ERSA: Error Resilient System Architecture for Probabilistic Applications

Larkhoon Leem[1], Hyungmin Cho[1], Jason Bau[1], Quinn A. Jacobson[3], Subhasish Mitra[1, 2]
[1]Department of Electrical Engineering, [2]Department of Computer Science, Stanford University, Stanford, CA
[3]Nokia Research Center, Palo Alto, CA

Abstract

There is a growing concern about the increasing vulnerability of future computing systems to errors in the underlying hardware. Traditional redundancy techniques are expensive for designing energy-efficient systems that are resilient to high error rates. We present Error Resilient System Architecture (ERSA), a low-cost robust system architecture for emerging killer probabilistic applications such as Recognition, Mining and Synthesis (RMS) applications. While resilience of such applications to errors in low-order bits of data is well-known, execution of such applications on error-prone hardware significantly degrades output quality (due to high-order bit errors and crashes). ERSA achieves high error resilience to high-order bit errors and control errors (in addition to low-order bit errors) using a judicious combination of 3 key ideas: (1) asymmetric reliability in many-core architectures, (2) error-resilient algorithms at the core of probabilistic applications, and (3) intelligent software optimizations. Error injection experiments on a multi-core ERSA hardware prototype demonstrate that, even at very high error rates of 20,000 errors/second/core or $2x10^{-4}$ error/cycle/core (with errors injected in architecturally-visible registers), ERSA maintains 90% or better accuracy of output results, together with minimal impact on execution time, for probabilistic applications such as K-Means clustering, LDPC decoding and Bayesian networks. Moreover, we demonstrate the effectiveness of ERSA in tolerating high rates of static memory errors that are characteristic of emerging challenges such as Vccmin problems and erratic bit errors. Using the concept of configurable reliability, ERSA platforms may also be adapted for general-purpose applications that are less resilient to errors (but at higher costs).

1. INTRODUCTION

Reliability is a major concern for power-constrained computing systems in advanced CMOS technologies. Several mechanisms threaten to significantly increase the number of errors experienced by future systems – erratic bit errors, transient (soft) errors, early-life failures (infant mortality), transistor aging, Vccmin challenges, process variations and variations in operating conditions (e.g., voltage droops) [Agostinelli 05, Borkar 04, Van Horn 05]. To overcome these challenges, two classes of techniques have been employed in many systems:

1. Conservative design to ensure correct operation, e.g., guardbanding, conservative voltage scaling. Many of these techniques appear to be running out of steam [Gelsinger 06].

2. Fault-tolerant systems that detect and recover from errors through expensive redundancy.

The *Error Resilient System Architecture* (*ERSA*), presented in this paper, leverages two emerging trends in future computing platforms:

1. Proliferation of multi- and many-core systems.

2. New killer applications, e.g., data mining, market analysis, cognitive systems and computational biology, which are expected to drive demands for computation capacity. Such applications are also referred to as Recognition, Mining, Synthesis or *RMS* applications [Dubey 05].

Unique properties of RMS applications include:

1. **Massive parallelism**: Massive amounts of data are processed to build mathematical models and to apply models to help answer real-world questions.

2. **Algorithmic resilience:** Unlike conventional computing, RMS applications are tolerant to imprecision and approximation to make analysis of complex systems tractable. In addition, an iterative approach is often used to refine computation results such that the effects of inaccuracies can be reduced by subsequent iterations.

3. **Cognitive resilience:** Computation results need not always be correct as long as the accuracy of the computation is "acceptable" to human users [Breuer 05].

Several aspects of algorithmic and cognitive resilience of various applications to low-order data bit errors have been addressed previously by several researchers [Breuer 05, Chakrapani 06, Hayes 07, Li 06, Shanbhag 02, Yu 00, Wong 06]. While such data error resilience is clearly a win, it alone is not sufficient for probabilistic applications to converge and generate useful results when executed on unreliable computing hardware. At high error rates, the errors in high-order bits of data and application control flow significantly affect application performance (details in Sec.2).

ERSA uses the following techniques to overcome the above challenges of high-order bit errors and control errors:

1. *Asymmetric reliability,* i.e., mixing processor cores of various "reliability levels" in many-core architectures.

2. Software optimizations including minimally-intrusive yet effective modifications to RMS algorithms.

3. Light-weight checks such as timeouts and memory bounds violation checks. ERSA does not rely on expensive error detection techniques.

Error injections in actual ERSA hardware platforms demonstrate that, even at extremely high error injection rates of 20,000 errors/sec/core or $2x10^{-4}$ error/cycle/core into architecturally-visible registers, ERSA delivers RMS applications with the following characteristics:

1. No system crashes.

2. 90% or better accuracy of output results (within cognitive resilience limits as demonstrated using actual applications).

3. Minimal execution time increase (20% or less).

While ERSA is optimized for probabilistic applications, ERSA may also be used for executing general-purpose applications that are less resilient to errors through the concept of configurable reliability. However, the associated costs can be higher.

Major contributions of this paper are:

1. Introduction of the concept of ERSA for probabilistic applications without requiring expensive error detection.

2. Detailed description of ERSA hardware and software architectures and probabilistic algorithm-aware optimizations.

3. Experimental results from ERSA hardware prototypes.

4. Analysis of computation accuracy and execution time trade-offs of ERSA over a wide range of error rates.

2. ERROR RESILIENCE OF PROBABILISTIC APPLICATIONS

We use the following three probabilistic applications (from RMS benchmarks [Kestor 09] and related previous work [May 08]) to demonstrate the effectiveness of ERSA:

1. K-means clustering**:** A classification algorithm that partitions input data (points in *n*-dimensional space) into *K* clusters.

2. Low-Density Parity-Check (LDPC) decoding: A decoder module for the LDPC code, an error-correcting code that is widely

978-3-9810801-6-2/DATE10 © 2010 EDAA

used in communication applications. The decoding algorithm is based on loopy belief propagation.

3. Bayesian network inference: Bayesian network provides ways of "extracting and learning" new information from raw data. Given an image, our specific benchmark identifies cars in that image using its context (e.g., trees, roads, houses in the image).

In this section, we present experimental data on the resilience characteristics of the above applications in a non-ERSA system. In the first set of experiments, errors are injected only to the data by limiting error injections to floating-point registers. As shown in Fig. 1 (and also observed by other researchers), errors injected at bit positions 53 or below do not impact the output quality by more than 1%, revealing the inherent resilience of such applications to low-order bit data errors. However, errors in higher-order bits, beyond bit position 53, very significantly impact application outputs.

In our second set of experiments, we injected errors into general-purpose registers by randomly selecting one of them and flipping one bit at a randomly chosen location. These injections target errors in loop-indices and memory pointers and alter application control flows. The applications were brought down to crash after an average of 1.5 to 1.9 error injections when the error injection rate was as low as 3 errors / sec. These results confirm that, without proper system design, we cannot expect probabilistic applications to produce useful results on unreliable hardware simply because of error resilience in low-order bits.

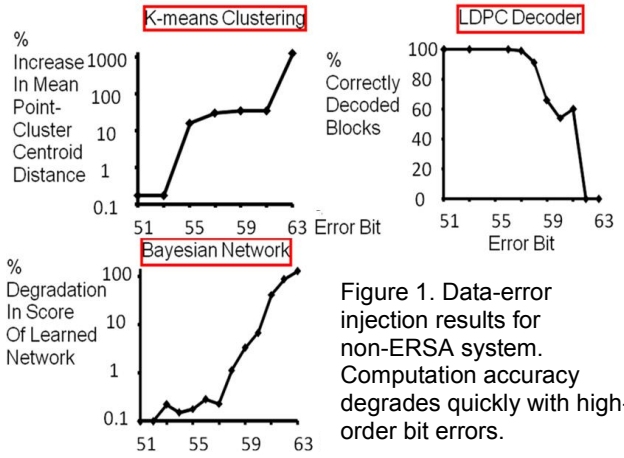

Figure 1. Data-error injection results for non-ERSA system. Computation accuracy degrades quickly with high-order bit errors.

3. ERSA OVERVIEW

ERSA is a multi-core architecture with asymmetric reliability (Fig. 2). ERSA consists of small number of highly reliable cores, referred to as *Super Reliable Cores or SRCs*, together with a large number of cores that are less reliable but account for most of the computation capacity, referred to as *Relaxed Reliability Cores or RRCs* (1 SRC and 8 RRCs in our hardware prototype). The motivation for asymmetric reliability comes from the computation model of probabilistic applications. An entire application can be divided into control-intensive resource management code (main thread in Fig. 3) that needs to be executed on error-free hardware while data-intensive computations are often more error-tolerant. By assigning the control-related code to SRC(s) and the computation intensive code to RRCs, we can minimize the number of processor cores that require high reliability and hence, avoid highly conservative overall system design. In our current ERSA implementation, interconnects between SRCs and RRCs are assumed to be reliable, e.g., using efficient error-correcting codes (ECC) [Raghunathan 03].

Two important points about ERSA are:

1. ERSA does not rely on traditional concurrent error detection in RRCs. Instead, it uses very few "light-weight" checks in RRCs, e.g., timeout and illegal memory access checks as discussed later.

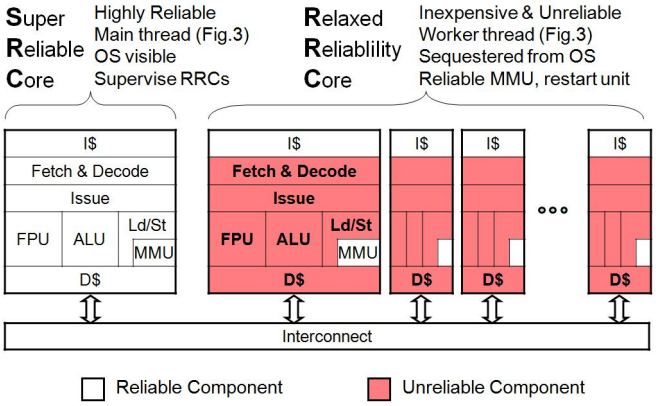

Figure 2. ERSA hardware architecture.

2. ERSA is not restricted to probabilistic applications only. The ERSA platform may be used to execute general-purpose applications as well using the concept of "configurable reliability" [Mitra 08]; i.e., according to application needs, reliability levels of various ERSA hardware components may be adjusted by modifying supply voltage, clock speed or by turning error protection mechanisms on/off.

3.1. Super Reliable Core (SRC)

An SRC is responsible for:

1. Executing the main thread that also performs RRC memory bounds check (Sec. 3.2) setup, RRC computation results reduction and convergence checking (Fig. 3).

2. Executing operations that are not resilient to errors (e.g., the operating system).

3. Distributing tasks to RRCs: ERSA uses task-queues to distribute tasks to RRCs during run-time. Our experience shows that run-time task distribution produces superior results than compile-time ones for ERSA. This is because not all RRC tasks complete execution in the presence of errors – RRCs can be non-responsive while executing tasks. Even when RRC tasks complete execution, they may not pass convergence checks (details in Sec. 3.3) performed by the SRC. All these tasks need to be dynamically reassigned to RRCs. Run-time scheduling with task-queues can adaptively reassign tasks to RRCs. It can also create "diversity" in the mapping of tasks onto RRCs (e.g., if a task repeatedly fails on a particular RRC, the run-time task scheduler can assign that task to another RRC).

4. Supervising RRCs: *Timeout bounds*, similar to watchdog timers [Mahmood 88], are used to detect when an RRC becomes non-responsive. The SRC checks for the liveness of an RRC via the corresponding completion bit in the task-queue. If it is not set within a timeout bound, the SRC terminates the execution of that RRC task and reboots it. In our ERSA experiments, we obtained timeout bound for each application through trial and error. In addition, SRC

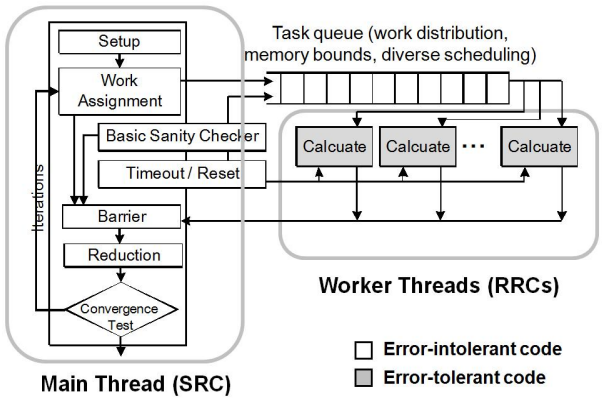

Figure 3. ERSA computation model.

performs basic sanity checks on RRC computation results. Examples include probability value checks (must be between 0 and 1) and loop index checks (RRC loop indices are checked to determine if an RRC terminated early due to errors in loop indices).

3.2. Relaxed-Reliability Cores (RRCs)

RRCs constitute the large majority of on-chip processor cores in ERSA. They provide inexpensive and massive computing power. RRCs are sequestered from the operating system (OS) because the OS is highly vulnerable to errors [Kao 93]. RRC hardware is assumed to be unreliable except the Memory Management Unit (MMU) and the L1 instruction cache. The MMU is where memory access bound violations are detected. Memory bounds-checking is a popular way of detecting invalid memory accesses by producing additional checks for each read or write. ERSA uses memory bounds checks to prevent invalid memory write accesses from RRCs that may be caused by hardware errors. Memory bounds are obtained by collecting all the base addresses and size information of all static, heap and stack objects used in the RRC codes. Compiler-assisted approaches, e.g., [Jones 97], may be used for this purpose as well. In order to reduce the overhead of memory bounds checks, the same memory bounds are used for all memory instructions in an RRC. Incorrect memory accesses inside the memory bounds may create errors that are handled by ERSA similar to other error sources.

For the software code executed on an RRC, *function in-lining* is used to protect RRCs from stack pointer-related errors. Errors in stack pointer can overwrite the return address of a function on the stack so that, when the function returns, program control can jump to arbitrary locations. Most of the function calls in RRC codes are in-lined to protect the RRCs from the stack pointer errors at the expense of increased code size.

3.3. Algorithmic Convergence Test

The ERSA system described so far, i.e., incorporating asymmetric reliability and memory / timeout bounds checks and function in-lining is referred to as *Basic ERSA*. It is far more error-resilient compared to no-ERSA (details in Sec. 4). However, for error rates greater than 5,000 errors/sec/RRC or $5x10^{-5}$ error/cycle/RRC, *Basic ERSA* can result in large computation inaccuracies and significant execution time overheads. In order to overcome these challenges, algorithm-aware software techniques are needed. At high error rates, iterations in probabilistic applications often fail to converge, which leads to high execution time overheads. We designed a set of application algorithm-aware software techniques, which when used in conjunction with *Basic ERSA*, results in a system we refer to as *Enhanced ERSA*. Sections 3.3.1 and 3.3.2 present the relevant details.

3.3.1. Convergence Damping

Hardware errors in RRCs can result in fluctuating behaviors in computation results. Damping these fluctuations to make the computation results more stable is an effective way of mitigating the impact of hardware errors.

The damping scheme used in *Enhanced ERSA* is to "partially update" the output of an RRC; i.e., if the change (from the corresponding output in the previous iteration) exceeds a given amount, referred to as *saturation limit*, then simply update that output with the saturation limit. If the change is within the saturation limit, we simply accept the output of that RRC. As shown in Table 1, in K-Means clustering, if a cluster diameter changes by more than +/-30% of the previous value (corresponding to a cluster assigned to an RRC), the coordinates of that cluster center are adjusted to make the diameter change by +/-30% of the previous value only. In LDPC decoding, the probability of a bit being '1' is allowed to be updated by +/-0.25 at maximum until it reaches 0 or 1 in one iteration. Similarly, the probabilities of Bayesian-Network nodes are allowed to change by at most +/-0.3 per iteration. Choosing appropriate saturation limits is very important because over-damping can delay convergence (and increase execution time). For our ERSA experiments, we carefully chose these values through trial and error. One may choose not to use convergence damping until the occurrence of a pre-determined number of iterations in order to accelerate convergence.

3.3.2. Convergence Filtering

Although convergence damping reduces fluctuations**,** it is better to discard the results from an iteration that shows excessively large fluctuations. *Convergence filtering* "selectively updates" the computation output by deciding whether to accept or to discard the results from all RRCs produced during the iteration. The difference between convergence filtering and damping is that filtering applies to the results produced by all RRCs during an iteration (i.e., all results accepted or discarded) whereas convergence damping is applied individually on the results obtained from each RRC. The metrics used by the SRC for convergence filtering are listed in Table 1. For example, LDPC decoding uses the total number of bits that failed parity check at the end of an iteration (referred to as *failed bits*) as the convergence criterion and declares convergence when the number of failed bits becomes zero. If this number increases by any amount, convergence filtering discards the results from that entire iteration. In K-Means clustering, the total sum of cluster diameters, which monotonically decreases to minimum value, is used as the metric for convergence filtering. If the sum increases by greater than 10% of the previous sum, the iteration is discarded. In Bayesian-network, the Euclidean distance between the probabilities of present and previous iterations is measured. If the distance is greater than 10% of the sum of probabilities in all nodes, new results are rejected. Similar to convergence damping, the parameters used for convergence filtering are obtained through trial and error. Convergence filtering can also be skipped until the occurrence of a pre-determined number of iterations.

Table 1. Parameters for algorithmic convergence checking of ERSA. Numbers in bold are example parameters for our experiments at error rate of 30,000 error/sec/RRC and are subject to change for different error rates.

Bench-marks	Convergence damping	Convergence filtering	Accuracy metric (in Sec. 4.1.1)
K-Means	Change in a cluster diameter **(+/- 30% of previous value)**	Increase in the sum of cluster diameters **(Iteration discarded if the increase is greater than 10% of previous sum)**	Increase in cluster diameter (%)
LDPC decode	Change in the probability of a bit being '1' **(probability value +/-0.25)**	Parity bit check **(Iteration discarded if the number of bits that failed parity check increases)**	Correctly decoded blocks (%)
Bayesian Network	Change in the probability of Bayesian-net nodes **(probability value +/-0.3)**	Euclidean distance between probabilities **(Iteration discarded if the distance is greater than 10% of the sum of probabilities in all nodes)**	Correct classification (%)

4. ERSA EXPERIMENTS

4.1. ERSA Experimental Results: Logic Errors in RRCs

We present experimental results from an ERSA prototype using PowerPC 405 100MHz cores in a Xilinx Virtex II Pro FPGA system (Fig. 4(a)). In our prototype system, we have one SRC and eight RRCs, which are mapped to two PowerPC hard cores in the FPGA. One PowerPC core is dedicated to the SRC and the other core is time-multiplexed to emulate RRCs. Time slices to each RRC process are assigned in equal portions and in circular order. Separate stack memories are allocated for RRCs. For each emulated RRC, memory bounds checking is implemented using translation lookaside buffer (TLB) exception handler. At every TLB miss, memory access address is checked using the memory bounds (Sec. 3.2) assigned to the corresponding RRC. RRC timeout bound checking utilizes the hardware watchdog timer in PowerPC core, which times out and induces RRC core restart whenever its processor becomes non-responsive.

In this section, we demonstrate the effectiveness of ERSA through hardware error injections in RRCs. In the PowerPC 405 architecture, there are 32 general purpose registers (GPR) and no floating point registers. GPRs also serve as stack and base pointers. Thus, injecting errors into GPRs will emulate the effects of both control and data errors. RRC error injection flow is shown in Fig. 4(b). The error injection rate was varied from zero to 30,000 errors/sec/RRC, which corresponds to $3x10^{-4}$error/cycle/RRC (processor clock frequency of 100MHz). In reality, hardware errors can be generated by any component such as adders, functional units, latches and flip-flops, etc. Injecting errors into those components directly will improve the accuracy of our experiments. This was not possible in PowerPC hard cores. Reconfigurable computing platforms may be used for more detailed experimentation.

The error injection routine is invoked by a programmable interrupt timer. One register from the 32 GPRs is randomly chosen and one bit out of 32 bit locations is randomly chosen and flipped. In PowerPC 405 architecture, there are other 32 special purpose registers that are used for controlling processor resources such as debugger and timers, etc. Since they are not visible to RRC code, errors were not injected into those registers. Similar experimental setup was used in [Kao 93, Li 06, Wong 06] except that, unlike the ERSA hardware prototype, errors were injected using software simulators.

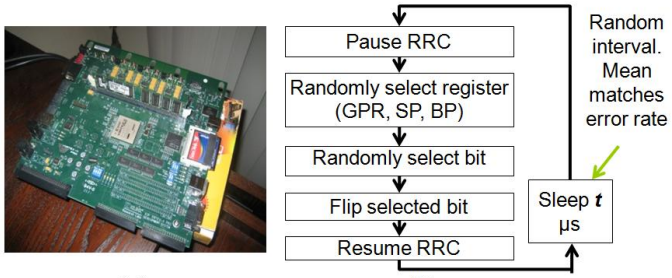

Figure 4. ERSA hardware prototype. (a) Xilinx Virtex II Pro board with Dual PowerPC 405. (b) Error injection model.

4.1.1. ERSA Computation Accuracy

We compare three implementations: *No ERSA, Basic ERSA* and *Enhanced ERSA* (Sec. 3.3). For the *No ERSA* case, probabilistic applications are executed under error injections without using the ERSA framework. *Basic ERSA* implements ERSA with memory/timeout bounds checks, task-queue, sanity checks and function in-lining (details in Sec.3). The *Enhanced ERSA* implementation includes *Basic ERSA* together with convergence damping / filtering. Figures 5(a)-(c) show the computation accuracy of ERSA. Individual metrics used to measure the computation

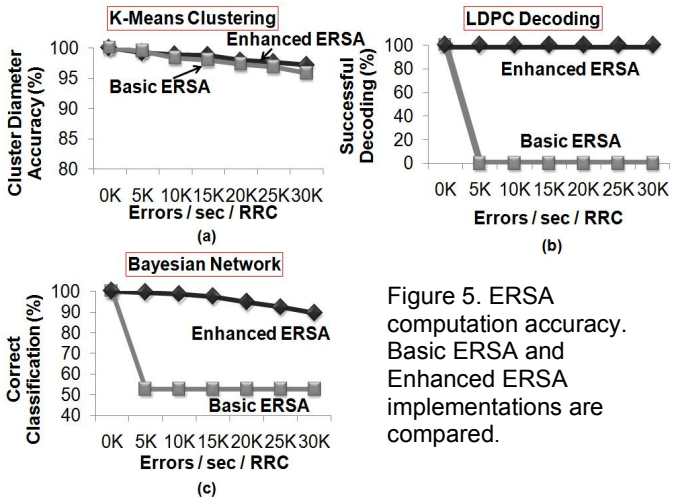

Figure 5. ERSA computation accuracy. Basic ERSA and Enhanced ERSA implementations are compared.

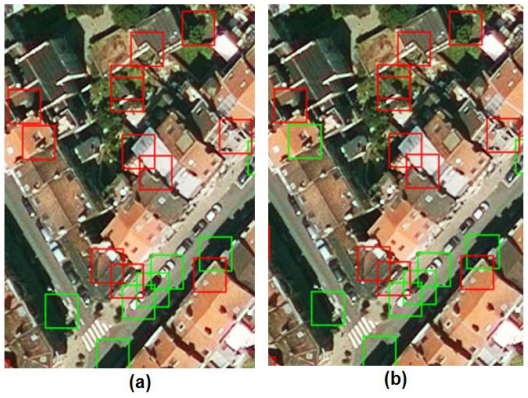

Figure 6. Output images of Bayesian Network Inference with (a) 100% (b) 90% accuracy. Objects inferred as cars are marked in green squares and others in red.

accuracy are detailed in Table 1. Without ERSA (i.e., for the *No ERSA* case), the system crashes after 1 to 3 errors in average and no useful output is generated. *Basic ERSA* does not cause any system crash and achieves convergence. However, computation accuracy degrades significantly (by 50 to 100%) as error injection rate increases. K-Means clustering shows less degradation of computation accuracy, which can be explained by its high error resiliency due to averaging operations.

With *Enhanced ERSA*, computation accuracy achieved is better than 90% even for extremely high error rates of 30,000 errors/sec/RRC. In Fig. 6, we further analyze the cognitive resilience aspect of the Bayesian network application which exhibits 90% computation accuracy at 30,000 errors/sec/RRC (Fig. 5(c)). Bayesian network inference is used to identify cars in a satellite image. Objects inferred as cars are in green squares and others are in red. Depending on the final intended use of car identification, e.g., retrieving overall traffic conditions, 10% inaccuracy in Fig. 6 may not be very significant.

4.1.2. ERSA Execution time

Figure 7 shows execution times of ERSA applications. For *No ERSA*, execution times are virtually infinite because applications crash or do not converge even at very low error injection rates. With *Basic ERSA*, applications converge but execution time overheads can be very significant: 5 to 7 times compared to the no-error case. With *Enhanced ERSA*, the execution time overhead is less than 20 to 30% and increases gradually (shown in zoomed-in plots in Figs. 7(b,d,f) at extremely high error rates of 20,000 errors/sec/RRC. Note that, there is execution time overhead in ERSA even with zero errors. This overhead comes from convergence and sanity checks on RRC results.

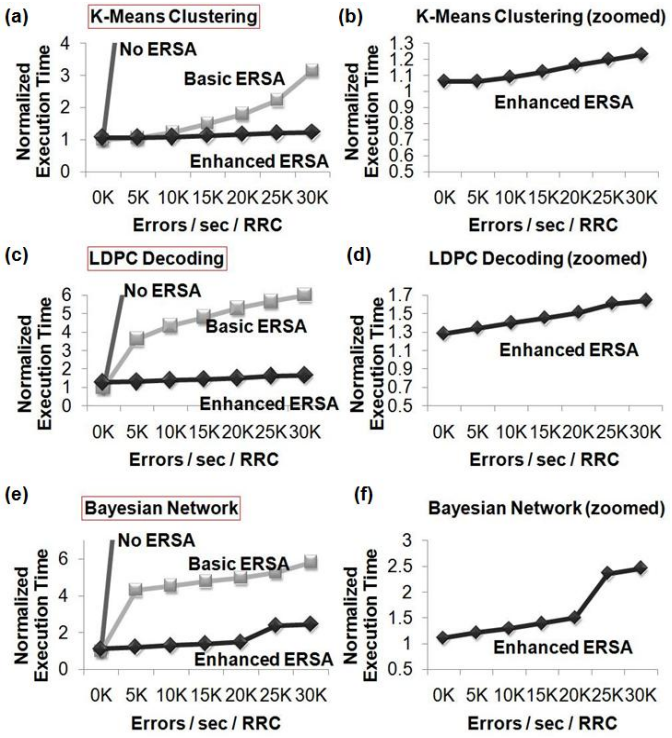

Figure 7. ERSA execution times (a, c, e) and the corresponding zoomed-in plots (b, d, f).

In the current ERSA implementation, RRCs may stall until the SRC completes the checking, which can be improved in the future.

The execution time overhead of ERSA can be traded off with computation accuracy if necessary, i.e., the convergence criteria may be relaxed to make it easier to meet. Applications without errors can also expedite convergence using this technique [Chakradhar 09].

4.2. RRC L1 Data Cache Errors

ERSA techniques are also effective for non-transient static errors in SRAMs, e.g., in L1 data caches. SRAM errors are becoming significant in future technologies [Mukhopadhyay 05, Agostinelli 05]. Vccmin, the minimum voltage at which SRAMs can reliably operate, is a major challenge. SRAM Vccmin errors have permanent locations because they originate from manufacturing-induced variations [Wilkerson 08]. There is another class of Vccmin-related errors called erratic bit errors [Agostinelli 05] that are temporary in nature. Permanent locations of SRAM Vccmin errors make probabilistic applications less error-resilient because the same errors continue to appear over iterations. ERSA overcomes this challenge by moving data around in the L1 cache (since error locations cannot be moved). From an application's perspective, moving data around has similar overall effects as changing error locations. This is accomplished by adding random offsets to memory addresses (Fig. 8(a)). In our ERSA implementation, each RRC has 16KB, two-way set associative data cache with 32 byte line size. A Linear Feedback Shift Register (LFSR) generates a random offset that is to be used over a given period of time. When data is stored in the cache, the offset is added to the address bits to move data to a new memory location chosen by the offset. This offset is subtracted from the address bits when the data is evicted, which makes using the address offset invisible outside the cache. New address offset is generated when there is an explicit erroneous event in the RRC such as RRC reboot or when RRC results are discarded during convergence filtering.

As shown in Fig. 8, the execution time and computation inaccuracy gradually increase until they reach a "knee" point beyond which execution time and computation inaccuracy increase sharply (at SRAM bit error probabilities of 10^{-4} to 10^{-3}, which correspond to

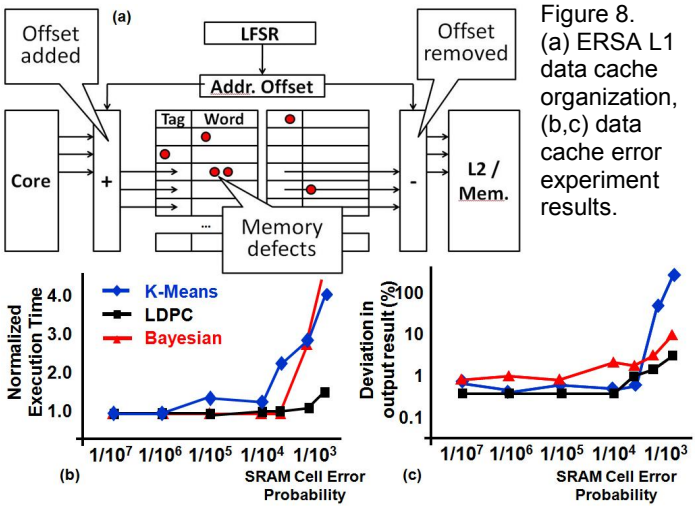

Figure 8. (a) ERSA L1 data cache organization, (b,c) data cache error experiment results.

0.32 to 3.2 % of erroneous cache words). Further research is required to push these knee points to higher error rates.

5. RELATED WORK

The notion of allowing errors in computation results to improve computation time and energy is not new. For example, *imprecise computation* [Liu 91] was used in mission-critical real-time systems to satisfy deadlines. In software engineering, *acceptability-oriented computing* [Rinard 03] aims to reduce software development cost by making sure errors manifest only within acceptable ranges. [Breuer 05] suggested ways of improving chip yield by using imperfect chips in applications where degradation of output quality is acceptable (e.g., multimedia, compression). In algorithmic noise-tolerance (ANT) [Shanbhag 02], the supply voltage of a DSP system is scaled beyond the critical voltage imposed by the critical path delay. Errors were compensated for using algorithmic noise-tolerance in DSP algorithms. Stochastic Sensor Network on a Chip (SSNoC) [Varatkar 07] extends ANT by using distributed computational units (or sensors) whose outputs are combined using robust statistical signal processing techniques. [Eltawil 05] utilized cognitive, wireless application-level error resilience and defect maps to implement fault-tolerant embedded memories for System-On-Chips (SOCs). However, coverage for control errors is not clear. [Wong 06, Li 06] reported error resilience of probabilistic applications mainly for single-event upsets. Finally, there is a large body of literature on application-

Table 2. ERSA vs. related work.

	Error rates	High-order bit errors / control flow errors	Non-transient memory errors	HW prototype
Imprecise computation [Liu 91]	N/A	No	No	N/A
Acceptablilty-oriented computing [Rinard 03]	N/A	No	No	N/A
Error-tolerance [Breuer 05]	Single stuck-at-fault	Depends	No	Yes
Algorithmic noise-tolerance [Shanbhag 02]	10^{-3} BER	Yes	No	Yes
Soft error resilience [Wong 06]	10^{-9} error/cycle	Yes (single error)	No	No
Cooperative redundancy [Eltawil 05]	10^{-3} BER	No	No	Yes
ERSA (this paper)	$2x10^{-4}$ error/cycle /core	Yes (high error rates)	Yes	Yes

dependent error checking techniques [Huang 84, Pattabiraman 06]. Table 2 presents a qualitative summary of related work. ERSA comprehensively addresses major classes of errors: control errors, high-order bit errors (and low-order bit errors), and non-transient memory errors. It utilizes probabilistic and cognitive error resilience, together with application algorithm-aware convergence checking to withstand extremely high error rates, as demonstrated by ERSA hardware prototype results.

6. CONCLUSION

The Error Resilient System Architecture (ERSA), presented in this paper, demonstrates that it is possible to utilize error resilience properties of probabilistic killer applications for designing error-resilient systems on inexpensive hardware with very high error rates without incurring the high costs of traditional redundancy techniques. However, special hardware-software-algorithm design considerations, like the ones used in ERSA, are necessary to achieve this objective – simply running error-resilient applications on hardware with high error rates does not produce useful results. As experimental results demonstrate, the ERSA hardware prototype is resilient to: 1. Very high error rates of the order of 20,000 errors/sec/RRC or $2x10^{-4}$ error/cycle/RRC in architecturally-visible registers with competitive performance; 2. High error rates in data caches, such as those caused by Vccmin. Such high error rates extend far beyond radiation-induced soft error rates and may be caused by highly frequent erratic intermittent errors, process variations, voltage droops, and Vccmin. ERSA maintains 90% or better accuracy of output results, together with execution time overhead of 20~30%. In addition, ERSA is not restricted to probabilistic inference applications and can execute general-purpose applications using the concept of "configurable reliability" at higher cost.

Several future ERSA enhancements and optimizations are possible: 1. Synergies with traditional error detection (e.g., parity and residue checking, timing error detection), application-specific error detection, and algorithm-based fault-tolerance techniques; 2. Systematic techniques for convergence damping and filtering (rather than trial-and-error based on error injection experiments); 3. Utilizing ERSA for a broader range of applications including speech recognition and multimedia applications and for diverse hardware architectures; 4. Diversified RRC error rates to maximize ERSA throughput at minimized energy consumption.

7. ACKNOWLEDGMENT

This work was supported in part by the Focus Center Research Program (FCRP) Gigascale Systems Research Center (GSRC) and the National Science Foundation.

8. REFERENCES

[Agostinelli 05] M. Agostinelli, *et al.*, "Erratic fluctuations of SRAM cache Vmin at the 90nm process technology node", *Proc. Int'l Electron Devices Meeting,* pp.655-658, 2005.

[Borkar 04] S. Borkar, *et al.*, "Design and reliability challenges in nanometer technologies", *Proc. Design Automation Conference,* pp.75, 2004.

[Breuer 05] M. A. Breuer, "Multi-media Applications and Imprecise Computation", *Proc. Euromicro Conference on Digital System Design,* pp.2-7, 2005.

[Chakrapani 06] L. Chakrapani, *et al*., "Ultra Efficient Embedded SOC Architectures based on Probabilistic CMOS Technology", *Proc. Design Automation and Test in Europe*, pp.1110-1115, 2006.

[Dubey 05] P. Dubey, "Recognition, Mining and Synthesis Moves Computers to the Era of Tera", *Technology at Intel Magazine*, 2005.

[Eltawil 05] A. Eltawil, F. Kurdahi, "Improving Effective Yield Through Error Tolerant System Design", *Proc. Int'l Conference on Electronics, Circuits and Systems*, pp.125-129, 2005.

[Gelsinger 06] P. Gelsinger, "Into the Core…", *Stanford Computer Systems Colloquium,* June 7, 2006.

[Hayes 07] J. Hayes, *et al.*, "An Analysis Framework for Transient-Error Tolerance", *Proc. VLSI Test Symposium,* pp. 249-255, 2007.

[Huang 84] K. Huang and J. Abraham, "Algorithm-based fault tolerance for matrix operations", *IEEE Transactions on Computers,* Vol.C-33, Issue 6, 1984.

[Jones 97] R.W.M. Jones, P. Kelly, "Backwards-compatible bounds checking for arrays and pointers in C programs", *Int'l Workshop on Automated Debugging*, pp. 13-16, 1997.

[Kao 93] W. Kao, *et al.*, "FINE: A Fault Injection and Monitoring Environment for Tracing the UNIX System Behavior under Faults". *IEEE Trans. on Software Engineering,* Vol.19, Issue 11, pp. 1105-1118, 1993.

[Kestor 09] G. Kestor, *et al.*, "RMS-TM: A Transactional Memory Benchmark for Recognition, Mining and Synthesis Applications", *Proc.Workshop on Transactional Computing,* 2009.

[Li 06] X. Li, D. Yeung, "Exploiting Soft Computing for Increased Fault Tolerance", *Proc. Workshop on Architectural Support for Gigascale Integration*, 2006.

[Liu 91] J. Liu, *et al.,* "Algorithms for Scheduling Imprecise Computations", *Computer*, Vol.24, Issue 5, pp.58-69, 1991.

[Mahmood 88] A. Mahmood, E.J. McCluskey, "Concurrent Error Detection Using Watchdog Processors-A Survey", *IEEE Trans. Computers,* Vol. C-37, No. 2, pp.160-174, 1988.

[May 08] M. May, *et al.,* "A Case Study in Reliability-Aware Design: A Resilient LDPC Code Decoder", *Proc. Design Automation and Test in Europe,* 2008.

[Mitra 05] S. Mitra, *et al.*, "Robust System Design with Built-In Soft-Error Resilience", *IEEE Computer,* Vol. 38, no. 2, pp. 43, 2005.

[Mitra 08] S. Mitra, "Globally Optimized Robust Systems to Overcome Scaled CMOS Reliability Challenges", *Proc. Design Automation and Test in Europe,* pp. 941-946, 2008.

[Mukhopadhyay 05] S. Mukhopadhyay, "Modeling of Failure Probability and Statistical Design of SRAM Array for Yield Enhancement in Nanoscaled CMOS", *IEEE Trans. CAD*, Vol. 24, No. 12, pp.1859-1880, 2005.

[Pattabiraman 06] K. Pattabiraman, *et al.*, "Dynamic Derivation of Application-Specific Error Detectors and their Implementation in Hardware", *Proc. European Dependable Computing Conference,* pp.97-108, 2006.

[Rinard 03] M. Rinard, "Acceptability-oriented computing", *Conference on Object Oriented Programming Systems Languages and Applications,* pp.221-239, 2003.

[Shanbhag 02] N. Shanbhag, "Reliable and energy-efficient digital signal processing", *Proc. Design Automation Conference,* pp.830-835, 2002.

[Raghunathan 03] V. Raghunathan, *et al.*, "A survey of techniques for energy efficient on-chip communication", *Proc. Design Automation Conference,* pp.900-905, 2003.

[Van Horn 05] J. Van Horn, "Towards Achieving Relentless Reliability Gains in a Server Marketplace of Teraflops, Laptops, Kilowatts, & "Cost, Cost, Cost"…", *Proc. Int'l Test Conference,* pp. 671-678, 2005.

[Wilkerson 08] C. Wilkerson, *et al.*, "Trading off Cache Capacity for Reliability to Enable Low Voltage Operation," *Proc. Int'l Symp. on Computer Architecture,* pp. 203-214, 2008.

[Varatkar 07] G. Varatkar, *et. al.*, "Sensor Network-On-Chip," in Proc. *Int'l. Symp. on System-on-Chip*, 2007.

[Wong 06] V. Wong, M. Horowitz, "Soft Error Resilience of Probabilistic Inference Applications", *Workshop on Silicon Errors in Logic-System Effects,* 2006.

[Yu 00] S. Yu, *et al.*, "An ACS Robotic Control Algorithm with Fault Tolerant Capabilities", *Proc. Int'l Symp. Field-Programmable Custom Computing Machines*, pp.175, 2000.