Data Movement Control on a PowerPC

Silas Boyd-Wickizer and Asif Khan
What this presentation is about

• Intuition for why multicore caches are underutilized
• Preliminary design for three new instructions
  • Toy benchmarks show improved performance
Caches are crucial for performance
Caches are crucial for performance

250 cycles
Potential solution is one giant shared cache
Potential solution is one giant shared cache

- Applications have access to entire cache capacity, no false sharing issues, etc.
Potential solution is one giant shared cache

- Applications have access to entire cache capacity, no false sharing issues, etc.

50 cycles
Multicore caches are distributed
Multicore caches are distributed

3 cycles
Multicore caches are distributed.
Multicore caches are distributed.

- L1: 3 cycles
- L2: 13 cycles
- L3: 50 cycles
Difficult to use multicore caches efficiently
Hard to access all of on-chip cache

- Size: 64 Kbytes
- Size: 512 Kbytes
- Size: 1.7 Mbytes
- Size: 2 Mbytes
Expensive to access far away caches
Expensive to access far away caches
Expensive to access far away caches

100 cycles
Prototype extensions to hardware: DMC instructions

- **cpush**: store a cache line in another core's cache
- **clookup**: lookup which cache holds an address
- **cmsg**: efficient access to data in another core's cache

- Provide some of the benefits of a single fast shared cache
Prototype extensions to hardware: DMC instructions

- `cpush`: store a cache line in another core's cache
- `clookup`: lookup which cache holds an address
- `cmsg`: efficient access to data in another core's cache

- Provide some of the benefits of a single fast shared cache
Memory hierarchy

- Per-core L1 caches
- Inclusive shared L2
- MSI cache coherence protocol
cpush: copy cache line to another core's cache

- cpush address, core-id
  - Copies cache line at address to core with core-id
**cpush**: copy cache line to another core's cache

- **cpush address, core-id**
  - Copies cache line at address to core with core-id

- If address is marked S in source L1, copy to destination, and mark S.
- If address is marked M in source L1, set source copy to I, copy to destination, and mark M.
- If address is marked I in source L1, ignore
**cpush example: thread migration**

- To migrate thread:
  - source core: saves register values in buffer
  - source core: puts buffer on destination core's run-queue
  - destination core: restores register values to execute thread
cpush example: thread migration

• To migrate thread:
  • source core: saves register values in buffer
  • source core: puts buffer on destination core's run-queue
  • destination core: restores register values to execute thread
cpush example: thread migration

- To migrate thread:
  - source core: saves register values in buffer
  - source core: puts buffer on destination core's run-queue
  - destination core: restores register values to execute thread
- Source core's cache will hold the buffer and thread's working set
cpush example: thread migration

- To migrate thread:
  - source core: saves register values in buffer
  - source core: puts buffer on destination core's run-queue
  - destination core: restores register values to execute thread

- Source core's cache will hold the buffer and thread's working set

- Use `cpush` to move the buffer and thread's working set
clookup: lookup location of an address

- clookup address
  - returns the nearest core ID that caches address
clookup: lookup location of an address

- clookup address
  - returns the nearest core ID that caches address

- If address is M or S in source L1, return source ID
- If address is invalid in source L1, it's marked S or M in L2 directory, return remote ID
- If address is invalid in source L1, and invalid in L2 directory, return -1
**cl**ookup example: cache management run-times?

- Originally implemented `cl`ookup to help test `cmsg`
- Some software run-times try to manage cache contents.
- Maintain a map from object/address to cache
cl ookup example: cache management run-times?

- Originally implemented cl ookup to help test cm s g
- Some software run-times try to manage cache contents.
- Maintain a map from object/address to cache
  - Essentially tries duplicates hardware state
  - Inaccurate
  - Expensive
clookup example: cache management run-times?

- Originally implemented `clookup` to help test `cmsg`
- Some software run-times try to manage cache contents.
- Maintain a map from object/address to cache
  - Essentially tries duplicates hardware state
  - Inaccurate
  - Expensive
- Replace software map with `clookup`
cmsg: efficient access to data in another core's cache

- cmsg address, pc, argument
  - Looks up that core that caches address.
  - Interrupts the core, causing it to execute the function at pc, passing argument as an argument.
cmsg: efficient access to data in another core's cache

- cmsg address, pc, argument
  - Looks up that core that caches address.
  - Interrupts the core, causing it to execute the function at pc, passing argument as an argument.

- If address is M or S in source L1, return 0, drop message
- If address is I in L2, return 0, drop message
- If address is cached in a remote L1, return 1, send message
cmsg: efficient access to data in another core's cache

- **cmsg address, pc, argument**
  - Looks up that core that caches address.
  - Interrupts the core, causing it to execute the function at pc, passing argument as argument.
- If address is M or S in source L1, return 0, drop message
- If address is I in L2, return 0, drop message
- If address is cached in a remote L1, return 1, send message

Cost roughly equivalent to L2 cache miss, or ½ the cost of inter-core miss
Example: shared data structures

- Many applications use shared data structures.
- E.g., Linux uses linked lists in many subsystems.

```c
void do_something()
{
    spin_lock(lock);
    item = list_pop(list);
    update_metadata(list);
    spin_unlock(lock);
}
```
CMSG example: shared data structures

- Many applications use shared data structures.
- E.g. Linux uses linked lists in many subsystems.

```c
void do_something()
{
    spin_lock(lock);
    item = list_pop(list);
    update_metadata(list);
    spin_unlock(lock);
}
```
CMSG example: shared data structures

- Many applications use shared data structures.
- E.g. Linux uses linked lists in many subsystems.

```c
void do_something()
{
    spin_lock(lock);
    item = list_pop(list);
    update_metadata(list);
    spin_unlock(lock);
}
```

Three cache misses:
- `list_entry`
- `list_entry->next->prev`
- `list_entry->prev->next`
cmsg example: shared data structures

- Many applications use shared data structures.
- E.g. Linux uses linked lists in many subsystems.

```c
do_something:
spin_lock(lock);
item = list_pop(list);
update_metadata(list);
spin_unlock(lock);
```

- Source calls cmsg(lock, do_something) and destination calls cmsg(sourceId) to reply.
- About the cost of one inter-core cache miss.
DMC implementation

- Modified an existing PowerPC implementation
  - Runs as on an FPGA as a cycle accurate simulator
  - FPGA is important
- Added/modified about 1000 lines of BSV
- Wrote a software run-time in about 2000 lines of C for testing and benchmarking
Preliminary evaluation

- Can software improve performance using DMC instructions?
  - Thread migration benchmark
  - List manipulation benchmark

- Dual core, L2 access 31 cycles, DRAM access 255 cycles

- Caveats: no hardware pre-fetching, no SMT, etc.
cpush improves thread migration performance
cpush improves thread migration performance

- Pushing 6 cache lines cuts latency in half
- For more than 6, the messages FIFOs fill up
cmsg improves the performance of list operations
cmsg improves the performance of list operations

- Reduce update latency by $\frac{1}{2}$
cmsg improves the performance of list operations

- Reduce update latency by \( \frac{1}{2} \)
- More benefit as benchmark updates more meta-data
Future work

- closg loose ends
  - How to handle multiple address spaces?
  - How to deal with fairness?
- When shouldn't applications use DMC instructions?
- Experiment with real applications
Related work

- Managing multicore caches
- Computation migration systems
Conclusion

• Three new instructions (cpush, clookup, and cmsg) for managing cache contents
• Promising preliminary results
• Next step: generalize to real workloads