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 off 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

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:

<table>
<thead>
<tr>
<th>Config. #</th>
<th>Add</th>
<th>Addr</th>
<th>Array</th>
<th>Multi.</th>
<th>4X Reg.</th>
<th>4X Multi.</th>
<th>4X Add</th>
<th>4X Sys.</th>
<th>Conv.</th>
<th>Occ.</th>
<th>TMR Add 1</th>
<th>Conv.</th>
<th>Occ.</th>
<th>TMR Add 2</th>
<th>Conv.</th>
<th>Occ.</th>
<th>TMR Unit 1</th>
<th>Conv.</th>
<th>Occ.</th>
<th>TMR Unit 2</th>
<th>Conv.</th>
<th>Occ.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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