Architecture Optimizations for Synchronization and Communication on Chip Multiprocessors

Sevin Fide and Stephen Jenks

Workshop on Multithreaded Architectures and Applications
International Parallel and Distributed Processing Symposium
April 18, 2008
Outline

- Introduction
- Register-Based Synchronization
- Data Communications via Prepushing
- Simulation Environment
- Simulation Results
- Conclusion
**Introduction**

- **Chip Multiprocessors (CMPs)**
  - Increasingly popular to address the growing demand for higher performance
  - Enable concurrent execution of multiple threads

- No explicit synchronization and communication support for multithreaded applications running on CMPs
Problems

- Synchronization Overhead
  - Spin Waits
- Memory Bandwidth Bottleneck
  - Many Simultaneous Accesses
- Cache Pollution
  - Data Evictions from Shared Cache
- Demand-Based Data Transfers
  - Depend on Coherence Mechanisms
Conventional Parallel Programming

- Data parallelism by splitting data across multiple threads
- Memory interface is overburdened
- Performance degrades due to large number of cache misses
Synchronized Pipelined Parallelism Model

- SPPM: Producer-consumer parallelism targeted for CMPs

- Producers and consumers communicate via caches
  - Producer fetches data into cache and modifies it
  - Consumer uses modified data

- A large number of cache misses are converted into hits
Outline

- Introduction
- Register-Based Synchronization
- Data Communications via Prepushing
- Simulation Environment
- Simulation Results
- Conclusion
for $i \leftarrow 0$ to $N$ do
    while sync window for $\text{dataBlock}[i]$ is violated
        wait
        read $\text{dataBlock}[i]$
        generate results

- No Useful Work During Spin Waits
- Synchronization Window
  - Access synchronization variables in memory
- Coherence Traffic
for i←0 to N do
  while sync window for dataBlock[i] is violated
    wait
  read dataBlock[i]
generate results

- Optimal Miss Rate: 15%
- L1 Cache Latency: 1 cycle
- L2 Cache Latency: 12 cycles
- Average Access Time
  = L1 Latency + Miss Rate * (L1 Latency + L2 Latency)
  = 3 cycles
Register-Based Synchronization

- To avoid spin waits in multithreaded applications
- Register-Based Synchronization (RBS)
  - Employs shared registers with full/empty status bits
- Improvements:
  - Reduced miss rates
  - Reduced coherence traffic
  - Reduced execution time
  - Idle mode can save power
RBS Implementation

- Memory mapped locations to keep track of synchronization variable accesses

- Device driver allocates kernel memory and allows it to be mapped to user space

- Device’s reserved locations accessed as if they were hardware registers
Outline

- Introduction
- Register-Based Synchronization
  - Data Communications via Prepushing
- Simulation Environment
- Simulation Results
- Conclusion
Prepushing

- **Improvements:**
  - Reduced data requests
  - Reduced cache misses
  - Reduced communication latency
Prepushing Models

- **Shared Prepushing**
  - PUSH-S-L1
    - send data in shared state, write it to L1 cache
  - PUSH-S-L2
    - send data in shared state, write it to L2 cache

- **Exclusive Prepushing**
  - PUSH-X-L1
    - send data in exclusive state, write it to L1 cache
  - PUSH-X-L2
    - send data in exclusive state, write it to L2 cache
Prepushing Implementation

```
for i = 0 to N {
    while sync window violation
    wait
    process dataBlock[i]
    signal prepusher
    ...
}
```

CPU 0

```
for i = 0 to N {
    while sync window violation
    wait
    process dataBlock[i]
    ...
}
```

CPU 1

**UCIrvine**
Outline

- Introduction
- Register-Based Synchronization
- Data Communications via Prepushing
- Simulation Environment
- Simulation Results
- Conclusion
Benchmarks

- Red-Black Solver
  - Solves a partial differential equation using a finite differencing method

- Finite-Difference Time-Domain Method
  - Extremely memory intensive electromagnetic simulation

- ARC4 Stream Cipher
  - Stream cipher used in protocols such as SSL and WEP
Simulation Environment

- Simics - Full System Simulator
- GEMS Ruby - Memory Model
- Multi-Core System
  - 2 GHz UltraSPARC III+ Processors
  - L1 Cache: 64 KB, 2-way associative, 1 cycle
  - L2 Cache: 1 MB, 8-way associative, 12 cycles
  - Cache Line Size: 64 B
  - Main Memory: 4 GB, 120 cycles
  - Operating System: Solaris 9
Outline

- Introduction
- Register-Based Synchronization
- Data Communications via Prepulsing
- Simulation Environment
- Simulation Results
- Conclusion
RBS Results

- **Estimated Access Time per Iteration**
  
  $= \text{Average Access Time} \times \text{Access Count per Iteration}$

- **RBS Gain**
  
  $= \frac{\text{Estimated Access Time per Iteration}}{\text{Execution Time per Iteration}}$
RBS Results (cont’d.)

- RB Solver - RBS Gain: 2-5% per iteration

<table>
<thead>
<tr>
<th>Grid Size</th>
<th>200 x 200</th>
<th>400 x 400</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exec. Time per Iter. (cycles)</td>
<td>2,056,725</td>
<td>13,905,891</td>
</tr>
<tr>
<td>Access Count per Iter.</td>
<td>13,531</td>
<td>211,933</td>
</tr>
<tr>
<td>Est. Access Time per Iter.</td>
<td>40,593</td>
<td>635,799</td>
</tr>
</tbody>
</table>

- FDTD - RBS Gain: 6-11% per iteration

<table>
<thead>
<tr>
<th>Grid Size</th>
<th>30x30x30</th>
<th>40x40x40</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exec. Time per Iter. (cycles)</td>
<td>5,741,480</td>
<td>8,124,822</td>
</tr>
<tr>
<td>Access Count per Iter.</td>
<td>120,425</td>
<td>291,506</td>
</tr>
<tr>
<td>Est. Access Time per Iter.</td>
<td>361,275</td>
<td>874,518</td>
</tr>
</tbody>
</table>

- ARC4 - RBS Gain: negligible

<table>
<thead>
<tr>
<th>Stream Size</th>
<th>10 MB</th>
<th>50 MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exec. Time per Iter. (cycles)</td>
<td>1,327,413</td>
<td>1,330,287</td>
</tr>
<tr>
<td>Access Count per Iter.</td>
<td>56</td>
<td>72</td>
</tr>
<tr>
<td>Est. Access Time per Iter.</td>
<td>168</td>
<td>216</td>
</tr>
</tbody>
</table>
- Improvement depends on application behavior
- Exclusive prepushing models are more effective at reducing execution time
Consumer’s L1D Cache Misses

- No improvement in PUSH-S-L2 and PUSH-X-L2 because consumer cannot find its data in L1 cache
- Better than accessing remote cache or main memory
Consumer’s L2 Cache Misses

![Bar Chart]

- SPPM
- PUSH-S-L1
- PUSH-X-L1
- PUSH-S-L2
- PUSH-X-L2

Legend:
- ARC4 - 10
- ARC4 - 50
- FDTX - 20
- FDTX - 30
- FDTX - 40
- RB - 200
- RB - 400

UCIrvine
Consumer’s Shared Data Requests

- Not many explicit shared data requests because consumer receives data beforehand
Consumer’s Exclusive Data Requests

- No need to invalidate producer’s copy in exclusive prepushing models
Conclusion

- RBS and Prepushing improve synchronization and communication support for multithreaded applications.

- RBS employs hardware registers to reduce miss rates and help power savings.

- Prepushing provides an efficient communications interface where data can be moved/copied from one cache to another before it is needed at the destination.