

# Error Resilient System Architecture (ERSA) For Probabilistic Applications

Jason Bau<sup>1</sup>, Richard Hankins<sup>2</sup>, Quinn Jacobson<sup>2</sup>, Subhasish Mitra<sup>1</sup>, Bratin Saha<sup>3</sup>, Ali-Reza Adl-Tabatabai<sup>3</sup>

**Abstract**—Probabilistic applications such as data mining, image recognition and case synthesis (often referred to as Recognition, Mining and Synthesis or RMS), constitute a class of emerging applications expected to drive future demand for computational capacity. This paper describes the design and prototyping of a low-cost *Error-Resilient System Architecture (ERSA)*, specifically targeted for these applications, featuring a combination of cheap, unreliable compute power together with a small fraction of reliable processor cores for running system software, controlling application flow, and recovering from errors generated on unreliable cores. ERSA enables economical system designs that are resilient to very high error rates, beyond current radiation-induced soft error levels, caused by highly frequent erratic intermittent errors, process variations and transistor aging. Experimental results on a prototype system with representative probabilistic computation applications show that this architecture is resilient to error rates on the order of trillion-plus failures-in-time (FITs, defined as the number of failures in  $10^9$  hours), while still delivering robust results with competitive performance.

**Index Terms**—Probabilistic Applications, Error Resilient Architecture, Fault Recovery

## I. INTRODUCTION

In future scaled CMOS technologies, error sources such as radiation-induced soft errors, highly frequent erratic intermittent errors, process variations, and transistor aging [Agostinelli 05, Borkar 05, Gelsinger 06, Van Horn 05], will severely challenge the correct operation of the circuit components underlying a digital processors. As a result, processor designs for these scaled processes must correct or mitigate the occurrence of circuit errors. Unfortunately, classical techniques for error correction such as duplication, Triple Modular Redundancy (TMR), radiation hardening, and concurrent error detection are very expensive in terms of power and performance. Utilizing these costly approaches to exhaustively guarantee the reliability of every processor component and operation will significantly diminish the realized performance / power benefit of further scaling.

Hardware implementation results from the Error Resilient System Architecture (ERSA) presented in this paper demonstrate that the unique properties of emerging probabilistic applications, often classified into Recognition,

Mining, and Synthesis (RMS), may be successfully utilized to create robust systems resilient to error rates of the order of trillions of FITs without incurring the major costs associated with classical techniques.

Highly-parallelizable RMS applications, exemplified by probabilistic belief propagation, clustering and classification algorithms, form the basis of numerous emerging application domains such as cognitive systems, complex data set analysis, and bio-informatics and are widely believed to significantly influence future hardware designs by driving demand for computational capacity [Dubey 05, Gelsinger 05]. Previous work [Yu 00, Wong 06, Li 06, Chakrapani 06] and our results show (details in section 4) that a significant percentage of errors in the data-calculations of these applications have minimal impact on the overall quality of results. Thus, the applications have a degree of inherent resilience against data-errors that allows the reliability requirement of their calculation hardware to be relaxed.

Despite their data-error resilience, these applications are still susceptible to hardware errors causing faulty execution control resulting in crashes or non-termination [Wong 06, Saggese 05]. Recovery from these errors is the key functionality we provide in ERSA. ERSA complements RMS application characteristics by providing reliable architectural features that recover from errors in execution control and ensure successful application completion. This enables ERSA to exploit the data-error resiliency of its target applications by relaxing reliability requirements for the calculation hardware. With the ERSA architecture, we were able to achieve error-resilient execution of three sample RMS applications on an implemented prototype system under trillion-FIT system error-levels, with results and performance competitive with optimized baseline versions of the applications.

In section 2, we will discuss the general iterative structure of RMS applications for the purpose of mapping the application structure to the ERSA architecture. Section 3 will then detail the ERSA architecture and a ERSA prototype before section 4 presents experimental results from the prototype.

## II. RMS APPLICATION STRUCTURE

Most RMS applications may be characterized by their iterative nature. Figure 1 illustrates the control flow and structure of such applications. As shown in Fig. 1, the *main application thread* is responsible for initiating each iteration pass. In the beginning of each iteration pass, the main thread initiates numerous *worker threads*, and the *work assignments* for these threads are parallel *data-calculations*. For example,

This work was supported in part by the Gigascale Systems Research Center and Intel Corporation.

<sup>1</sup>Computer Systems Laboratory, Stanford University, Stanford, CA.

<sup>2</sup>Nokia Corp., Palo Alto, CA. This work was performed while the authors were at Intel Corp.

<sup>3</sup>Intel Corp., Santa Clara, CA.

in the k-means clustering algorithm [MacQueen 67], each worker thread averages a subset of data-points according to their assigned clusters to find the subtotal-centroids of each cluster. During this time the main thread waits at a *barrier* for completion of the worker threads. Once the worker threads notify the main thread of their completion, the main thread performs a *reduction* step to collect each thread’s results. For example, the main thread will calculate a weighted average of the centroid subtotals to calculate final centroids for the k-means clustering example. After the reduction step, the main thread performs *condition testing* to determine whether to invoke another iteration or not. If it decides to invoke another iteration, it again initiates work to the worker threads, and the results obtained from the previous iteration are used by the worker threads to calculate the next iteration.

This structure conveniently separates RMS application tasks into resilient or non-resilient. The data calculations performed by worker threads are resilient to errors, usually because errors average out over multiple iterations and over redundant data, while errors in the tasks performed by the main threads can cause catastrophic errors such as crashes or infinite loops. Thus, the ERSA architecture provides both strict and relaxed reliability hardware for software execution as well as mechanisms for segregating the software to each type of hardware.

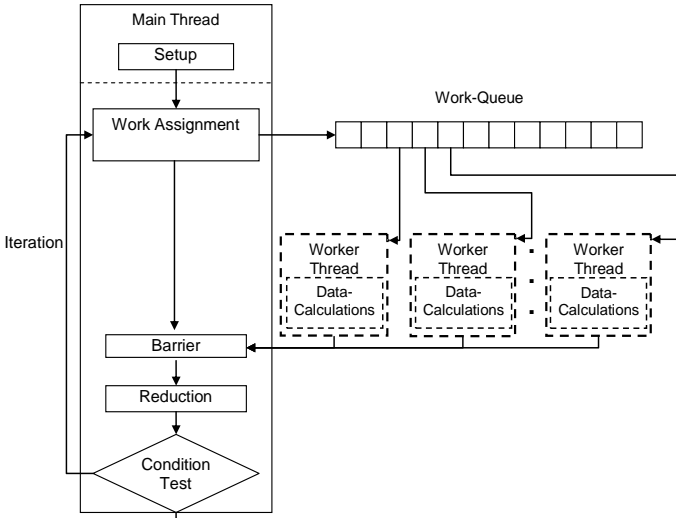


Fig 1: Structure of typical RMS applications

### III. THE ERROR RESILIENT SYSTEM ARCHITECTURE

ERSA is an asymmetric multi-core architecture consisting of both reliable and unreliable cores. The unreliable cores, called the Relaxed-Reliability Cores (RRCs), are high performance, low power, unreliable cores used for running error-resilient algorithms. In contrast the Strict Reliability Cores (SRCs) are for running those operations that cannot tolerate errors; for example, system critical operations, such as the operating system the application’s main thread. Figure 2 illustrates a simple eight-core ERSA processor, with one SRC and seven RRCs. Note that the caches and the interconnect fabric of the entire ERSA processor are assumed to be reliable, because

several techniques such as robust circuit designs, error correcting codes (ECC) and cyclic redundancy checks (CRC) are available for protecting them at a low cost.

Application execution occurs on the RRCs only if explicitly directed to do so by the application. ERSA provides two architectural enhancements for isolating RRC errors from affecting the application. The first protection mechanism allows the application program to bound the virtual memory accesses for each user-level thread executing on an RRC. The RRC’s memory-management logic then enforces these bounds and issues a memory-access exception if they are violated. The second mechanism allows the application to limit user-level thread execution time on the RRC.

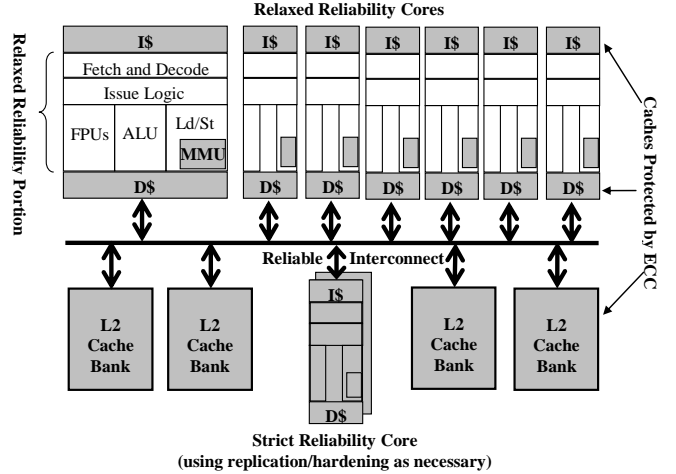


Fig 2: Illustration of ERSA Architecture

#### A. Strict-Reliability Cores (SRCs)

The SRCs are constructed to provide entirely reliable operation. Reliable designs are expensive but feasible through methods such as pessimistic design (often called overdesign) and special circuit designs that suppress the occurrences of errors at the circuit level (e.g., DIVA [Austin 99], Razor [Ernst 03], BISER [Mitra 05]), tightly-coupled fine-grained duplication and transparent instruction retry [Spainhower 99] or lockstep execution among others. Hence, we limit the number of SRCs to a small fraction of the total number of cores in a chip (a 1-to-7 ratio on our prototype, and an interesting optimization problem in general). On the prototype strict-reliability is achieved by suspending error-injection on this core.

The SRCs are responsible for:

- a) Executing operations that are not resilient to errors. This includes operating system software and, by extension, the main thread of any application.
- b) Delegating work to the RRCs via worker-thread creation and specification of worker-thread virtual address space limits and timeout information to the RRC.
- c) Receiving and handling any exception occurring on the RRC. These exceptions include attempts to access memory addresses outside of the permissible bounds, normal page faults, and attempts to execute system level code

- d) Executing reduction and condition-check application tasks after worker threads return at the iteration-end barrier.

On the ERSA prototype, b) above was accomplished by extending the inter-processor communication SIGNAL instruction from the Multiple-Instruction Stream Processor (MISP) architecture [Hankins 06] to provide SRCs the following management powers over RRCs: execution suspension, execution resumption in-place or at a new PC value, specification of base memory address and bounds, specification of execution duration for the watchdog timer, and readout and setting of architectural and register state.

Also, once a user-level thread has been created on the RRC, the SRC's microarchitecture listens for any exceptions on the RRC. Upon detection of such an event, the SRC and immediately redirects control flow into an application-specified software exception handler. This software handler interprets the exception event and then makes a decision on how to handle it. Further details on exception handling are described in 3.3

### B. Relaxed-Reliability Cores (RRCs)

RRCs make up the large majority of on-chip cores and provide the application an abundant supply of inexpensive compute power. RRCs are kept invisible to the operating system and must be enabled by the main thread running on the SRC. Most hardware units of the RRC are assumed to be error-prone at trillion-to-quadrillion FIT rates; this is emulated on the prototype by error-injection into general-purpose and floating-point registers. A few units on the RRC need to be reliably designed for error isolation and recovery purposes. These units include:

- The memory management unit (MMU). The MMU must accurately detect violations of the RRCs virtual address space limits. Exceptions must be reliably raised when an offending access is attempted. This isolation is essential for both application correctness and system security. Since a majority of MMU circuitry is in the form of tables that can be protected by coding techniques at low cost, this requirement only covers a small portion of MMU logic and is not onerous in cost.
- The cache subsystem. RRCs are assumed to reliably maintain correct coherency state as well as the integrity of memory data. An error in the coherency unit would be potentially fatal to the entire system. As aforementioned, reliable caches are already currently being constructed with inexpensive coding techniques.

By maintaining the reliability of the MMU and cache subsystems of the RRC, ERSA preserves the integrity of the shared-memory architecture fundamental to multi-processor systems. No data located at an address intended to be secure can be corrupted, and while cache and memory locations can be written with incorrect values, these are properly shared across cores. This enables the SRCs ability to monitor and control execution progress on the RRCs.

### C. Handling RRC Exceptions

The MISP architecture includes a proxy execution

mechanism that allows one core to execute on behalf of another. We extend that mechanism to handle exceptions originating on the RRCs. Upon detecting an exception on the RRC, such as from an illegal memory-access or timeout exception, the SRC is immediately redirected to execute an application-defined software exception handler, to perform recovery. The handler code is responsible for interpreting the exception, and determining whether to restart the RRC, as in the case for a timeout or memory-bounds violation, or proxy-execute the faulting instruction, as in the case of a page fault. It also makes application-level decisions about the worker-thread assigned to the faulty RRC and can choose to roll-back data and re-issue the worker thread or to accept the best-effort results of the worker thread and move on to new work-assignment. After the exception has been handled, the RRC is either restarted or resumed from its current suspended state. The SRC then resumes its own execution from the point where it received the RRC's exception.

## IV. HARDWARE PROTOTYPE AND EXPERIMENTAL RESULTS

The ERSA prototype hardware is a physical 4-socket SMP system populated with 4 dual-core IA-32 processors, each clocked at 2.67 GHz, with 2GB of system main memory.

### A. Error Injector

All of the error-injection experiments were performed by flipping bits in actual hardware registers. A separate injection-initiation thread runs simultaneous to the application on the SRC. At specified periods, this thread utilizes a CPU virtualization layer, implemented by the MISP infrastructure [Hankins 06] to pause application execution on the prototype CPUs representing the RRCs, flip hardware bits in their general-purpose registers or floating-point registers as appropriate, and then resume application execution. Thus the hardware error injections occur asynchronous to the application.

### B. Example Applications

We selected three major representative applications from the RMS domain as the target workload for our prototype.

a) Low-Density Parity Check Decode: Probabilistic belief propagation [Pearl 88] is a method for making inferences on Bayesian networks that has broad applicability in the fields of computer-vision, bio-informatics, and expert-systems. One of its instances is the decoding of Low-Density Parity Check codes [Gallagher 63], an important class of codes in the communications field that performs the forward error-correction (FEC) technique. We chose this application because there is a clear criterion for correctness of results unlike applications such as data mining where the quality of results may be subjective. Details of LDPC decoding algorithms can be obtained from [MacKay 03] and [Neal 06].

b) K-Means Clustering: K-means clustering [MacQueen 67] is the most classical algorithm that organizes data points into clusters based on proximity, and finds applicability in fields such as bio-informatics and market research. This algorithm

was chosen because it offers an objective goodness measure of the outcome based on the tightness of the resultant clusters.

c) Bayes-Net Structure Learning: The Bayes-net structure learning algorithm is a greedy “hill-climbing” algorithm that successively adds edges to a Bayes-net by trying to optimize a scoring function. Details are available at [Chen 05]. Our example network attempts to learn the influence strengths of various loci in single nucleotide polymorphisms.

### C. Application Data Error Injection

Both LDPC-decode and K-means clustering double-length IEEE floating point numbers to represent data, so we tested their data error-resiliency characteristics by injecting bit-flips of various significance during application runs into the RRC floating-point registers.

Figure 3 presents the results for LDPC-decode. The error resilience metric for our LDPC decoder is the number of blocks correctly decoded. Low-significance data error injections, below the 2<sup>nd</sup> exponential bit (bit 53), have no significant effect on final decode output even when injected at FIT rates in the order of  $10^{17}$ . Higher-significance exponential bits do impact performance at this very high FIT rate, though the impact is lessened at the  $10^{15}$  FIT rate. Fortunately, errors in the most significant bits are easy to detect via application invariant techniques and can be combined with simple checkpointing to provide correct results. Figure 3 shows the dramatic improvement to result quality that application invariant-based detection and simple two-phase checkpointing (iteration results stored into alternating sets) provided, with all cases now correcting decoding > 96% of the blocks.

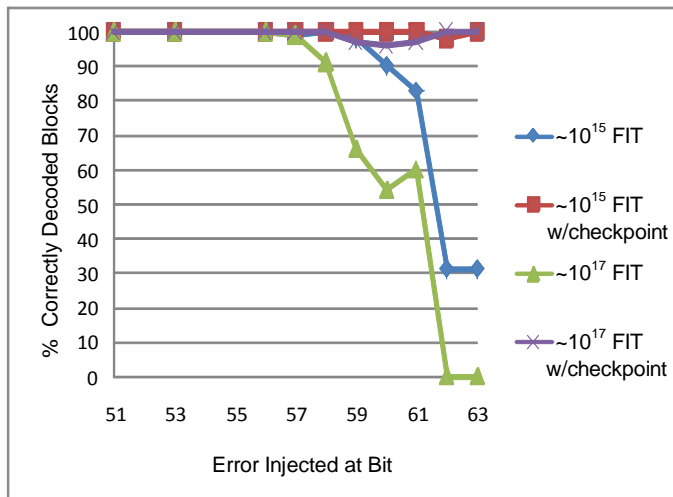


Fig 3: LDPC decode error injection results

Figure 4 below summarizes the experimental results for k-means clustering. The error-resiliency metric here measures the tightness of the resultant clusters by measuring the mean point-cluster centroid distance. We see very similar error resilience behaviors as LDPC-decode, with the low-significance bits below the 2<sup>nd</sup> exponential bit (bit 53) not degrading output quality more than 1% even at  $10^{17}$  FIT and the highest significance bits degrading output quality by less than 10% at  $10^{15}$  FIT. Again using application invariant-based

error detection and two-phased checkpointing for recovery, we were able to make all cases perform within 2% of baseline at  $10^{17}$  FIT and 0.2% at  $10^{15}$  FIT.

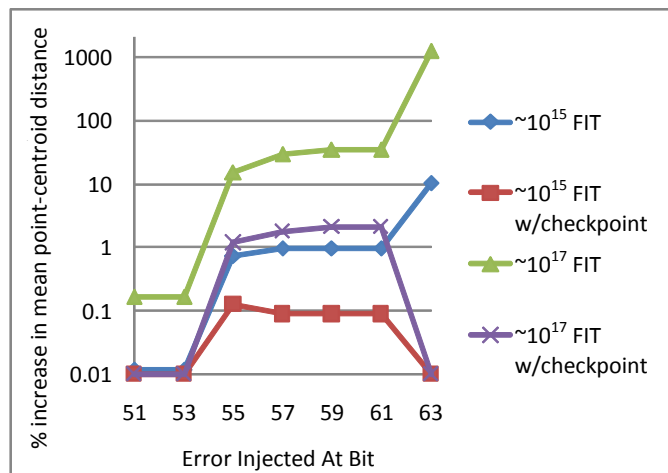


Fig 4: K-means clustering error injection results

### D. ERSA Robustness and Performance

To demonstrate the robust execution of the ERSA architecture, we ported LDPC-decode, K-means clustering, and Bayes-Net learning applications over to ERSA hardware prototype, and conducted error injections on both floating-point registers and general purpose registers. Figure 5 measures the throughput of useful work produced by ERSA versions versus baseline non-ERSA versions (i.e., simply running these applications on error-prone hardware without the ERSA) in the presence of very high FIT error rates. Compared against baseline versions, we found that the ERSA versions of the applications retained the ability to produce useful work even under extremely high FIT rates which the non-ERSA version could not survive.

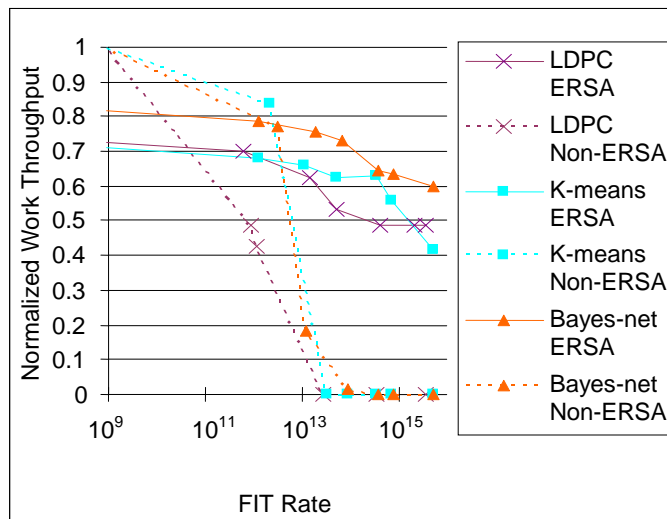


Fig 5: Effective throughput of ERSA and non-ERSA application modes. The throughput of each application is normalized against the non-ERSA, zero-FIT performance. LDPC work is measured in decoded codewords, K-means work in number of clustering runs, and Bayes-net work in number of graph edge manipulations.

We also measured the performance overhead of ERSA versions of LDPC and K-means against optimized baseline version. The resultant performance data is in figure 6. The overhead incurred by ERSA comes from the fine-grained worker-threads used by ERSA applications to provide recovery points, the latency of the error-handling callback and RRC restarts, and the additional instructions required to implement memory bounds on ported applications. This last source of overhead is an artifact of the prototype since the additional instructions may be completely removed with a customized compiler. The improvement from this removal is plotted under “potential resilient” We see that the final ERSA implementation costs about 30% overhead over the baseline version for all applications and may cost as low as 15% overhead with a customized compiler.

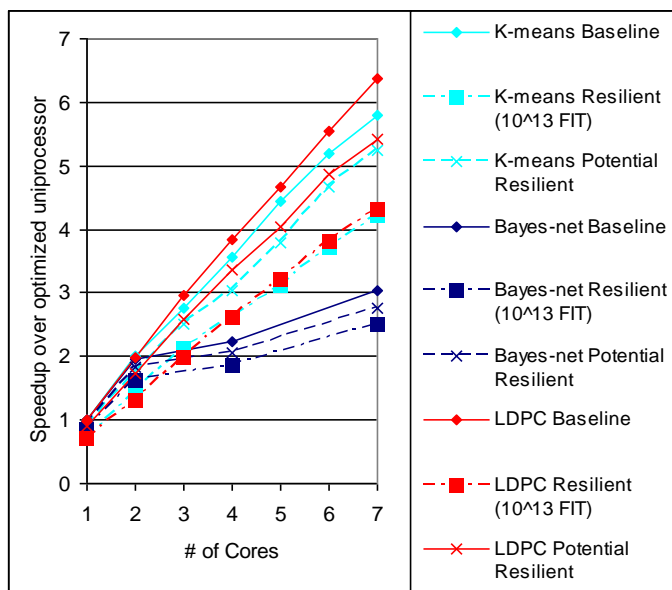


Fig 6: ERSA application performance versus baseline versions

## V. CONCLUSIONS

The Error Resilient System Architecture (ERSA) presented in this paper demonstrates that it is possible to utilize error resilience properties of future probabilistic killer applications for designing error-resilient systems on inexpensive hardware with very high error rates without incurring high costs associated with traditional redundancy techniques. As experimental results indicate, the ERSA hardware prototype is resilient to very high error rates of the order of trillions of FITs with competitive performance. ERSA’s tolerance for such high error rates extends far beyond radiation-induced soft error rates to also cover highly frequent erratic intermittent errors, process variations and transistor aging, making ERSA an effective design for garnering reliability and increasing power and performance gains in future scaled technologies.

## REFERENCES

[Agostinelli 05] Agostinelli, M., *et al.*, “Erratic Fluctuations of SRAM Cache Vmin at the 90nm Process Technology Node,” *Proc. Intl. Electron Devices Meeting*, pp. 655-658, 2005.

- [Austin 99] Austin, T., “DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design,” *Proc. Intl. Symp. Microarch.*, pp. 196-207, 1999.
- [Borkar 05] 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.
- [Chakrapani 06] Chakrapani, L.N., *et al.*, “Ultra Efficient Embedded SOC Architectures based on Probabilistic CMOS (PCMOS) Technology,” *Proc. Design Automation and Test in Europe*, 2006.
- [Chen 05] Chen, Y., *et al.* “Performance Scalability of Data-Mining Workloads in Bioinformatics.” *Technology at Intel Magazine*, May 2005.
- [Dubey 05] Dubey, P., “Recognition, Mining and Synthesis Moves Computers to the Era of Tera”, *Technology at Intel Magazine*, Feb. 2005.
- [Ernst 03] Ernst, D., *et al.*, “Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation,” *Proc. Intl. Symp. Microarchitecture*, 2003.
- [Gallager 63] Gallager, R.G., *Low Density Parity Check Codes*, Cambridge, MA: MIT Press, 1963.
- [Gelsinger 05] in “Recognition, Mining and Synthesis Moves Computers to the Era of Tera”, *Technology at Intel Magazine*, Feb. 2005.
- [Gelsinger 06] Gelsinger, P., “Into the Core...,” *Stanford EE Computer Systems Colloquium*, June 7, 2006, <http://www.stanford.edu/class/ee380/Abstracts/060607.html>.
- [Hankins 06] Hankins, R.A., *et al.* “Multiple Instruction Stream Processor”, *Proceedings of the 2006 International Symposium on Computer Architecture*.
- [Li 06] Li, X., D. Yeung, “Exploiting Soft Computing for Increased Fault Tolerance,” *Workshop on Arch. Support for Gigascale Integration*, 2006.
- [MacKay 03] MacKay, D.J.C., *Information Theory, Inference, and Learning Algorithms*, Cambridge University Press, 2003.
- [MacQueen 67] MacQueen, J.B. “Some Methods for classification and Analysis of Multivariate Observations,” *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*, Berkeley, University of California Press, 1:281-297.
- [Mitra 05] Mitra, S., N. Seifert, M. Zhang, Q. Shi and K.S. Kim, “Robust System Design with Built-In Soft Error Resilience,” *IEEE Computer*, Vol. 38, Number 2, pp. 43-52, Feb. 2005.
- [Neal 06] <http://www.cs.utoronto.ca/~radford/ldpc.software.html>
- [Pearl 88] Pearl, J., *Probabilistic Reasoning in Intelligent Systems*, Morgan Kaufmann, 1988.
- [Saggese 05] Saggese, G.P., A. Vetteth, Z. Kalbarczyk and R.K. Iyer, “Microprocessor Sensitivity to Failures: Control vs Execution and Combinational vs Sequential Logic,” *Proc. Intl. Conf. Dependable Systems and Networks*, pp. 760-769, 2005.
- [Spainhower 99] Spainhower, L., and T.A. Gregg, “S/390 Parallel Enterprise Server G5 Fault Tolerance,” *IBM Journal Res. and Dev.*, Vol. 43, pp. 863-873, Sept./Nov., 1999.
- [Van Horn 05] 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.
- [Wong 06] Wong, V., and M. Horowitz, “Soft Error Resilience of Probabilistic Inference Applications,” *Proc. Intl. Workshop System Effects of Logic Soft Errors*, 2006.
- [Yu 00] Yu, S.Y., N. Saxena and E.J. McCluskey, “An ACS Robotic Control Algorithm with Fault Tolerant Capabilities,” *Proc. Intl. Symp. Field-Programmable Custom Computing Machines*, 2000.