Software Coherence Management on Non-Coherent-Cache Multicores

Jian Cai, Aviral Shrivastava

Compiler Microarchitecture Lab
Arizona State University, USA
Non-coherent-Cache Multicores

- **Coherence**
  - Mechanism by which cores get the latest copy of shared data

  ![Diagram of cache coherency example](image)

- The overheads of cache-coherence increase with the number of cores
  - Each core has to tell every other core about the data it is accessing
  - Area and power of the coherence logic increases dramatically

- Remove the cache coherence logic
  - Intel 48-core Single-chip Cloud Computer
  - Digital Signal Processors, e.g., the 8-core TI 6678
Software Coherence Management

- One way is to disable caches, and execute from main memory
- Software Coherence Management
  - identify the shared data
  - fetch the latest copy of shared data before it is used → by DMA instructions
  - update it
  - merge the updates back
- Overhead of extra instructions
- Performance will be dependent on:
  - the memory consistency model
    - how much we can reorder
  - granularity of management
    - how often we need to read/write
Memory Consistency

- Sequential consistency
  - The order to each read and write to the memory must be maintained
  - no out-of-order execution allowed

- Relaxed consistency
  - Memory accesses that can be reordered are within critical section, surrounded by acquire() and release()
  - requires strict ordering only of acquire() and release()
  - loads and stores can be reordered

**Proc0**
- acquire(lock)
- write x
- release(lock)
- acquire (lock)
- read x
- write y
- release (lock)

**Proc1**
- w(x) rel
- acq r(x) w(y) rel
- acq r(y)
Granularity of Management

- Inter-core coherence is traditionally done in hardware at cache block granularity
  - fine-grain management

- Software cache coherence is typically implemented among processors
  - page-level granularity
  - coarse-grain management

- Inter-core software cache-coherence must be:
  - fine-grain
  - less false sharing
  - fast inter-core communication
State-of-the-Art: COMIC

- Page-level granularity
- Requires a core to serve as coherence manager
- Duplicates pages at acquire, and merges pages at release
Our approach

- Byte-level granularity
- No coherence manager
- No duplicates and merges are required at acquire and release, respectively
Write Notices

- A record of a write (but not the value)
  
  ```
  a write: x = 5;
  write notice: address: &x size: sizeof(x)
  ```

- Write notices can be merged to improve performance
  
  ```
  write1: a[i]++; write notice: address: &a[i] size: sizeof(a[i])
  ```
  
  ```
  write2: a[i+1]++; write notice: address: &a[i+1] size: sizeof(a[i+1])
  ```

  merge

  ```
  write notice: address: &a[i] size: sizeof(a[i])*2
  ```
Performance Improvement

![Bar chart showing Speedup over Disabling Caches for different tasks and techniques. The chart compares COMIC and Our Approach. The tasks include Compress, Laplace, Lowpass, Wavelet, MMT, MV, MM, MT, and Average. The techniques are represented by bars with different colors. The chart illustrates the performance improvement for each task and technique.](image-url)
Management Overhead Reduction

Breakup of Execution

- Coherence Management
- Actual Execution

Breakup of Execution:
- Compress
- Laplace
- Lowpass
- Wavelet
- MMT
- MV
- MM
- MT
- Average

comic vs. our approach
Summary and Future Work

- Overhead of hardware coherence increases dramatically with the no. of cores
- Software implemented coherence on Non-coherent-Cache architectures
  - light hardware,
  - but need to keep the software overhead of coherence management low
- Presented
  - Fine-grain coherence management in software
  - No extra management core required
- Ongoing work
  - Integrate our approach into compiler
  - Currently our work is provided as a run-time library
- Future Work
  - Software coherence management can be extended to SPM-based multicores
    - SPM has 34% less area and consumes 40% less power than a cache of the same capacity
    - Data transfers between main memory and SPM have to be managed by software
If two cores modify the same data, then the updates will be overwritten.

- Yes, but that is what the programmer intended.
- If the programmer wanted the updates to the same data in an order, or exclusively, they would have used locks or barriers.

In parallel programming, the programmer has some responsibility in deciding the order to execution.
- locks provide exclusive access
- barriers provide ordered execution

- If none of present
  - implies that overwrites are acceptable
Why does COMIC need triplicate pages?

Coherence Manager

Core 1

Core 2

original  twin  update  modified  compare

compare  update  modified
Coherence vs. Consistency

- Coherence is about memory accesses to the same address by different cores.
  - One core writes 5 to address A, and another one reads, it will it get the new value or the old?

- Consistency is about memory accesses to different addresses.
  - Core 1 writes to address X first, and address Y second.
  - Can these two writes to the main memory be reordered?
  - If they can be, then another core could read new value to Y, but old value of X?
    - Sequential consistency says no reordering.
    - Relaxed consistency says, between acquire() and release(), all reordering is allowed.
Previous Work: COMIC

Creating and merging duplicates is compute-intensive, but **required** when multiple writers try to modify different locations of the same page (false sharing), to prevent a writer from overwriting modifications of others, which may happen if writers write to the original page directly.
Advantages of Our Approach

- False sharing happens at smallest resource block (such as a cache line or a page) which depends on the granularity of coherence management.
- Our byte-level approach eliminates false sharing, and thus avoid creating duplicates and merging them at acquire and release operations as in COMIC.