Explicitly Parallel Programming with Shared-Memory is Insane:
At Least Make it Deterministic!

Joe Devietti, Brandon Lucia, Luis Ceze and Mark Oskin
Parallel Programming is Hard

- Race conditions make life difficult

```c
barrier_wait();
x = 1;
return x;
```  

```c
barrier_wait();
x = 2;
```

Deterministic parallel execution would be nice!
Wouldn't it be nice if...

- ...execution were reproducible on a machine?
  - No more heisenbugs!
  - Run parallel programs forwards and backwards

- ...execution were reproducible across machines?
  - Reduces the parallel testing coverage problem to the single-threaded testing coverage problem
  - Can reproduce bugs found in the field
  - Increases robustness of deployed parallel code
Related Work

- Deterministic, implicitly-parallel languages
  - StreamIt [ASPLOS 2002], Jade [TOPLAS 1998]
  - Typically domain-specific

- Record+replay
  - RecPlay [TOCS 1999], FDR [ISCA 2003], Rerun [ISCA 2008], DeLorean [ISCA 2008]
  - Log ordering of memory operations

- Serialize execution (Simics [Computer 2002])
DMP: A Deterministic Multiprocessor

- Determinism: same input yields same output
  - What is “input”?
  - Input is value and timing of I/O and OS events
- DMP provides deterministic interleaving of memory operations
  - Serialize execution in a consistent, but arbitrary way
Valid Non-deterministic Executions

- lock, scheduling decision, race condition, ...

DMP picks the same valid execution every time

Each root-leaf path is a valid execution

DMP serializes execution in a consistent way
Serialized Execution

store A
store P
store A
store P

store B
load A
store B
load A

store P'
store B
store P'
store B

“Deterministic Token” gets passed after every insn

6 steps

Deterministic Multiprocessors – Joe Devietti – *SHCMP 2008*
Serialized Execution

6 steps

Passing token after every insn is expensive

One of many possible serializations
Serialized Execution: Flow of Data

DMP enforces the same serialization across program runs

A particular serialization enforces a particular flow of data

store A
store P
store B
load A
store P'
store B

store P
store B
store P'
DMP Interface and Implementation

- Interface: deterministically serialized execution
  - Preserves program behavior across runs

- Naïve implementation:
  - Execute insns in round-robin order
  - \(n\times\) slowdown on \(n\) threads :-(

- Better implementation: “OoO superscalar”
  - Serialize only \texttt{when} necessary
  - Serialize only \texttt{for as long as} necessary
Recovering Parallelism

- Parallelize thread-private accesses
  - Sharing Table
- Speculatively parallelize execution
  - Transactional Memory (TM)
- TM + Speculative Value Forwarding
  - TM-Forward
- Smarter Quantum Building
DMP-Sharing Table: Exploiting Thread-private Data

- Thread-private accesses can't affect other threads
  - Okay to execute private accesses in parallel
- Sharing Table: locations are Shared or Private
  - Shared = S state, Private = M/E state
  - Need to hold DT to update sharing table
DMP-TM: Leveraging Speculation

- Execute quanta as implicit transactions
  - Quanta execute speculatively in parallel
    - Abort+retry if serialization was violated
  - **Commit quanta in order** (need DT to commit)
DMP-TM Execution

- store A
- store P
- load A
- store B
- load A
- store B
- store P'
- store B

66 steps
DMP-TM Execution

Ordering + isolation = “memory renaming”
WAW/WAR are “false” conflicts

4 6 steps
DMP-TM-Forward: Speculative Value Forwarding

- Speculatively forward values to “future” quanta
  - Can potentially avoid squashes even with true (RAW) data dependences
  - Must squash yourself if data you were forwarded is overwritten by “past” quantum
  - When you squash, must squash all your consumers
TM-Forward Execution

24/6 steps

Same serial flow of data, but highly parallel execution!
Recovering Parallelism

- **Sharing Table**
  - Parallelizes accesses to thread-private data
  - Non-speculative

- **TM and TM-Forward**
  - Speculation allows for more parallelism
  - “memory renaming” means fewer squashes

- **Smarter Quantum Building**
Quantum Building

- Building quanta by just counting dynamic insns is simple, but can be slow
Naïve Quantum Building: Convoying

lock L
store A
unlock L
...

lock L
lock L

lock L
store A
Smarter Quantum Building

• Enclose critical sections in a single quantum!
  – Start new quantum after an “unlock”
• Other quantum building strategies in paper

lock L
store A
unlock L
...

lock L
store A
unlock L
...

Deterministic Multiprocessors – Joe Devietti – SHCMP 2008
Experimental Methodology

• Simulator using PIN
  – Functionally models effects of serialization
  – Models address conflicts, limited TM buffering
  – Assume 1 IPC, free commits

• SPLASH2 benchmark suite
Runtime Overhead

runtime normalized to non-deterministic parallel execution

benchmark

TM-Forward
TM
SharingTable
Serial
Runtime Overhead

runtime normalized to non-deterministic parallel execution

benchmark

TM-Forward
TM
SharingTable
Serial
Runtime Overhead

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>TM-Forward</th>
<th>TM</th>
<th>SharingTable</th>
<th>Serial</th>
</tr>
</thead>
<tbody>
<tr>
<td>fft</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>lu-c</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>lu-nc</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>ocean-c</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>ocean-nc</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>radix</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>volrend</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>water-ns</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>
Runtime Overhead

Runtime normalized to non-deterministic parallel execution

- TM-Forward
- TM
- SharingTable
- Serial

benchmark

Deterministic Multiprocessors – Joe Devietti – SHCMP 2008
Non-deterministic events

DMP

- Memory
- Interleavings
- Environmental factors

I/O
- File I/O
- Network I/O

OS events
- Thread Scheduling
- IPC

Deterministic Multiprocessors – Joe Devietti – SHCMP 2008
Conclusions

• Determinism is a Good Thing
  – Simplifies debugging, testing and (potentially) deployment of parallel programs
  – We want sequential behavior with parallel performance

• We show several ways to build efficient DMPs
  – No memory log
  – Competitive performance
Questions?