Improving Yield and Reliability of Chip Multiprocessors

Abhisek Pan
University Of Massachusetts
Amherst, MA, USA

Omer Khan
University Of Massachusetts
Amherst, MA, USA

Sandip Kundu
University Of Massachusetts
Amherst, MA, USA

Abstract— **An increasing number of hardware failures can be attributed to device reliability problems that cause partial system failure or shutdown. In this paper we propose a scheme for improving reliability of a homogeneous chip multiprocessor (CMP) that also serves to improve manufacturing yield. Our solution centers on exploiting the natural redundancy that already exists in multi-core systems by using services from other cores for functional units that are defective in a faulty core. A micro-architectural modification allows a core on a CMP to use another core as a coprocessor to service any instruction that the former cannot execute correctly. This service is accessed to improve yield and reliability, but at the cost of some loss of performance. In order to quantify this loss we have used a cycle-accurate simulator to simulate the performance of a dual-core system with one or two cores sustaining partial failure. Our results indicate that when a large and sparingly-used unit such as a floating point arithmetic unit fails in a core, even for a floating point intensive benchmark, we can continue to run each faulty core with help from companion cores with as little as 10% impact to performance and less than 1% area overhead.**

Keywords- yield; reliability; micorarchitecture; multiprocessors

I. INTRODUCTION

Relentless advancement in process technology during the last four decades has led to processor designs with progressively higher transistor count and increased clock frequency. However, sustaining this explosive device-count growth on a chip is going to be difficult due to critical yield and reliability problems [1].

The traditional accelerated life tests (burn-ins) used to filter out defective chips at their infancy are losing their effectiveness due to reduced headroom for stress testing [2]. Consequently, users of semiconductor chips may experience infant mortality problems. Infant mortality problems are aggravated further due to new aging defect mechanisms such as NBTI [3]. Yield recovery in field has been proposed as a potential solution to these problems [4]. This has been described in ITRS [4] as: "Relaxing the requirement of 100% correctness for devices and interconnects may dramatically reduce costs of manufacturing, verification, and test. Such a paradigm shift is likely forced in any case by technology scaling, which leads to more transient and permanent failures of signals, logic values, devices, and interconnects."

Another area of concern in foreseeable future is system reliability. Current design practice for processors (except high-end mainframes and some safety-critical systems) is to assume that the underlying fabric of transistors and interconnects will always operate correctly during the product lifetime. However continuous push for smaller devices and interconnects has moved the technology closer to a point of unreliability where such design paradigm is not valid [5]-[7]. For example, at 90nm technology Negative Bias Temperature Instability (NBTI), where a PMOS device degrades continuously with voltage and temperature stress, had become a major reliability concern, [8]-[9]. At 45nm, the problem has exacerbated due to lower threshold voltage, and Positive Bias Temperature Instability (PBTI) for NMOS devices has been added to the list [8]. Impact of other device-failure mechanisms like time dependent dielectric breakdown (TDDB) are also increasing because of extremely high on-chip temperatures, high current densities and thinner gate oxides [10]-[12]. Copper electro-migration, stress voiding, and electrical breakdown of low-k ILD have compromised the reliability of interconnects [4]. Erratic bit errors in SRAM and SEU (single event upset) based soft errors are on the rise. These problems, cumulatively referred to as PVT issues (process corner, voltage and temperature), are expected to worsen in future nano-CMOS technology [13].

These reliability defects cannot be pre-screened during manufacture. The majority of these defects will appear under specific voltage, temperature, frequency and workload conditions. Aggressive adaptive voltage and frequency management of modern processors are expected to compound such defects [14]. Hence, design paradigm of the future need to concentrate on building systems with imperfect transistors and interconnects; systems that will continue to function in spite of deteriorating components and multiple field failures [15]-[16]. A well-known approach towards building such systems involves the use of redundancy inherent in modern processors [17]-[24].

The existing solutions consider redundancies at granularities from system to intra-processor levels. However sharing of hardware across multiple cores on a chip has remained a relatively less explored area. In this paper we suggest a low-overhead micro-architectural scheme to exploit the redundancies across multiple cores in a CMP. The proposed scheme involves hardware-assisted communication between the cores to share functional units across them. This sharing is exposed at an instruction-level granularity.

The remainder of the paper is organized as follows: literature survey in section 2; proposed solution in section 3; architectural prerequisites in section 4; simulation and results in sections 5 and 6; summary conclusion in section 7.

II. RELTAED WORK

The idea of incorporating redundancy in microprocessors for yield and defect tolerance is well entrenched. The 16-bit HYETI microprocessor is an early example of such a system [24]. Commercial high-availability multi-processor systems like the

978-3-9810801-5-5/DATE09 © 2009 EDAA

IBM p690 have been designed to exploit redundancy at chip and module level [21]. These systems can map detected failures to individual CPUs and achieve system recovery by de-configuring the faulty processor during runtime or boot-up. More recently, researchers have proposed ideas to use redundancy at finer granularities in order to achieve more efficient use of redundant hardware [19]- [20].

Within a processor, storage components and logic arrays are provided with redundant rows, columns and sub-arrays for effective yield and reliability improvement [23], [26]. However structural irregularity and testability issues in logic and control units render them unsuitable for such partial redundancy [17]. Hence entire units need to be replicated to achieve better yield or fault-tolerance. In this regard Shivakumar *et al* explored the possibility of using multiple execution units already present within a processor to improve manufacturing yield at the cost of performance degradation [19]. In [20] somewhat similar ideas of exploiting existing redundancies or building idle 'spare' units within a processor to improve lifetime reliability were introduced. Reference [18] analyzes the benefits of sharing resources across partially damaged cores for yield enhancement. It proposes software-controlled thread swapping across cores to avoid faulty units. Along with the inherent performance degradation due to presence of faulty units, this scheme suffers from additional performance penalties due to repeated core hopping. The CASH (CMP And SMT Hybrid) architecture also advocates sharing of sparsely used functional units across several cores as a way to save area and reduce hardware complexity of individual SMT superscalar cores [25].

In this paper, we consider hardware-controlled resource sharing across cores.

III. PROPOSED SCHEME

A simplified version of the homogeneous dual-core CMP described in [22] is presented to illustrate the proposed scheme. Each core is assumed to be a superscalar out-of-order execution machine with private L1 data and instruction caches, and shared L2 cache. We assume a Symmetric Multiprocessor (SMP) paradigm for this proposed scheme. Basic modification involves incorporation of an Inter-Core queue (ICQ). The architecture of the ICQ and the structural and behavioral modifications required in the cores are described below.

A. Inter-Core Queue

Inter-Core Queue forms the interface for data-flow between a faulty and helper core in this proposed scheme [Figure 1]. It acts as a temporary storage for instructions that are to be transferred across cores. The ICQ maintains ordering of instructions using FIFO order. Each entry in the ICQ has data fields for the instruction including the source operands and destination, as well as some control bits to manage the control flow. We define the following control bits for proper communication between the ICQ and the cores. The details of the faulty and helper core are described in the later sections.

- Valid: Identifies occupancy of an ICQ entry
- Emergency: When set, identifies that the instruction has been in the queue long enough and needs to be served as early as possible
- Executed: When set, it indicates that instruction has completed execution at the helper core and the result is available in the ICQ
- Exception Info: These bits contain the exception information specific to the instruction. The helper core is responsible for exception detection, but the faulty core handles in during the retirement stage.

In addition, each entry needs to have bits identifying the source core, in which the instruction was originally issued, and the destination core, which is going to execute the instruction.

The reset valid bit signifies that the slot is empty and can be used. When a faulty core schedules an instruction to the ICQ, the source and destination core fields, valid bit and the execution bits are updated. We propose a push-pull scheme for scheduling the head of the ICQ to a helper core. When an instruction is ready in the ICQ to be serviced by the helper core, a bit in the decode unit of the helper core is set to convey the information. Subsequently, at each cycle, after its native instructions are scheduled, the decode unit looks for an empty slot to schedule this instruction. If the decode unit finds a slot it pulls the instruction from the ICQ to its pipeline. On the other hand, if the helper core does not pull the ICQ, the emergency bit is set after a specified maximum wait period called idling interval. A hardware counter is used to count the number of cycles the instruction spends in the ICQ, and the emergency bit is set after the idling threshold is crossed. Once the emergency bit for this instruction is flagged in the ICQ, the instruction is given higher priority than the native instructions, and is pushed into the helper core pipeline before any additional native instructions are processed. Once the helper core is ready to retire this instruction, the ICQ is updated with the result and the Executed bit is set. Any exception detected during the execution of this instruction by the helper core is also updated in the ICQ.

Two important design parameters involving the ICQ are the depth per core and the idling interval. The depth per core refers to the number of instructions from each core that simultaneously resides in the queue. Increasing the depth is expected to improve performance of the faulty core for workloads that have clusters of instructions requiring the use of the faulty resource. However the area overhead will also increase. The idling interval, on the other hand, is expected to have a bearing on the performance of the helper core. Higher values for the interval would mean less frequent force-through from the ICQ, possibly leading to lower performance degradation for the helper core. The advantage is expected to be more pronounced if the helper core also faces a strong demand for the shared functional units from its native instructions.

The ICQ forms a critical component of the scheme. Hence it has to be protected by redundancies. Fortunately the regular structure of buffers means they can be provided effective protection with low hardware overhead [26].

B. Faulty Core

During the decode stage of the pipeline, a parallel lookup of the hardware fault table identifies whether an instruction requires the use of a faulty unit or not. When an instruction requires the use of a faulty unit, a flag, called the migration bit, is set in the in the control store entry for that instruction to ensure proper control flow in the subsequent pipeline stages. Then the faulty instruction is allowed to flow through the pipeline. The schedule unit dispatches the instruction to the reservation station (RS) from the

fetch queue when it finds an empty reservation station and an empty slot in the Reorder Buffer (ROB). When the instruction reaches the head of the ROB, all dependencies are resolved and operands are available. Usually, the instruction is now ready to be retired. However, if the migration bit is set, the instruction is scheduled to the IC queue if an empty slot is available. Otherwise, the instruction waits in the ROB for an IC queue slot. After the instruction is sent to the ROB, the IC queue is polled for results and exception information. When the instruction is marked executed in the IC queue it is de-allocated and updated in the ROB. Now, the commit unit handles the instruction commit to the architectural state. Exceptions are handled during this stage, depending upon the exception information received from the IC queue. The use of the migration bit in the control-store information and the exception bit in the IC queue preserves the speculative and precise interrupt behavior of the processor. An alternative mode of execution would be to send the instruction to the IC queue as soon as all the source operands are available. However such a scheme would cause speculative instructions to be sent to the IC queue and executed in the helper core. Sending instructions from the head of ROB prevents the use of the helper core for speculative and potentially futile instructions.

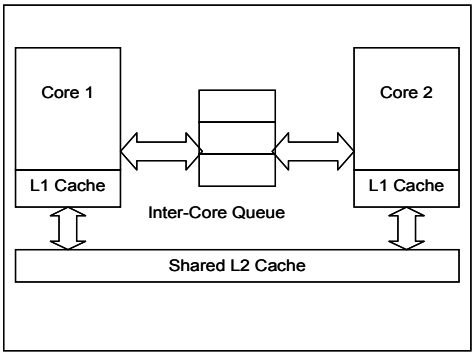

Figure 1. Inter-Core Queue: A Logical View

C. Helper Core

An instruction is either pushed or pulled by the helper core as described before. In either case a control store entry is created for proper control flow for this instruction in the subsequent stages of the pipeline. When the instruction is sent to the schedule unit, the operands are pulled from the ICQ. We note that the operands can also be pulled by the issue unit or execution units, wherever the critical path is mitigated. Once the instruction completes execution, the results and any exception detected during the instruction execution in the helper core are written back to the ICQ. The executed-bit in the ICQ for this instruction is also set.

The data flow of the instructions through the faulty core and the helper core is shown [Figure 2]. A single ICQ can be used for transferring instructions from each of the two cores to the other one as required. The change in the data-flow is constrained within the pipeline, and introduces no data consistency problems in the architectural state of the system.

D. Overhead

This scheme certainly entails some overhead in terms of area and complexity. The additional hardware for incorporating this scheme involves the following:

- ICQ and the buses connecting the ICQ to each core,
- Extra complexity in the control and synchronization logic of the cores to control the migration of instructions,
- A couple of bits in the control-store entry for an instruction, and
- Hardware fault-map and associated wires to read and write the map.

Compared to the area of a dual-core multi-processor, the area overhead is quite low. The ICQ needs to be placed symmetrically between two cores to ensure equal in-flight time for instructions between the queue and the cores. This places an additional constraint on the layout of the chip. The controller design complexity is also increased, which will require some extra design and test effort.

IV. Architectural Pre-requisites

In order to make use of the proposed micro-architectural modification for yield and reliability enhancement, the processors must be able to execute the following functions correctly:

- Online detection of hard faults,
- Diagnosis of the faulty unit, and
- De-configuration of the faulty unit.

Incorporating such fault awareness requires non-trivial modifications to the system hardware. An existing scheme that can be used for this purpose is the hard-fault detection and diagnosis framework described in [27], involving a low-cost hardware checker [28] and saturating counters. We provide a brief outline of the methodology in [27] for the sake of completeness. We have no contributions towards this end.

Detection: De-configuration of faulty units requires detection and diagnosis of faults. Such detections can be performed during periodic health check involving built-in self-test (BIST) or redundant multi-threading or online detection such as DIVA [28]. Discussions on these are out of scope here.

Diagnosis: In order to identify and diagnose hard faults, sub-structures in the cores that we wish to isolate and de-configure are classified as field de-configurable units (FDUs). Additional bits in the instructions are used to track FDU usage by an instruction. If an instruction result is found to be erroneous, the faulty FDU in use is recorded by incrementing a saturating counter corresponding to that FDU. If an FDU is found to be in error more than a pre-specified number of times within a pre-specified time interval, the fault in that unit is considered to be permanent [27].

De-configuration: There are several ways to de-configure a faulty unit [27]; the one suitable for our scheme involves maintaining a hardware fault-table of the FDUs. There has to be one entry for each FDU, containing its operational health information. For many-core systems, the table can be extended to a fault-map, mapping the helper cores to be accessed for each FDU. This table will be updated online depending upon the entries in the saturating counters. For yield enhancement purposes, the fault-table can be initialized offline during pre-shipment testing to de-configure any faulty FDU.

V. Simulation Framework

We use our modified version of SESC execution-driven cycle-level MIPS simulator to model a dual-core CMP with proposed modifications [29]. Support for chip multiprocessing in SESC

made it possible for us to implement process scheduling for multi-programmed workloads on the CMP.

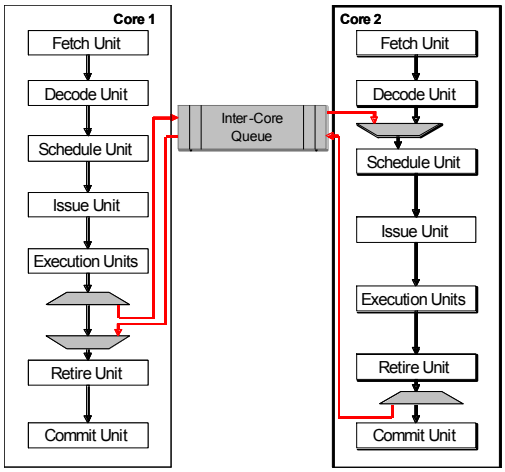

Figure 2. Data-Flow with Remote Execution

A. *Processor Configuration*

The processor we model here is a 90 nm 32-bit symmetric dual-core CMP. Each core is a four-way speculative out-of- order superscalar running at 3GHz frequency. Relevant system parameters are summarized in Table 1. In simulation, we model one or both cores as being damaged permanently. Since we are concentrating on high-area, high-latency and low-utilization units in our discussion, we model one of the floating-point ALU, multiplier, and divider units as the faulty unit in each damaged core.

B. *Workloads*

For any simulation run, we combine two benchmarks to form a multi-programmed workload, and then spawn the threads separately on two cores. We classify the benchmarks used according to the proportion of floating point instructions contained in them. SPEC2000 benchmarks *equake* and *gcc* are picked with low floating-point instruction count, and *flops* and *fbench* with high floating-point instruction count. We combine these to form an appropriate mix that is interesting for our analysis, as shown in Table 2 for a single faulty core. These combinations form a representative set of the workloads that the cores can face with respect to floating point intensity. For each workload, we set each of the FP ALU, multiplier and divider units as faulty and measure the performance loss in the degraded system compared to a fault-free system. In addition we vary the two design parameters, the ICQ depth and the maximum idling interval and analyze their impact on the performance loss. We also record the percentage of instructions that go through the ICQ and exceed the maximum idling interval. We run one billion instructions across both cores after fast-forwarding the initial two billion instructions in each core. The performance of each core is measured based on number of instructions issued per cycle (issued IPC). Hence an instruction issued in a faulty core and served by a helper core will be counted in the IPC of faulty core. The IPC for the helper core reflects its performance in executing its native thread only.

VI. RESULTS AND ANALYSIS

It has been noted earlier that when a unit of a core becomes faulty, functioning neighboring core helps with the execution. This may lead to performance degradation for both the cores. In Figure 3 and Figure 4 we show simulation results for selected workloads illustrating the performance degradation when any one of the cores is faulty. The Y-axis shows the relative performance of each core compared to fault-free situation when no neighborly help is sought. The performance varies with depth of ICQ, type of the faulty unit, and nature of the workload. For example, if floating-point unit is defective, it is more likely to impact performance of a floating-point intensive program. The X-axis in Figure 3 represents the faulty unit type and the depth of ICQ. The depth of the ICQ was varied from 2 to 20 entries per core, keeping the idling interval constant at 5 cycles. The X-axis in Figure 4 represents the idling interval after which an instruction forces its way through. The idling interval was varied from 2 to 10 cycles for constant ICQ depth of 10. In both cases, the results were more-or-less consistent for the static parameter (idling interval or ICQ depth), so we only show results for a single constant variable. The performance of an *infinite depth* ICQ was also studied. However, it was found that the performance improvement obtained from increasing the depth tends to saturate at a value around 20. Hence we report result up to depth 20 only. Figure 5 on the other hand, shows results when both cores have different faulty units, so that both the cores have to utilize the other core simultaneously and the flow of instructions through the ICQ occurs both ways. Here the performance improvement with ICQ depth saturated at a depth of 40 instructions in the worst case. The X-axis represents the depth of ICQ for the various combinations of faulty units in both cores, and the Y-axis denotes the relative performance of the cores. Table 3 contains the various combinations of faulty units used in simulation.

TABLE I. CMP CONFIGURATION

Individual Cores	
Fetch Width	4
Issue Width	4
Retire Width	4
FP FU Latency	ALU:1, Mult:6, Div:12
Integer FU Latency	ALU:1, Mult:4, Div:12
# Ld St Units	2
# FP Units	1 each (ALU, Mult, Div)
# Integer Units	2 each (ALU, Mult, Div)
ICQ Access Latency	2 cycles
Memory Configuration	
L1 I Cache (private)	64 Kb, 4-way, WB
L1 D Cache (private)	64 Kb, 4-way, WB
L2 (shared)	8 Mb, 8-way, WB

Workload equake-gcc: For this workload, for a faulty FP-ALU, less than 1% of the fetched instructions are switched from the faulty core to the helper core, while the helper core has no floating-point instructions of its own. The idling interval has consistently shown to have no impact on performance. This is of particular interest when the helper core is running a critical thread region and would incur a wait period to service remote instructions. As expected for this workload, system performance is similar in presence of a single faulty core or two simultaneous faulty cores.

Workload flops-gcc: Here the ICQ Depth is found to be quite dominant in terms of performance impact. For faulty FP-ALU unit, the faulty core used a helper for about 14% of the issued instructions. Varying the ICQ depth from 2 to 20, the faulty core performance loss improved from 75 to 12%. Similar results are seen for a faulty FP-Multiplier unit that has approximately 12% of the issued instructions sent to helper core for execution (67 to

11%). In case when the FP-Divider is not working, about 2% of the instructions are sent to the helper core. The worst and best case degradations are 30% and 10% respectively. There is no impact of idling interval on the faulty-core performance. The monotonic improvement in performance of the faulty core with increase in ICQ depth can be seen in Figure 3.

In this workload, the helper core had no native floating-point instructions. Hence there was no contention for the floating-point execution units. The base-case IPC for the 4-way helper core is only around 0.95, which means the schedule and issue units are also utilized only partially, primarily due to the lack of instruction-level parallelism in its native thread. Hence these units have enough free resources available to serve any foreign instruction that is injected. Almost all switched instructions were served within the idling interval and very few had to be forced through the helper core. Instead the helper core actually observes 2-5% performance improvement. This apparent oddity is due to the nature of the simulator. The simulator actually stops execution when the sum of fetched instructions in both cores equal the specified number. Hence while the faulty core incurred more dead cycles due to extra latency of executing faulty instructions and executed lesser instructions, the helper core fetched and executed more instructions, thus changing its native workload profile slightly.

When both the cores have faulty units (Figure 5), core 1 running the floating point intensive benchmark *flops* sees marked performance improvement with increase of the ICQ depth, and the improvement saturates at a depth of 30. The recovery is better for a faulty fp-divider than for a fp-ALU because of lower demand on the divider unit. Core 2, which executes the low-intensity benchmark *gcc* recovers the performance loss almost entirely at a depth of around 10.

TABLE II. WORKLOADS

Workload		**FP instruction intensity**	
Faulty Core	***Helper Core***	***Faulty Core***	***Helper Core***
equake	gcc	Low (0.3%)	Low (0.0%)
flops	gcc	High (27.5%)	Low (0.0%)
flops	fbench	High (27.5%)	High (18.5%)

Workload flops-fbench: This mix of floating-point intensive applications represents the worst-case combination that the system can face since both faulty and helper cores have significant floating-point load. Although the percentage of instructions switched remains same as the previous case, the best-case degradation achieved goes down to from 12 to 16% for faulty FP-ALU unit, and from 11 to 15% for FP-Multiplier unit. Results for the FP-Divider were similar to the previous case (see Figure 3). When both the cores have faulty units, there can be a permanent performance degradation of around 10% in both the cores in the worst case, and the improvement saturates at a higher ICQ depth of 40. Since there was no significant performance improvement with variations in the maximum idling interval, the results for varying the idling interval for two simultaneous faulty cores were not shown here.

VII. CONCLUSION

Multi-core processors have inherent redundancy in them. In this paper, a micro-architectural technique was proposed to exploit such redundancy for salvaging yield and improving reliability. Central idea was to implement an inter-core queue to seek execution help from functioning neighboring cores. The resulting design changes are minimal and impose negligible cost in terms of area and power. Simulation shows that significant yield recovery is possible with only 10-15% performance degradation in the worst case. In future we plan to investigate the scalability of the scheme for many-core systems requiring the use of multiple queues. The proposed scheme is useful for high-latency instructions that are executed sparingly. Hence we are looking at incorporating alternative in-core execution mechanisms for instructions that require very few clock cycles and are used frequently.

REFERENCES

[1] Guardiani, C., Bertoletti, M., Dragone, N., Malcotti, M., and McNamara, P. 2005. An effective DFM strategy requires accurate process and IP pre-characterization. In *Proceedings of the 42nd Annual Conference on Design Automation, 2005*.

[2] Shyam, S., Constantinides, K., Phadke, S., Bertacco, V., and Austin, T. 2006. Ultra low-cost defect protection for microprocessor pipelines. SIGPLAN Not. 41, 11 (Nov. 2006), 73-82.

[3] Mitra S., Agarwal M., "Circuit failure prediction to overcome scaled CMOS reliability challenges," International Test Conference, 2007

[4] http://www.itrs.net/Links/2007ITRS/Home2007.htm

[5] Borkar S. Y., "Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation,"*IEEE Micro*,Vol.25,Issue.6,pp.10-16,Nov.-Dec. 2005.

[6] Carulli J. M., and Anderson T.J., "Test Connections – Tying Application to Process," *Proc. Intl. Test Conf.*, pp. 679-686, 2005.

[7] Van Horn J., "Towards Achieving Relentless Reliability Gains in a Server Marketplace of Teraflops, Laptops, Kilowatts, & "Cost, Cost, Cost" ... (Making Peace between a Black Art and the Bottom Line)," *Proc. Intl. Test Conf.*, pp. 671-678, 2005.

[8] Zafar, S., "A Model for Negative Bias Temperature Instability in Oxide and High-K pFETs," Integrated Circuit Design and Technology, 2007. IEEE International Conference on, pp.1-5, May 30 2007-June 1 2007.

[9] Denais M., Huard V., Parthasarathy C., Ribes G., Perrier F., Revil N., and Bravaix A., "Interface Trap Generation and Hole Trapping Under NBTI and PBTI in Advanced CMOS Technology With a 2-nm Gate Oxide," IEEE Transactions on Device And Materials Reliability, vol. 4, no. 4, December 2004.

[10] Wu E. Y., Abadeer W. W., Han L.-K., Lo S.-H., and Hueckel G., "Challenges for accurate reliability projection in the ultrathin oxide regime," in Proc. IRPS, p. 57, 1999.

[11] Stathis J.H., Reliability limits for the gate insulator in CMOS technology, *IBM Journal of Research and Development* 46 (2/3), pp. 265–286, 2002.

[12] Srinivasan, J.; Adve, S.V.; Bose, P.; Rivers, J.A., "The impact of technology scaling on lifetime reliability," Dependable Systems and Networks, 2004 International Conference on, pp. 177-186, June- July 2004.

[13] Groseneken G., Degraeve R., Kaczer B., and Rousel P., "Recent Trends in Reliability Assessment of Advanced CMOS Technology," Proceedings of IEEE 2005 International Microelectronics Test Structure, vol. 18, April 2005.

[14] Choi K., Soma R., and Pedram M., "Fine-Grained Dynamic Voltage and Frequency Scaling for Precise Energy and Performance Trade-off based on the Ratio of Off-chip Access to On-chip Computation Times," Proc. of Design Automation and Test in Europe, Feb. 2004.

[15] He C., Jacome M. F., and G. de Veciana, "A reconfiguration-based defect-tolerant design paradigm for nanotechnologies," IEEE Design & Test of Computers, pp. 316-326, July-Aug.2005.

[16] Wang T., Qi Z., and Moritz C. A., "Opportunities and challenges in application-tuned circuits and architectures based on nanodevices," in Proc. International Conference on Computing Frontiers, pp. 503-511, Apr. 2004.

[17] Koren I. and Koren Z., "Defect Tolerant VLSI Circuits: Techniques and Yield Analysis," Proc. of the IEEE, Vol.86, pp.1817-1836, Sept. 1998.

[18] Joseph R., "Exploring Salvage Techniques for Multi-core Architectures," HPCRI-2005 Workshop in Conjunction With HPCA-2005, February 2006.

[19] Shivakumar P.; Keckler, S.W.; Moore, C.R.; Burger, D., "Exploiting microarchitectural redundancy for defect tolerance," Computer Design, 2003. Proceedings. 21st International Conference on, vol., no., pp. 481-488, 13-15 Oct. 2003.

[20] Srinivasan, J.; Adve, S.V.; Pradip Bose; Rivers, J.A., "Exploiting structural duplication for lifetime reliability enhancement," Computer Architecture, 2005. ISCA '05. Proceedings. 32nd International Symposium on, pp. 520-531, 4-8 June 2005.

[21] Bossen D. C., Kitamorn A., Reick K.F., and Floyd M.S., "Fault-tolerant design of the IBM pSeries 690 system using POWER4 processor technology," IBM Journal Of Research and Development, vol. 46, no. 1, January 2002.

[22] Tendler J. M., Dodson J. S., Fields J. S. Jr., Le H., and Sinharoy B., "POWER4 System Microarchitecture," IBM Journal Of Research and Development, vol. 46, no. 1, January 2002.

[23] Stapper, C. H. 1993. Improved Yield Models for Fault-Tolerant Memory Chips. IEEE Trans. Comput., vol. 42, no. 7, pp. 872-881, July 1993.

[24] Leveugle, R.; Koren, Z.; Koren, I.; Saucier, G.; Wehn, N., "The Hyeti defect tolerant microprocessor: a practical experiment and its cost-effectiveness analysis," Computers, IEEE Transactions on, vol.43, no.12, pp.1398-1406, Dec 1994.

[25] Dolbeau R., Seznec A.: "CASH: Revisiting Hardware Sharing in Single-Chip Parallel Processors", J. Instruction-Level Parallelism vol.6, 2004.

[26] Bower, F.A.; Shealy, P.G.; Ozev, S.; Sorin, D.J., "Tolerating hard faults in microprocessor array structures," Dependable Systems and Networks, 2004 International Conference on, pp. 51-60, June-July 2004.

[27] Bower, F.A.; Sorin, D.J.; Ozev, S., "A mechanism for online diagnosis of hard faults in microprocessors," Microarchitecture, 2005. MICRO-38. Proceedings. 38th Annual IEEE/ACM International Symposium on, pp. 12-16 Nov. 2005.

[28] Austin T. M., "DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design," Proc. of the 32nd Annual International Symposium on Microarchitecture, pp.196-207, Nov. 1999.

[29] R Renau J., *et al.* SESC Simulator, January 2005. http://sesc.sourceforge.net

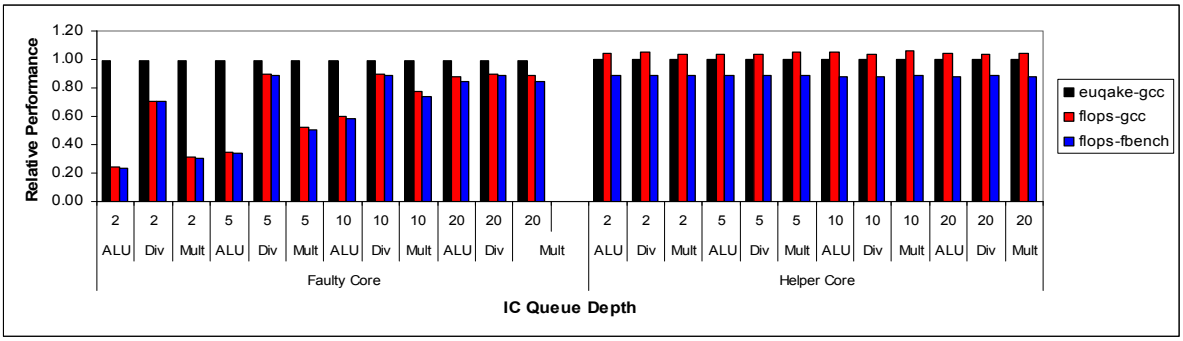

Figure 3. Relative Performance Variation with ICQ Depth (Idling Interval = 5 cycles)

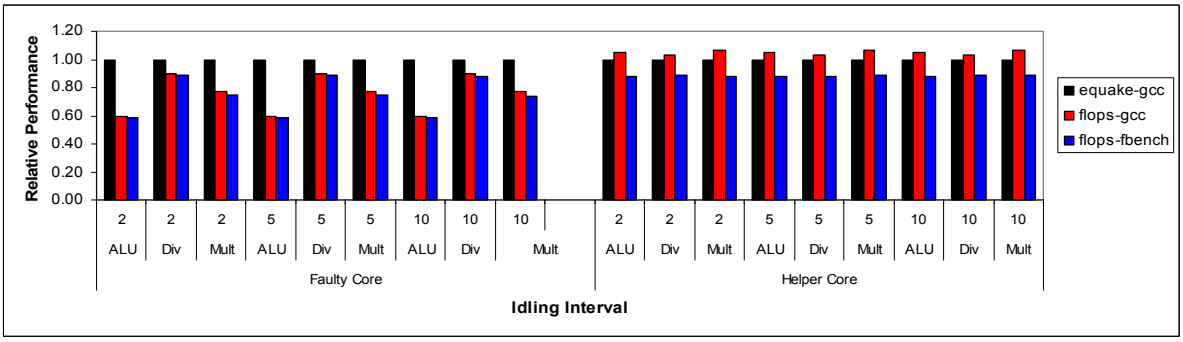

Figure 4. Relative Performance Variation with Idling Interval (Queue depth = 10)

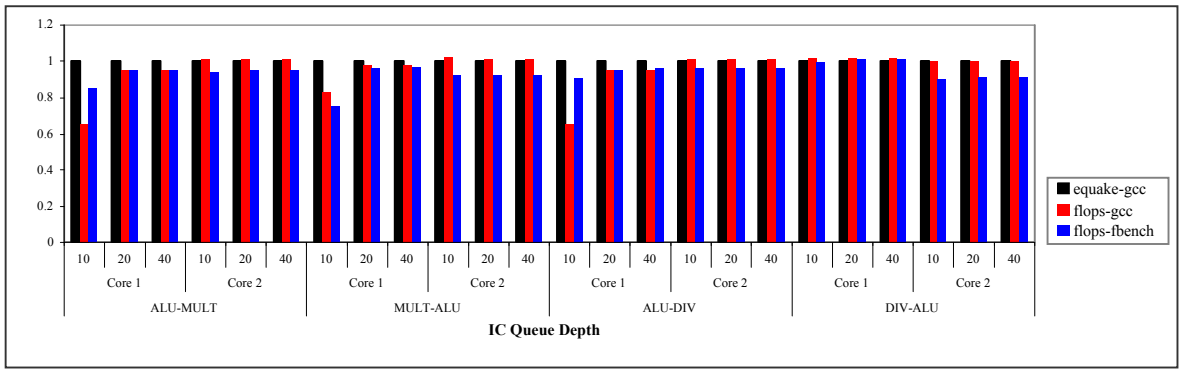

Figure 5. Relative Performance Variation with ICQ Depth: Both Cores Faulty (Idling Interval = 5 cycles)