Towards Transient Fault Tolerance for Heterogeneous Computing Platforms

Nishant George[1], John Lach[1], Sudhanva Gurumurthi[2]
[1]*Charles L. Brown Department of Electrical and Computer Engineering*
[2]*Department of Computer Science*
University of Virginia
<niche, jlach>@virginia.edu, gurumurthi@cs.virginia.edu

Abstract

The computing demands of applications coupled with the power wall problem in modern processors are expected to pave the way for heterogeneous computing platforms that are composed of a variety of processors and hardware accelerators. While current heterogeneous platform design analyses assess area, performance, and power, the tremendous increase in transient fault rates requires that reliability analyses also be included, especially since fault protection mechanisms can directly affect the aforementioned area, performance, and power analyses – and they affect these metrics differently when implemented on different processing components. Heterogeneous platform design therefore requires accurate characterization of fault protection mechanisms when used in different processing components. This work-in-progress report details the first step in this direction, providing a characterization of various transient fault protection mechanisms in ASICs and FPGAs.

1. Introduction and Motivation

Over the past few years, a large number of computationally rich applications have emerged in diverse fields such as bioinformatics, physical and environmental sciences, business analytics, and data mining. Concomitant with the emergence of these applications, there has also been a massive growth in the amount of digital data [Gan07]. We envision that future platforms will respond to the computing demands of these applications through increased levels of parallelism and through the use of heterogeneous hardware accelerators, such as multi-core and manycore processors [Asa06], GP-GPUs [Owe07], FPGAs, custom ASICs, SIMD/Vector units, and media processors. Using such heterogeneous accelerators can provide orders of magnitude improvements in performance and power efficiency [Cho07] and provide a scalable parallel performance for a large class of applications. Processor vendors have started providing support for the use of hardware accelerators (e.g., Intel QuickAssist™ Technology [Pal07], AMD CPU/GPU Fusion™ architecture [Sto06]). There is also a significant effort to develop the requisite software support for such heterogeneous computing platforms [Lin08, Str08].

When designing any computing platform, one of the most important issues to consider is reliability. Lower supply voltages and increased transistor integration densities are making processors more vulnerable to transient faults induced by externally charged particles, such as high-energy neutrons. Although traditionally considered a problem solely for large memory arrays, transient faults have become a serious concern for even the latches and logic within the processor core. Providing transient fault tolerance in microprocessors has been a topic of significant interest in the computer architecture and circuits communities in recent years [Muk08], but transient faults are also a problem for hardware accelerators, such as FPGAs [Sri04] and ASICs [Wis02]. Therefore, when we migrate parts of an application's computation from the main processor onto various hardware accelerators, we can no longer focus solely on the CPU. Fault tolerance methods for general purpose processors are designed to handle faults while running any supported application. Whereas in FPGAs and ASICs, it is possible to tailor the redundancy to an application's specific needs to achieve fault tolerance with maximum efficiency. *We need to consider reliability issues across the ensemble of processing components and develop platform-level transient fault tolerance solutions.*

In particular, we need to approach platform-level transient fault tolerance, both *horizontally* - across the ensemble of components that constitute the computing platform, and *vertically* - across the architecture and circuit layers. We need to develop techniques and tools for facilitating the mapping of an application's computation across the processors and accelerators to meet performance, power, and reliability requirements. In order to accomplish this, we first need to characterize various fault protection strategies for a range of accelerators with respect to area, performance, power, and reliability. This paper represents a first step in this direction, presenting preliminary results from a characterization study we have conducted on transient fault protection for ASICs and FPGAs. In this paper,

we use a data mining benchmark kernel from the MineBench suite [Pis05] to characterize several fault protection strategies for ASICs and FPGAs with respect to key design metrics.

The outline of the rest of the paper is as follows. The next section describes our workload and experimental setup, and Section 3 discusses the various fault protection strategies that we have implemented. Section 4 presents the quantitative results from our characterization study, and Section 5 concludes this paper and discusses future directions.

2. Workload and Experimental Setup

Our experiments are carried out using the *Distance* benchmark kernel from the MineBench [Pis05] suite. This kernel calculates the Euclidean distance between any two points in an *n*-dimensional space. Prior work has demonstrated that the performance of this kernel can be improved by several orders of magnitude by using an FPGA-based accelerator [Cho07].

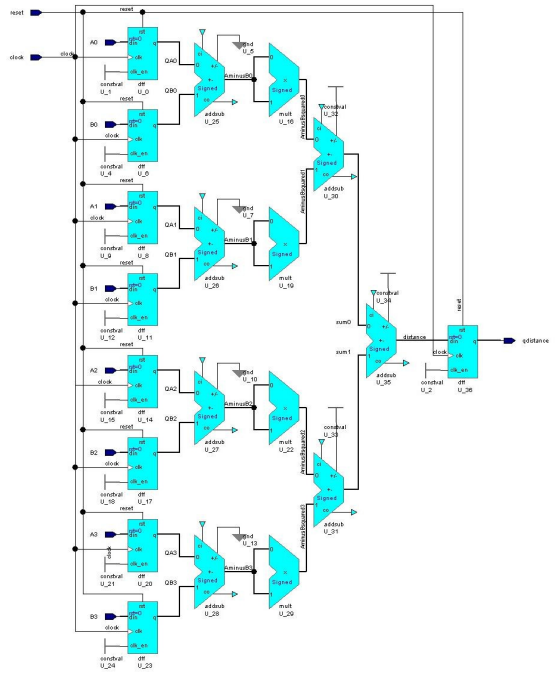


Figure 1. Baseline implementation of the *Distance* kernel that computes the Euclidean distance between two points in an *n*-dimensional space.

All of our designs were implemented in VHDL, and the Mentor Graphics FPGAdvantage and ModuleWare libraries were used for the implementation. After the hardware descriptions were generated using these tools, they were tested using the ModelSim simulator. All designs were then synthesized using Xilinx ISE for the Virtex XCV50 devices to perform area, power, and timing analyses. We used the XPower tool in the Xilinx ISE to compute the static (quiescent) and dynamic power for various hardware designs. The baseline implementation of the *Distance* kernel for n=4 (we also include a scalability analysis) is for execution in a single C-step (an arbitrary time unit that is often equal to one clock period), with registers only at the inputs and output. The datapath is 8 bits wide before the multipliers and 16 bits wide thereafter.

All of the designs analyzed in this paper provide 100% fault-detection coverage with respect to single-point transient faults that do not occur in the checking circuitry of the design (faults in the checking circuitry would result in false positives). Although recent research on transient fault protection in general-purpose processors has suggested that it is feasible to sacrifice small amounts of coverage to gain significant improvements in performance [Par06] and/or power [Ras05], we have not attempted to make such tradeoffs in this initial study and shall explore such techniques in our future work.

3. Transient Fault Tolerance Mechanisms

When designing a custom ASIC or mapping an algorithm to an FPGA, one has significant flexibility in not only functional implementation but also reliability enhancement mechanisms. Designers of ASICs and FPGAs for safety- or mission-critical applications have long used triple modular redundancy (TMR), willingly trading off area and power efficiency for high reliability. But now that reliability issues are affecting commercial applications, the overhead associated with fault-tolerance for ASICs and FPGAs must be reconsidered, including trading off reliability with other metrics of interest, such as area, performance, power, etc. This is especially important in the realm of platform-level design, in which applications are mapped to a heterogeneous architecture, selecting the implementation platform that executes a particular function most efficiently. We propose that fault-tolerance (and the overhead associated with it) needs to be considered in this design space exploration, which calls for an analysis of existing and new fault tolerance mechanisms for ASICs and FPGAs.

For example, the traditional approaches to transient fault tolerance in general-purpose processors involve some form of redundancy. Such redundancy can be incorporated by duplicating the hardware as done in DMR/TMR-based systems (spatial redundancy), by running duplicate copies of the program with a time delay (temporal redundancy), or through the use of coding techniques (informational redundancy). Depending on their implementation, each of these

techniques imposes overheads with respect to area, power consumption, and performance, and can also differ in the amount of transient fault coverage that they provide.

We have examined various spatial and informational redundancy implementations (some new, some well-known) for the baseline *Distance* kernel shown in Figure 1. From this baseline design in which the entire kernel is executed in a single clock cycle, we consider two implementations of spatial redundancy: fine-grained and coarse-grained. In the fine-grained approach, each storage and logic unit is duplicated, and the redundant outputs are compared at each level along the logic sequence of the kernel, thus allowing for both the detection and fine-grained localization of a fault. The implementation of fine-grained spatial redundancy is shown in Figure 2. In the coarse-grained approach, each logic unit is again duplicated but the redundant outputs are checked only at end of the kernel's logic sequence. For each of these approaches, we implemented both designs that use no informational redundancy and designs that use parity to protect each of the storage elements. The results and analysis are presented in Section 4.

Another ASIC/FPGA architecture that has significant flexibility for fault-tolerance and functionality implementation in general is the *synchronous dataflow architecture*, in which a register file feeds a number of arithmetic and/or logical components, whose results are written back to the register file. Components can be reused across C-steps, and the C-steps themselves can be much shorter than in the one C-step implementations. Behavioral synthesis techniques (e.g., traditional force directed list scheduling [Pau89]) take a dataflow graph (DFG) representation of an algorithm, schedule the operations, allocate the components, and bind the operations to the allocated components. Most techniques synthesize a DFG derived directly from the algorithm, but fault-tolerance may call for the duplication of some computation and storage.

For example, a reduced version (n=2) of an already-scheduled DFG for the *Distance* kernel is shown by the white nodes in Figure 3. This schedule requires two adder/subtractor units, two multiply units, and three C-steps (the duration of which are determined by the longest path from the register file through a component back to the register file – as a result, multiplications are often performed across multiple C-steps). Traditional redundancy techniques may just duplicate the entire synchronous dataflow block and compare the outputs, but synthesis techniques can take advantage of unused components to implement redundant computations. The shaded nodes in Figure 3 represent a simple schedule of redundant computation using unused components, such as using the adder/subtractors in t_1 to re-compute the subtractions performed in t_0.

This schedule requires one additional C-step but no additional arithmetic units other than comparators. Existing synthesis algorithms can automate this design space exploration when a redundant computation DFG is provided as input, weighing various metrics as desired.

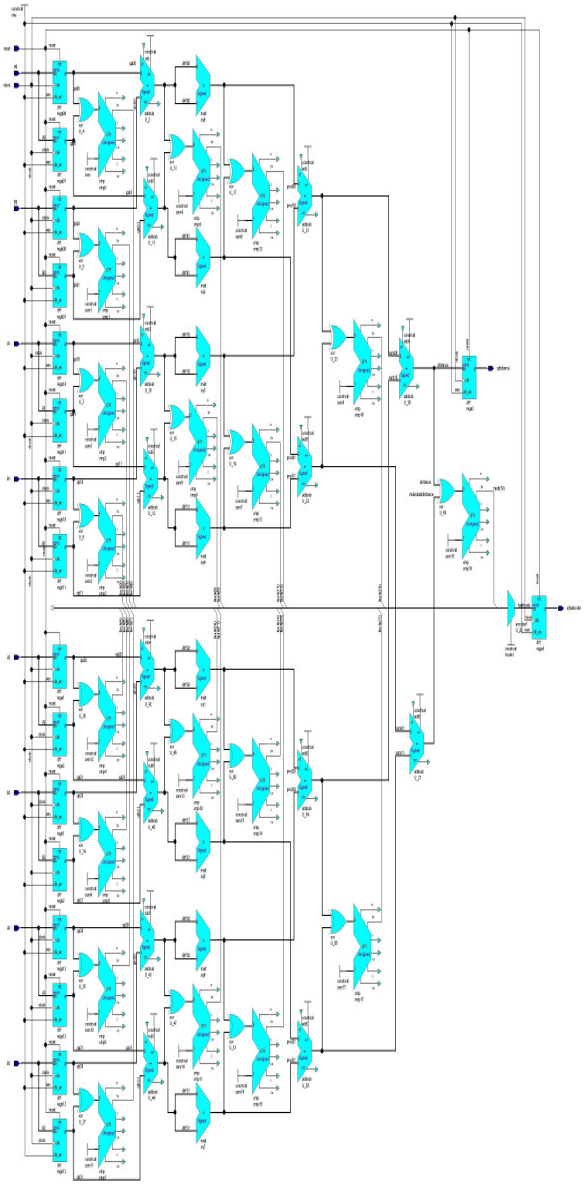


Figure 2. Implementation of the *Distance* kernel with fine-grained DMR spatial redundancy.

It is also interesting to consider when and where the comparisons will be performed. For example, each redundant node could include an implied comparison, but that would likely extend the critical path in the system, thereby extending the duration of each C-step.

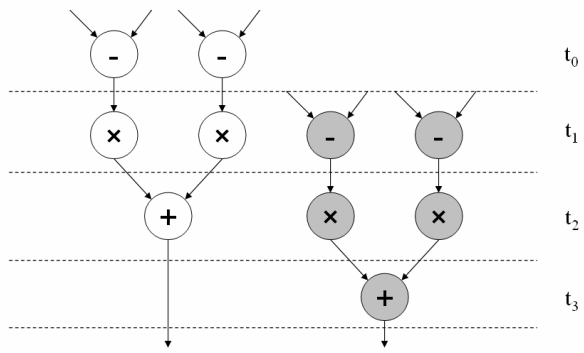


Figure 3. Dataflow graph for time redundant computation of the *Distance* kernel.

Conversely, the comparisons could be performed in the C-step following the redundant computation, but that would likely extend both the number of C-steps required for completion and the lifetime of each piece of data, which typically comes with additional overhead in terms of register requirements. We have implemented both of these synchronous dataflow based redundancy approaches, and the results are analyzed in Section 4.

Behavioral synthesis of synchronous dataflow systems provides one example of the many areas to be explored for platform-level fault tolerance. The design space for heterogeneous platforms is enormous, so different fault-tolerance techniques must be characterized in terms of their coverage and overhead for the various computing platforms. Our immediate future work on behavioral synthesis includes characterizing the area, performance, and power overhead associated with various schedules of redundant operations, including the additional interconnect and registers required for implementation.

4. Results

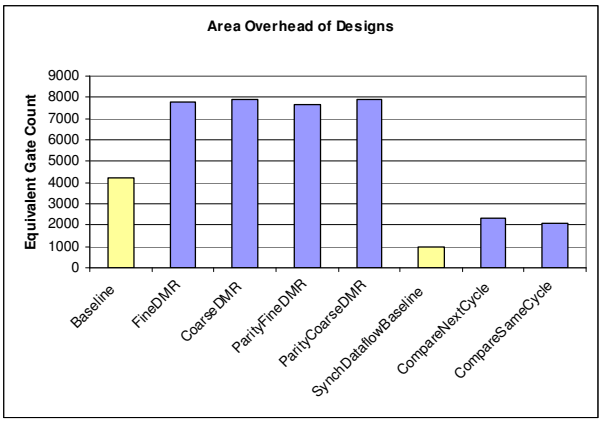
We now characterize the designs described in the previous section with regard to area, performance, power, and energy. One of the most important metrics to consider when incorporating redundancy is the fault detection coverage. As mentioned previously, all of the designs that we present in this study have been implemented to provide 100% fault-detection coverage for the single-point transient fault model. Future work will consider other models.

The set of designs we evaluated along with the number of C-steps for each implementation are given in Table 1. The area, length of minimum clock period, average power, and energy consumption of these designs are shown in Figure 4. The area overhead is expressed in terms of the equivalent gate count required to map a given design onto the Virtex XCV50 device (a metric that is useful for approximation purposes but known to be quite noisy). The minimum clock period is obtained from the synthesis tool based on the longest combinational delay. The power numbers include both static and dynamic power (evaluated at an input activity factor of 50%). The energy results include both static and dynamic power and the total execution time. Each of our designs take a different number of cycles and a different clock period, and the execution latency (i.e. the amount of time necessary to execute one iteration of the *Distance* kernel) is the product of the number of C-steps in Table 1 and the minimum clock period in Figure 4(b).

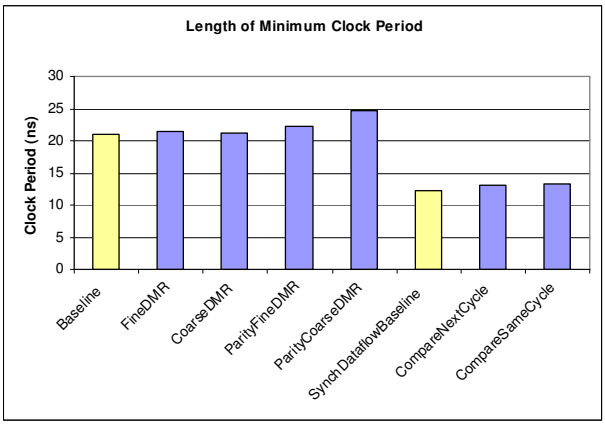
Designs	Design Variants	# C-steps
Single C-step DMR	Baseline	1
	Fine DMR	1
	Coarse DMR	1
	Fine DMR+Parity	1
	Coarse DMR+Parity	1
Synchronous Dataflow	Baseline	$2 + \log_2(n)$
	Compare in next cycle	$2 + \log_2(n) + 1$
	Compare in same cycle	$2 + \log_2(n)$

Table 1. Redundancy design space.

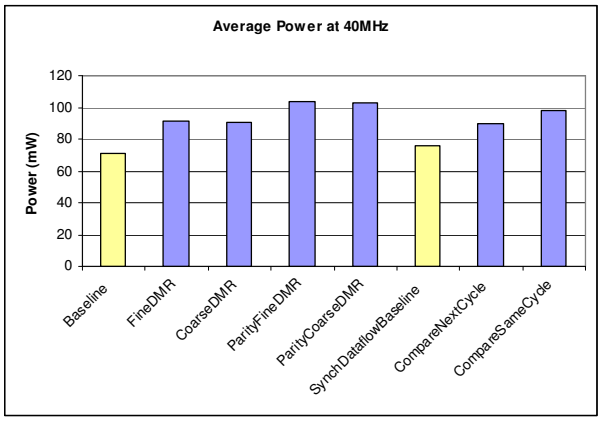
Looking at Figure 4(a), we can see that the DMR designs occupy roughly double the area of the baseline due to the duplication of all the logic units. Interestingly, the additional comparators in the fine-grained designs and the use of parity to protect the storage elements do not impose significant area overheads for the *Distance* kernel. In fact, the overhead is small enough that the equivalent gate count provided by the synthesis tool shows a smaller area for the fine-grained designs than the coarse-grained, an obviously noisy result but reflective of the small area overhead. Registers in the fine-grained and coarse-grained designs are duplicated and have comparators at their output for fault detection in the storage elements. In the design variants with parity, there are no redundant registers but storage is protected with parity. The area requirement associated with one less register is roughly the same as the additional logic required for parity generation and evaluation in terms of equivalent gate count, which is roughly 30 gates per input. However, Figure 4(c) reveals that the use of parity does lead to an increase in the power consumption, although the difference between the fine-grained and coarse-grained is still in the noise (likely due to the fact that the comparator does not change state when a fault is not present).



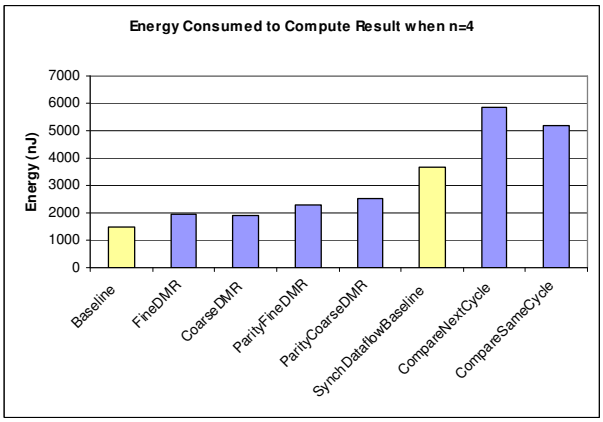
(a) Area overhead



(b) Length of minimum clock period



(c) Average power consumption



(d) Energy consumption

Figure 4. Characteristics of fault-tolerance mechanisms for the *Distance* kernel.

Although synchronous dataflow provides area benefits through component reuse (Figure 4(a)) and has similar average power consumption to the other designs (Figure 4(c)), we can see that the need for additional C-steps (Table 1) increases execution time and leads to higher energy consumption (Figure 4(d)). Such tradeoffs between area and energy would need to be considered when deciding between DMR and redundant synchronous dataflow based approaches to providing transient fault tolerance.

The above analyses are for the *Distance* kernel with n=4, but it is important to explore overhead as n scales. In terms of area, the number of arithmetic units in the DMR implementations will always be double that of the baseline implementation. The magnitude of the comparator overhead in the coarse-grained implementation will remain constant (there is only one comparator regardless of n), while the total number of comparators for the fine-grained implementation will be $4n$ pre-multiplier and n-1 post-multiplier (which are twice the bit width of the pre-multiplier comparators). The area overhead for the synchronous dataflow is more difficult to evaluate given the flexibility of implementation. The above synchronous dataflow analysis assumed that four adder/subtractor units and four multipliers were allocated to implement the kernel in the fewest number of C-steps. If this minimum C-step assumption continues to hold, the number of arithmetic units will scale linearly with n, but the controller and register overhead must still be evaluated. However, if the number of arithmetic units is kept constant, the number of C-steps will scale at a faster rate than shown in Table 1.

An experimental analysis to determine how power scales with n was not performed, but the power consumed in the pre-adder-tree components should scale linearly with n and the post-tree parts with $\log_2(n)$, but the total power of duplicate computation will always be at least twice that of the baseline with no redundancy.

5. Conclusions and Future Work

This paper detailed ongoing work towards characterizing the area, performance, and power overhead associated with a number of transient fault tolerance mechanisms implemented on ASICs and FPGAs, which are candidate accelerator components for the heterogeneous computing platforms of the future. Future work will include establishing a complete taxonomy and characterization of known and emerging reliability enhancement mechanisms for

every abstraction level and across every component in a heterogeneous computing platform in terms of both their coverage of a range of fault models and their power, performance, and area overhead. This will enable a platform-level approach to efficient reliable computing in a future of unreliable components and heterogeneous computing architectures.

Acknowledgements

This work is supported in part by the National Science Foundation under grant Nos. IIS-0612049 and EHS-0410526 and a grant from Intel.

6. References

[Asa06] K. Asanovic et al., "The Landscape of Parallel Computing Research: A View from Berkeley," Technical Report UCB/EECS-2006-183, Electrical Engineering and Computer Sciences, University of California at Berkeley, December 2006.

[Cho07] A. Choudhary, R. Narayanan, B. Ozısıkyılmaz, G. Memik, J. Zambreno, J. Pisharath, "Optimizing Data Mining Workloads using Hardware Accelerators," *Workshop on Computer Architecture Evaluation using Commercial Workloads*, 2007.

[Gan07] F. Gantz et al., "The Expanding Digital Universe," *IDC Whitepaper*, March, 2007.

[Kel07] J.H. Kelm, I. Gelado, M.J. Murphy, N. Navarro, S. Lumetta, W-M. Hwu, "CIGAR: Application Partitioning for a CPU/Coprocessor Architecture," *International Conference on Parallel Architectures and Compilation Techniques*, 2007.

[Lau05] K. Lauritzen, T. Sawicki, T. Stachura, C. Wilson, "Intel I/O Acceleration Technology Improves Network Performance, Reliability, and Efficiently," *Technology@Intel Magazine*, March 2005.

[Lin08] M.D. Linderman, J.D. Collins, H. Wang, T.H. Meng, "Merge: A Programming Model for Heterogeneous Multi-Core Systems," *International Conference on Architecture Support for Programming Languages and Operating Systems,* 2008.

[Muk08]S. Mukherjee, *Architecture Design for Soft Errors*, Morgan-Kaufmann, 2008.

[Owe07]J.D. Owens, D. Luebke, N. Govindaraju, M. Harris, K. Krüger, A.E. Lefohn, T.J. Purcell, "A Survey of General-Purpose Computation on Graphics Hardware," *Computer Graphics Forum*, March 2007.

[Pal07] N. Palaniswamy, "Intel® QuickAssist Technology," *Technology@Intel Magazine*, May 2007.

[Par06] A. Parashar, S. Gurumurthi, A. Sivasubramaniam, "SlicK: Slice-based Locality Exploitation for Efficient Redundant Multithreading," *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2006.

[Pau89] P.G. Paulin, J.P. Knight, "Force-Directed Scheduling for the Behavioral Synthesis of ASIC's," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, June 1989.

[Pis05] J. Pisharath, Y. Lu, W-K. Liao, A. Choudhary, G. Memik, J. Parhi, "NU-MineBench 2.0," CUCIS Technical Report CUCIS-2005-08-01, August 2005.

[Ras05] M. Rashid, E. Tan, M. Huang, D. Albonesi, "Exploiting Coarse-Grained Verification Parallelism for Power-Efficient Fault Tolerance," *International Conference on Parallel Architectures and Compilation Techniques*, 2005.

[Sri04] S. Srinivasan, A. Gayasen, N. Vijaykrishnan, M. Kandemir, Y. Xie, M.J. Irwin, "Improving Soft-Error Tolerance of FPGA Configuration Bits," *International Conference on Computer Aided Design*, 2004.

[Sto06] J. Stokes, "A Closer Look at AMD's CPU/GPU Fusion," *Ars Technica*, November 2006.

[Str08] J.A. Stratton, S.S. Stone, W.W. Hwu, "M-CUDA: An Efficient Implementation of CUDA Kernels on Multi-Cores," Technical Report IMPACT-08-01, University of Illinois at Urbana-Champaign, February 2008.

[Wis02] L. Wissel, S. Pheasant, R. Loughran, C. LeBlanc, B. Klaasen, "Managing Soft Errors in ASICs," *Custom Integrated Circuits Conference*, 2002.