FluidCheck: A Redundant Threading based Approach for Reliable Execution in Manycore Processors

Rajshekar Kalayappan, Smruti R. Sarangi
Dept of Computer Science and Engineering
Indian Institute of Technology Delhi
New Delhi, India.
Soft Errors

- Temporary nature

- Occurs due to particle strikes on the silicon

- Source of particles:
  - Solar ion flux
  - Explosion of distant stars
  - Impurities in the chip
Soft Errors

- Rare event
  - Particles need to strike at the right place, at the right angle, with the right amount of energy
- Not rare enough to be ignored
  - The critical charge required to flip a bit reduces with reducing feature size and operating voltage
Soft Errors

- Solutions
  - Device level radiation hardening
    - Two to four generations behind commercial counterparts [Courtland2015]
  - System level hardening techniques required
    - Redundancy

DMR

- Compare

TMR

- Vote
Problem Statement

• To *efficiently* execute a set of applications on a chip multi-processor (homogeneous SMT-capable cores), while ensuring *reliability* in the face of soft errors
Related Work: DIVA [Austin1999]

• Meant to provide reliability.

  • IP
  • Execution Assistance:
    • Branch Prediction Hints
    • Operand Value Hints
  • Result
  • Example
    <0x1234><op1=5><op2=2><res=7>
  • Cache line forwarding
Related Work

**SRT [Reinhardt2000], AR-SMT [Rotenberg1999]**
- Saves area
- Better throughput per core

**CRT [Mukherjee2002]**
- Improvement over SRT
- Circumvents hazards borne out of resource requirement similarity between a leader-checker pair
- Better throughput per core
Motivational Example

Without any checking, throughput = 4.84 instructions per cycle
Motivational Example

Without any checking, throughput = 4.84 instructions per cycle

SRT

- Throughput = 3.24
- Similarity in resource requirement
- High throughput threads together
Motivational Example

Without any checking, throughput = 4.84 instructions per cycle

SRT

• Throughput = 3.24
• Similarity in resource requirement
• High throughput threads together

CRT
Motivational Example

Without any checking, throughput = 4.84 instructions per cycle

• Throughput = 3.24
• Similarity in resource requirement
• High throughput threads together

SRT

CRT

• Throughput = 3.55
• Similarity is broken
• Can we do better?
Motivational Example

Without any checking, throughput = 4.84 instructions per cycle

<table>
<thead>
<tr>
<th>SRT</th>
<th>CRT</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="L_perlbench_C_perlbench" /></td>
<td><img src="image" alt="L_perlbench_C_perlbench" /></td>
</tr>
<tr>
<td><img src="image" alt="L_mcf_C_mcf" /></td>
<td><img src="image" alt="L_mcf_C_mcf" /></td>
</tr>
<tr>
<td><img src="image" alt="L_gromacs_C_gromacs" /></td>
<td><img src="image" alt="L_mcf_C_cactusADM" /></td>
</tr>
<tr>
<td><img src="image" alt="L_gromacs_C_cactusADM" /></td>
<td><img src="image" alt="L_gromacs_C_cactusADM" /></td>
</tr>
<tr>
<td><img src="image" alt="L_cactusADM_C_cactusADM" /></td>
<td><img src="image" alt="L_perlbench_C_perlbench" /></td>
</tr>
<tr>
<td><img src="image" alt="L_mcf_C_cactusADM" /></td>
<td><img src="image" alt="L_gromacs_C_perlbench" /></td>
</tr>
</tbody>
</table>

• Throughput = 3.24  
• Similarity in resource requirement  
• High throughput threads together

• Throughput = 3.55  
• Similarity is broken  
• Can we do better?

• Throughput = 3.76
**Motivational Example**

Without any checking, throughput = 4.84 instructions per cycle

<table>
<thead>
<tr>
<th>SRT</th>
<th>CRT</th>
<th>FluidCheck</th>
</tr>
</thead>
<tbody>
<tr>
<td>(L_{perlbench}) (C_{perlbench})</td>
<td>(L_{perlbench}) (C_{mcf})</td>
<td>(L_{perlbench}) (C_{mcf}) (C_{gromacs}) (L_{mcf}) (C_{cactusADM})</td>
</tr>
<tr>
<td>(L_{gromacs}) (C_{gromacs})</td>
<td>(L_{cactusADM}) (C_{cactusADM})</td>
<td>(L_{gromacs}) (C_{cactusADM})</td>
</tr>
</tbody>
</table>

- **SRT**
  - Throughput = 3.24
  - Similarity in resource requirement
  - High throughput threads together

- **CRT**
  - Throughput = 3.55
  - Similarity is broken
  - Can we do better?

- **FluidCheck**
  - Throughput = 3.76
  - Schedules based on the applications’ behavior
  - FluidCheck is a superset of schedules; SRT, CRT are instances within FluidCheck
Simplified Illustration of FluidCheck’s Working

Core A -> Arbiter -> Core B

Core C -> Arbiter -> Core D

L1 -> Arbiter

L2 -> Arbiter

L3 -> Arbiter

L4 -> Arbiter
Simplified Illustration of FluidCheck’s Working

Core A
L1

Core B
L2

Core C
L3
C2
C1

Core D
L4
C1
C3
C4

Arbiter

HELP

C1 unable to keep up
Simplified Illustration of FluidCheck’s Working
Simplified Illustration of FluidCheck’s Working

Arbiter

Core A

Core B

Core C

Core D

L1

L2

L3

L4

C1

C2

C3

C4
Simplified Illustration of FluidCheck’s Working

Periodic reassignment
Simplified Illustration of FluidCheck’s Working
Challenges to achieving FluidCheck

- Reactive phase-based scheduler
- Efficient transfer of hints
- Efficient forwarding of cache lines from the leader to the checker
- Circumventing subtle livelock scenarios
Hardware Architecture

LEGEND
(1) Arbiter
(2) Hint Buffers
(3) Construct hint and forward
(4) iRFB (5) iLFB
(6) dRFB (7) dLFB
(8) Victim cache
(9) Retirement RF
(10) Reliability specific circuitry: (i) request checker assignment
     (ii) send context to new checker
Overview of Redundant Execution

- `<START>` `<CONTEXT>`
  to set up context of checker thread

- `<CONTEXT-END>`
  to begin checker thread execution

- `<HELP>`
  if checker cannot keep up

- `<STOP>`
  to indicate no more hint packets

- `<CHECKED>`
  when all hint packets verified

- `<ERROR>`
  if error detected

- `<CONTEXT>` of checkpoint follows "error" message
Memory Checkpointing

Leader

Pipeline

Ct

L1

Checker

Pipeline

Ct

L1

L2
Memory Checkpointing

Leader

Pipeline

Ct

11010101 1

L1

Checker

Pipeline

Ct

L1

Hint

Store

L2
Memory Checkpointing

Leader

Pipeline

Ct

L1

11010101

1

Checker

Pipeline

Ct

L1

L2
Memory Checkpointing

Leader

Pipeline

Ct

Ld/St

11010101 1

L1

Checker

Pipeline

Ct

L1

L2
Memory Checkpointing

Leader

Pipeline

Ct

Miss!

Ld/St

11010101 1

L1

Checker

Pipeline

Ct

L1

L2
Memory Checkpointing

Leader

Pipeline

Ct

Ld/St

Miss!

11010101 1

L1

Checker

Pipeline

Ct

L1

L2
Memory Checkpointing

Leader

Pipeline

Ct

Ld/St

Evict!

11010101 1

L1

Checker

Pipeline

Ct

L1

L2
Memory Checkpointing

Leader

Pipeline

Ct

00001111

0

Evict!

Ld/St

1101.. 1

L1 Victim Cache

Checker

Pipeline

Ct

L1

L2
Memory Checkpointing

**Leader**
- Pipeline
- Victim Cache
- L1

**Checker**
- Pipeline
- L1

**L2**
Memory Checkpointing

**Leader**
- Pipeline
- Ct
- L1
- Victim Cache

**Checker**
- Pipeline
- Ct
- L1
- 11010101
- Store

**L2**
Memory Checkpointing

Leader

Pipeline

Ct

11001101 1
11010111 1
11110101 1
L1 Victim Cache

SYNC

Checker

Pipeline

Ct

1101.. 1
1001.. 1
L1

L2
Memory Checkpointing

**Leader**

- Pipeline
- Victim Cache
- L1

**Checker**

- Pipeline
- L1

**SYNC**

- 11001101
- 11010111
- 11110101

- 1101..
- 1001..
Memory Checkpointing

Leader

- Pipeline
- Ct
- L1 Victim Cache

Checker

- Pipeline
- Ct
- L1

SYNC

L2
Memory Checkpointing

Leader

Pipeline

Ct

Victim Cache

L1

Rollback

Checker

Pipeline

Ct

L1

L2
Memory Checkpointing

Leader

Pipeline

Ct

L1

Victim Cache

Rollback

Checker

Pipeline

Ct

L1

L2
Forwarding Filters

Leader

Pipeline

L1

Ct

L2
Forwarding Filters

Leader

Pipeline

L1

Ld/St

Ct

L2
Forwarding Filters

Leader

Pipeline

Ct

Hit!

Ld/St

L1

L2
Forwarding Filters

Leader

Pipeline

Ct

Hit!

Ld/St

L1

Do Not Forward

L2
Forwarding Filters

Leader

Pipeline

Ct

Miss!

L1

L2
Forwarding Filters

Leader

Pipeline

Miss!

L1

RFB

Ct

L2
Forwarding Filters

Leader

Pipeline

Ct

Miss!

Hit!

L1

RFB

L2
Forwarding Filters

Leader

Pipeline

L1 → RFB

Miss! → Hit!

Ct

Do Not Forward

L2
Forwarding Filters

Leader

Pipeline

Miss!

L1

Miss!

RFB

Do Not Forward

L2
Forwarding Filters

Leader

Pipeline

L1

Miss!

RFB

Miss!

LFB

11010011

Ct

Do Not Forward

L2
Forwarding Filters

Leader

Pipeline

Miss!

Miss!

Do Not Forward

11010011

L1

RFB

LFB

Ct

L2
Forwarding Filters

Leader

Pipeline

L1

Miss!

Miss!

RFB

11010011

LFB

Ct

L2
Forwarding Filters

Leader

Pipeline

L1

Miss!

Miss!

RFB

LFB

11010011

Ct

Forward

L2
Arbiter Logic: I

- Activity
  - IPC
  - WIPC(x)
- Mapping a Single Thread
  - Select the core with minimum *activity* that has free SMT slots
  - If activity is IPC, scheme is termed *minIPC*
  - If activity is WIPC(x), scheme is termed *minWIPC_x*
Arbiter Logic: II

- Mapping a Set of Threads
  - Scheduling Policies:
    - Pinned Leaders (SP-PL)
    - Unpinned Leaders (SP-UL)
    - Unpinned Leaders All Leaders First (SP-UALF)
- SMT Fetch Policy
  - Full Simultaneous Issue [Tullsen1995]
  - If $n$ threads on a core have activities $A_1, A_2, \ldots, A_n$, then the $i^{th}$ thread gets $\sum_{k=1}^{n} A_k \times B$ fetch cycles (cycle block of size $B$ considered)
Evaluation: Simulation Parameters

• 16-core processor, 4-way SMT
• Core configuration based on Intel Sandybridge and IBM Power7

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipeline width</td>
<td>4</td>
</tr>
<tr>
<td>i-cache and d-cache</td>
<td>32 kB</td>
</tr>
<tr>
<td>Shared L2 cache</td>
<td>12 MB</td>
</tr>
<tr>
<td>NOC topology</td>
<td>2D torus</td>
</tr>
<tr>
<td>Hint buffer</td>
<td>512 entry</td>
</tr>
<tr>
<td>Victim Cache</td>
<td>32 entry</td>
</tr>
<tr>
<td>RFB and LFB</td>
<td>64 entries each</td>
</tr>
</tbody>
</table>
Evaluation Methodology

• Tools
  ▫ Tejas Architectural Simulator
  ▫ McPAT and Orion2 models
• Workloads
  ▫ “low”: 16 applications (16 + 16 threads)
  ▫ “medium”: 24 applications (24 + 24 threads)
  ▫ “high”: 32 applications (32 + 32 threads)
  ▫ In each case 100 random combinations of SPEC CPU2006 benchmarks were considered
• Comparison Metric
  \[
  -1 + \sqrt{\prod_{b \in W} \frac{\text{cycles taken to reliably execute } b}{\text{cycles taken to unreliably execute } b}}
  \]
Evaluation: Results

Medium Load

<table>
<thead>
<tr>
<th></th>
<th>SRT</th>
<th>minIPC</th>
<th>minWIPC_0.65</th>
<th>minWIPC_x</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP-PL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SP-UL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SP-UALF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

mean slowdown (%)

- 47%
- 37%
- 27%
FluidCheck’s Mapping Ability
Performance of Forwarding Filters
Comparison with Generic Scheduling Schemes

- DCCS [Settle2004]
- TCA [Acosta2009]
- IPCS [Parekh 2000]
- L1 BW-aware [Feliu2013]
- RIRS [ElMoursy2006]
Conclusions

- Efficient system-level solutions to handle soft errors are critically sought
- The protection of modern multi-core, multithreading capable processors presents interesting challenges
- Our solution FluidCheck achieves reliability with a mere 27% reduction in performance on average, while seminal works such as SRT (47%) and CRT (37%) present much higher slowdowns
Extra slides
DIVA : Checker Operation

Fetch
- Check IP
- Fetch From IP
- \(<0x1234>\)

Decode
- \(<R_1=R_2+R_3>\)

Execute
- Using the operand value hints
- \(<5+2>\)

Writeback
- Check communication
  - \(R_2 == 5?\)
  - \(R_3 == 2?\)
- Check computation
  - \(7 == \text{res}?\)
  - Write 7 to \(R_1\)

Commit
- Complete store
DIVA : Execution Assistance

- The DIVA checker
  - Faces no data hazards
    - Operand value hints are passed from leader
  - Faces no control hazards
    - The stream of packets from the leader are in correct dynamic order (if no soft error struck the prediction or branching logic)
    - If a soft error occurred (rare event), it is detected when the branch condition is evaluated at the checker
DIVA : Consequence of Execution Assistance

- What gains can be achieved through execution assistance?
  - Checker can be made simpler
  - Checker can be made slower
  - Checker can be made to do more work
Resolving Livelock Issues

- Suppose a checker thread faces a decode stall since the ROB was full
- Suppose some other leader thread on the same core is occupying the head of the ROB and is facing a long latency miss
- The checker thread is forced to migrate
- Possibility of multiple forced migrations in quick succession – detrimental to performance
- Solution – Reservation. If a resource is greater than 95% full, it will not accept any more leader entries