, and the impact of photonics on computer systems architecture. Jouppi received a PhD in electrical engineering from Stanford University. Contact him at norm.jouppi@hp.com.*

James E. Smith *is a professor in the Department of Electrical and Computer Engineering at the University of Wisconsin-Madison. His research interests include high-performance processors and systems. Smith received a PhD in computer science from the University of Illinois. Contact him at jes@ece.wisc.edu.*