Resiliency-aware Scheduling: Resource Allocation for Hardened Computation on Configurable Devices

Jeremy Abramson & Pedro C. Diniz
USC / Information Sciences Institute

Introduction & Motivation

- Transient Error Rates likely to increase
  - Hostile environment (air- space-borne scenarios)
  - Shrinking features sizes and aging
- Classical Triple-Modular Redundancy
  - Ability to Correct but Very Inflexible
  - Resources Dedicated in Space and Time for a Specific Function

Research Focus and Technical Approach

- This Work: How best to allocate the [limited] resources of an FPGA to increase resiliency to transient Errors?
  - Explore Trade offs between:
  - Resiliency?
  - Performance?
  - FPGA area?
  - Power consumption?
- Technical Approach: Use a source-level, schedule-aware tool to test *many* different "designs" and pick the best one for each situation
  - Exploit Intrinsic Computation Resiliency
  - Supported by Flexible Execution using hybrid TMR units

Intrinsic Resiliency

- Limit to Instruction-Level Parallelism (ILP)
  - Very high in theory, often small in practice
  - 4-10, often lower for many scientific applications
- Excess resources!
  - Itanium2: 6 issue, 6 general purpose ALUs, 6 Multimedia functional units, 4 FP units
  - Virtex5: > 59 multipliers and 59 adders
- Low ILP + idle resources = *Intrinsic Resiliency*
  - The measure of how many operations in a computation can be replicated with no performance penalty for a given resource set
  - Logical inverse of ILP for a specific set of resources (dependencies + contention)
- How do we leverage this? Intrinsic resiliency + resiliency-aware scheduling & resource allocation

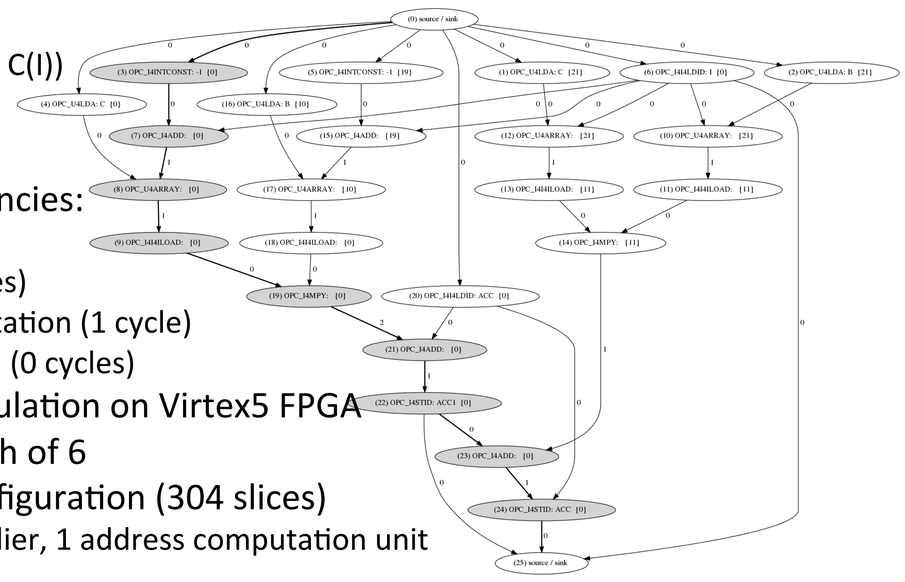
Hybrid TMR and Rollback

- Hybrid TMR: Decouple TMR units to allow for flexibility in scheduling
  - When dependencies allow, act as full TMR units
  - Act as individual units if resource contention
  - Minimal routing overhead
- Allows for tunable performance based on error rate
  - Low error rate: Decouple units, use temporal redundancy for verification / High error rate: Full TMR
- Restart computation from beginning if mismatch detected
  - Optimistic assumption: Simplifies routing, maximizes throughput
  - More complex rollback schemes in the future

Acknowledgements:

Example: Multiply Accumulate

- DO I=1,N
  acc = acc + (B(I)* C(I))
  END DO
- Unrolled 2x
- Operations / Latencies:
  - 4 Add (1 cycle)
  - 2 Multiply (2 cycles)
  - 4 Address computation (1 cycle)
  - Inputs are latched (0 cycles)
- Derived from simulation on Virtex5 FPGA
- Critical Path length of 6
- Minimal area configuration (304 slices)
  - 1 adder, 1 multiplier, 1 address computation unit



Schedule using Classical TMR (left) and hybrid TMR (right) units:

TMR Units

Cycle	TMR Add 1			Array 1	Array 2	TMR Mult 1		
1	7(1)	7(2)	7(3)	10(1)	12(1)			
2	15(1)	15(2)	15(3)	8(1)	10(2)	14(1)	14(2)	14(3)
3				17(1)	10(3)	14(1)	14(2)	14(3)
4				8(2)	8(3)	19(1)	19(2)	19(3)
5				12(2)	12(3)	19(1)	19(2)	19(3)
6	21(1)	21(2)	21(3)	17(2)	17(3)			
7	23(1)	23(2)	23(3)					

Equivalent Hybrid TMR Units

Cycle	Add 1	Add 2	Add 3	Array 1	Array 2	Mult 1	Mult 2	Mult 3
1	7(1)	15(1)	7(2)	10(1)	12(1)			
2	15(2)	7(3)	15(3)	8(1)	10(2)	14(1)		14(2)
3				17(1)	10(3)	14(1)	19(1)	14(2)
4				8(2)	8(3)		19(1)	
5	21(1)	21(2)	21(3)	12(2)	12(3)	19(2)	14(3)	19(3)
6	23(1)	23(2)	23(3)	17(2)	17(3)	19(2)	14(3)	19(3)

Area and Execution Latency Trade-Off for Various Resource Scenarios:

Config. Id #	Hardware Design Configuration: Adder	Array	Mult.	Regs.	4-to-1 Mux.	2-to-1 Mux.	Exec. Cycles	Operation Coverage %	FPGA Design Metrics: Slice LUTs	Slice FFs	Clock (MHz)	Size (LUT) Increase (%): base #0	base #6
0	1	1	1	18	4	2	8	0	254	255	124.3	0	–
1	1	1	1	28	4	2	8	30	305	367	124.3	20	–
2	2	1	1	28	4	4	8	50	328	457	124.3	29	–
3	1	2	1	28	6	2	7	50	352	431	124.2	38	–
4	1	1	2	22	4	0	7	20	397	332	135.7	56	–
5	2	2	2	34	8	2	6	80	566	525	124.2	123	–
6	3	2	3	38	10	2	6	100	700	476	124.2	176	0
7	1 TMR	2	1 TMR	26	6	2	7	100	687	399	116.2	170	-1.8
8	1 TMR	1 TMR	1 TMR	18	4	2	8	100	881	272	94.5	247	20.5
9	2 TMR	2 TMR	2 TMR	18	2	4	6	100	1,179	274	123.7	364	40.6

TABLE I
HARDWARE DESIGN CONFIGURATIONS AND FPGA DESIGN RESULTS FOR MAC 2X COMPUTATION.

Results and Conclusions

- Resiliency-aware Scheduling provides ≈ 20% area decrease and 25% latency improvement over (non Hybrid) TMR schemes
- Allows Tunable Parameterization
  - Pick the configuration that best fits the scenario (high performance, high resiliency, low area requirements, etc.)
- Resiliency aware Scheduling uses Hybrid TMR and intrinsic resiliency to take advantage of inherent properties of a computation to deliver the highest resiliency, lowest latency and smallest area.

References

1. K. Morgan, D. McMurtrey, B. Pratt, and M. Wirthlin, "A comparison of TMR with Alternative Fault-Tolerant Design Techniques for FPGAs," *IEEE Trans. on Nuclear Science,* 54(6), 2007.
2. T.Higuchi, M.Nakao ,and E.Nakano,"Radiation Tolerance of Readout Electronics for Belle II," *Journal of Instr.,* vol. 7, p. C2022, Feb. 2012.
3. L. Anghel and M. Nicolaidis, "Cost reduction and evaluation of a temporary error detecting technique," in *Intl. Conf. on Design, Automation and Test in Europe (DATE),* 2000, pp. 591–598.
4. X. She and P. Samudrala, "Selective Triple Modular Redundancy for Single Event Upset (SEU) mitigation," *2009 NASAESA Conference on Adaptive Hardware and Systems,* pp. 344–350, 2009.
5. S. Cuenca-Asensi, A. Martinez-Alvarez, F. Restrepo-Calle, F. Palomo, H. Guzman-Miranda, and M. Aguirre, "A novel co-design approach for soft errors mitigation in embedded systems," *IEEE Trans. on Nuclear Science,* 58(3), 2011.
6. G. Reis, J. Chang, and D. August, "Automatic instruction-level software- only recovery," *Micro, IEEE,* 27(1), 2007.
7. Y.-M. Hsu, J. Earl, and E. Swartzlander, "Time redundant error correcting adders and multipliers," *Defect Fault Tolerance VLSI Syst.,* 1992.
8. W. Townsend, J. Abraham, and E. Swartzlander, "Quadruple Time Redundancy Adders," *Defect Fault Tolerance VLSI Syst.,* pp. 250–250, 2003.
9. F. Kastensmidt, G. Neuberger, R. Hentschke, L. Carro, and R. Reis, "Designing Fault-Tolerant Techniques for SRAM-based FPGAs," *IEEE Design Test of Computers,* vol. 21, no. 6, pp. 552–562, 2004