OPERATING SYSTEM SUPPORT FOR REDUNDANT MULTITHREADING

Björn Döbel (TU Dresden)

Brussels, 02.02.2013
Hardware Faults

- Radiation-induced soft errors
  - Mainly an issue in avionics+space\(^1\)
- DRAM errors in large data centers
  - Google Study: > 2% failing DRAM DIMMs per year\(^2\)
  - ECC is not going to even detect a significant amount\(^3\)
  - Disk failure rate about 5%\(^4\)
- Furthermore: decreasing transistor sizes, higher rate of transient errors in CPU functional units\(^5\)

---


\(^2\) Schroeder, Pinheiro, Weber: *DRAM Errors in the Wild: A Large-Scale Field Study*, SIGMETRICS 2009

\(^3\) Hwang, Stefanovici, Schroeder: *Cosmic Rays Don’t Strike Twice: Understanding the Nature of DRAM Errors and the Implications for System Design*, ASPLOS 2012


\(^5\) Shivakumar, Kistler, Keckler: *Modeling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic*, DSN 2002
Fault Tolerance: State of the Union

Software errors

Hardware errors

non-COTS

COTS
Fault Tolerance: State of the Union

- Hardware errors
  - RAD-hard CPUs
  - Redundant Multithr.

- Software errors

non-COTS to COTS
Fault Tolerance: State of the Union

- Hardware errors
  - RAD-hard CPUs
  - Redundant Multithr.

- Software errors

- COTS
  - IBM z/OS
  - HP NonStop

- non-COTS
Fault Tolerance: State of the Union

- Software errors
  - Non-COTS
  - COTS

- Hardware errors
  - Non-COTS
  - COTS

Operating System Support for Redundant Multithreading
Fault Tolerance: State of the Union

Software errors

- RAD-hard CPUs
- Redundant Multithr.
- IBM z/OS
- HP NonStop
- Carburizer

Hardware errors

- COTS
- non-COTS

Operating System Support for Redundant Multithreading slide 2 of 19
Fault Tolerance: State of the Union

- COTS
- Radar-hard CPUs
- Redundant Multithr.
- HP NonStop
- IBM z/OS
- SeL4
- Minix3
- Carburizer

Operating System Support for Redundant Multithreading

slide 2 of 19
Transparent Replication as OS Service

Application

L4 Runtime Environment

L4/Fiasco.OC microkernel
Transparent Replication as OS Service

Replicated Application

L4 Runtime Environment

Romain

L4/Fiasco.OC microkernel
Transparent Replication as OS Service

Unreplicated Application

Replicated Application

L4 Runtime Environment

Romain

L4/Fiasco.OC microkernel
Transparent Replication as OS Service

- Replicated Driver
- Unreplicated Application
- Replicated Application

L4 Runtime Environment

Romain

L4/Fiasco.OC microkernel
Transparent Replication as OS Service

- Replicated Driver
- Unreplicated Application
- Replicated Application

L4 Runtime Environment
Romain
L4/Fiasco.OC microkernel

Reliable Computing Base
Process-Level Redundancy

Binary recompilation
- Complex, unprotected compiler
- Architecture-dependent

System calls for replica synchronization

Virtual memory fault isolation
- Restricted to Linux user-level programs

---

6 Shye, Blomsted, Moseley, Reddi, Connors: *PLR: A software approach to transient fault tolerance for multicore architectures*, DSN 2009
Process-Level Redundancy\(^6\)

**Binary recompilation**
- Complex, unprotected compiler
- Architecture-dependent

*Reuse OS mechanisms*

**System calls for replica synchronization**
*Additional synchronization events*

**Virtual memory fault isolation**
- Restricted to Linux user-level programs

*Microkernel-based*

---

\(^6\) Shye, Blomsted, Moseley, Reddi, Connors: *PLR: A software approach to transient fault tolerance for multicore architectures, DSN 2009*
Why A Microkernel?

- Small components
  - Microrebootable\(^7\)
  - Custom-tailor reliability to application needs\(^8\)


\(^8\) Sridharan, Kaeli: *Eliminating microarchitectural dependency from architectural vulnerability*, HPCA 2009
Why A Microkernel?

- Small components
  - Microrebootable\(^7\)
  - Custom-tailor reliability to application needs\(^8\)
- That’s what we do in Dresden (tm).
  - Reuse Fiasco.OC mechanisms instead of adding new code to the RCB

---


\(^8\) Sridharan, Kaeli: *Eliminating microarchitectural dependency from architectural vulnerability*, HPCA 2009
Why A Microkernel?

- Small components
  - Microrebootable\(^7\)
  - Custom-tailor reliability to application needs\(^8\)
- That’s what we do in Dresden (tm).
  - Reuse Fiasco.OC mechanisms instead of adding new code to the RCB
- Lean system call interface
  - Need to add special handling to fewer syscalls

---

\(^8\) Sridharan, Kaeli: *Eliminating microarchitectural dependency from architectural vulnerability*, HPCA 2009
Romain: Structure

Replica  Replica  Replica

Master
Romain: Structure

[Diagram showing a structure with Master at the bottom and three Replica nodes pointing to it]
Romain: Structure

Replica

Replica

Replica

Resource Manager

System Call Proxy

Master
Resource Management: Capabilities

Replica 1

1  2  3  4  5  6
Resource Management: Capabilities

Replica 1

Replica 2
Resource Management: Capabilities
Partitioned Capability Tables

Replica 1

Replica 2

Master

Marked used

Master private
Overhead vs. Unreplicated Execution

![Bar Chart]

- **Double Modular Redundancy**
- **Triple Modular Redundancy**

### Döbel, Härtig, Engel: Operating System Support for Redundant Multithreading, EMSOFT 2012

Operating System Support for Redundant Multithreading
## Romain Lines of Code

<table>
<thead>
<tr>
<th>Component</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base code (main, logging, locking)</td>
<td>325</td>
</tr>
<tr>
<td>Application loader</td>
<td>375</td>
</tr>
<tr>
<td>Replica manager</td>
<td>628</td>
</tr>
<tr>
<td>Redundancy</td>
<td>153</td>
</tr>
<tr>
<td>Memory manager</td>
<td>445</td>
</tr>
<tr>
<td>System call proxy</td>
<td>311</td>
</tr>
<tr>
<td>Shared memory</td>
<td>281</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>2,518</strong></td>
</tr>
<tr>
<td>Fault injector</td>
<td>668</td>
</tr>
<tr>
<td>GDB server stub</td>
<td>1,304</td>
</tr>
</tbody>
</table>
Hardening the RCB

- **We need**: Dedicated mechanisms to protect the RCB (HW or SW)
- **We have**: Full control over software
- Use FT-encoding compiler?
  - Has not been done for kernel code yet
  - Only protects SW components
- RAD-hardened hardware?
  - Too expensive
Hardening the RCB

- **We need**: Dedicated mechanisms to protect the RCB (HW or SW)
- **We have**: Full control over software
- Use FT-encoding compiler?
  - Has not been done for kernel code yet
  - Only protects SW components
- RAD-hardened hardware?
  - Too expensive

**Our proposal**: Split HW into ResCores and NonRes-Cores
Signaling Performance

Exploring master-replica communication

- 12x Intel Core2 2.6 GHz
- Replicas pinned to dedicated physical cores
- Hyperthreading off

---

---
How About Multithreading?
How About Multithreading?
How About Multithreading?
Problem: Nondeterminism
Problem: Nondeterminism
Deterministic Multithreading

- Related work: Debugging multithreaded programs
  - Slightly different requirement: determinism across runs

---

11 Liu, Curtsinger, Berger: DThreads: Efficient Deterministic Multithreading, OSDI 2011
12 Olszewski, Ansel, Amarasinghe: Kendo: Efficient Deterministic Multithreading in Software, ASPLOS 2009
Deterministic Multithreading

- Related work: Debugging multithreaded programs
  - Slightly different requirement: determinism across runs
- **Strong Determinism**: All accesses to shared resources happen in the same order\(^{11}\).
  - Requires heavy involvement with shared memory accesses
  - Replicating SHM is slow

---

\(^{11}\) Liu, Curtsinger, Berger: *DThreads: Efficient Deterministic Multithreading*, OSDI 2011

\(^{12}\) Olszewski, Ansel, Amarasinghe: *Kendo: Efficient Deterministic Multithreading in Software*, ASPLOS 2009
Deterministic Multithreading

- Related work: Debugging multithreaded programs
  - Slightly different requirement: determinism across runs
- **Strong Determinism**: All accesses to shared resources happen in the same order\(^\text{11}\).
  - Requires heavy involvement with shared memory accesses
  - Replicating SHM is slow
- **Weak Determinism**: All lock acquisitions in a program happen in the same order\(^\text{12}\)
  - Intercept calls to `pthread_mutex_{lock,unlock}`

---

\(^{11}\) Liu, Curtsinger, Berger: *DThreads: Efficient Deterministic Multithreading*, OSDI 2011

\(^{12}\) Olszewski, Ansel, Amarasinghe: *Kendo: Efficient Deterministic Multithreading in Software*, ASPLOS 2009
Externally Enforced Determinism

- Patch entries to `pthread_mutex_{lock,unlock}`
- Catch exception for every call
- Enforce ordering in side the master

Microbenchmark: 2 threads, global counter, 1 lock

<table>
<thead>
<tr>
<th>Replication kind</th>
<th>Execution time</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native execution</td>
<td>0.24 s</td>
<td>1.00 x</td>
</tr>
<tr>
<td>Unreplicated <code>RomainMT</code></td>
<td>4.50 s</td>
<td>18.75 x</td>
</tr>
<tr>
<td><code>RomainMT</code>: DMR</td>
<td>12.72 s</td>
<td>53.06 x</td>
</tr>
<tr>
<td><code>RomainMT</code>: TMR</td>
<td>18.02 s</td>
<td>75.00 x</td>
</tr>
</tbody>
</table>
Internal Determinism

- Exception for every lock/unlock hurts a lot!
- Non-contention case: get progress without exception
Internal Determinism

- Exception for every lock/unlock hurts a lot!
- Non-contention case: get progress without exception
- Replication-aware pthreads library
  - Shared memory between replicas
  - Exchange per-lock progress information
  - Only block if lock currently used by different thread in other replica
Internal Determinism

- Exception for every lock/unlock hurts a lot!
- Non-contention case: get progress without exception
- Replication-aware `pthreads` library
  - Shared memory between replicas
  - Exchange per-lock progress information
  - Only block if lock currently used by different thread in other replica

- Problems:
  - Replacing `pthreads` vs. binary-only support → `pthreads` is a shared lib anyway
  - No support for different synchronization mechanisms
  - New `pthreads` code becomes part of the RCB.
Internal Determinism

Microbenchmark: 2 threads, global counter, 1 lock

<table>
<thead>
<tr>
<th>Replication kind</th>
<th>Execution time</th>
<th>Overhead</th>
<th>External det.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native execution</td>
<td>0.24 s</td>
<td>1.00</td>
<td>x</td>
</tr>
<tr>
<td>Unreplicated RomainMT</td>
<td>0.27 s</td>
<td>1.13</td>
<td>18.75 x</td>
</tr>
<tr>
<td>RomainMT: DMR</td>
<td>0.46 s</td>
<td>1.92</td>
<td>53.06 x</td>
</tr>
<tr>
<td>RomainMT: TMR</td>
<td>1.43 s</td>
<td>6.01</td>
<td>75.00 x</td>
</tr>
</tbody>
</table>
Conclusion

- Redundant Multithreading as an OS service
- Support for binary-only applications
- Benefit from microkernel by reuse and design
- Overheads <30%, often <5%
- Multithreading – external vs. internal determinism
- Work in progress (not in this talk):
  - Shared memory handling is slow
  - Bounded detection latency using a watchdog
  - Dynamic adjustment of replication level and resource usage
Nothing to see here

This slide intentionally left blank.

Except for above text.
What about signalling failures?

- Missed CPU exceptions → detected by watchdog
- Spurious CPU exceptions → detected by watchdog / state comparison
- Transmission of corrupt state → detected during state comparison

Overwriting remote state during transmission

- NonResCore memory
- Accessible by ResCores, but not by other NonResCores
- Prevents overwriting other states
- Already available in HW: IBM/Cell
Romain

http://www.dynamo-dresden.de