Locking

- Synchronization primitives called locks are created in shared memory. When a core wishes to access a location in shared memory, it must first acquire the lock associated with that location.
- Cores that want the lock check repeatedly until it becomes free. Only one core is permitted to acquire the lock at a time, so the second core to have the lock will always see the changes made by the first.
- The association between the lock and the memory location is established by the programmer, so it is important to be careful when choosing what to lock and determining which locks are necessary.

Transactional Memory

- Programs are divided into transactions, groups of memory accesses (reads and writes) that access shared memory and depend on each other.
- Processor cores optimistically execute transactions, caching the changes they make instead of writing them to memory, in case there is a conflict.
- If the end of the transaction is reached without problem, the core commits it, and the values in the cache become the new values for those memory locations.
- If two transactions perform memory accesses that conflict, the transactional memory hardware detects this condition and aborts one of the transactions. The core running this transaction discards all of the memory changes that were made during the transaction, and sleeps for several cycles before retrying the transaction.

Virtual platform

Hardware transactional memory designs are tested on a simulated embedded hardware platform called MPARM.

Some features of this platform:
- Support for ARM cores, similar to the ones used in many popular mobile phones.
- Each core has a 4-way associative, transactional L1 cache.
- External per-core private memories, single shared memory.
- AMBA bus for communication with memory—can transmit one address and one word of data per clock cycle.
- Coherence maintained among caches by bus snooping.
- Can be configured to use locking or transactional memory.

The Bloom filter module

Although conflict detection can be implemented as part of the cache coherence protocol, this requires extensive modifications to a protocol that, when unmodified, is known to be reliable.

A simpler way is to incorporate dedicated hardware that detects conflicts.

Bloom filters are a data structure with very low space requirements and latency. However, they can sometimes produce false positives, reporting that they contain an entry that was never actually inserted.

Evaluation

Some success:
- Bloom filter module has been successfully incorporated into the simulated platform.
- Allows removal of 764 lines of code that modify core hardware protocols that are known to work.
- Instead, uses a relatively simple, standalone module.
- Capable of supporting multiple conflict-resolution policies.
- Initial worries about system performance were alleviated by limiting extra bus traffic for bloom filter module to the ones that pointed to shared memory addresses.

Some remaining performance issues:
- Bloom filters produce false positives, which the bloom filter module interprets as conflicts, unnecessarily increasing the abort rate.
- Supporting the bloom filter module requires more bus traffic than the previous system, as shown in Figure 6.
- Extra bus accesses are needed to inform the bloom module of changes in the internal states of the caches.

Application—Mobile Devices

This research discusses improvements to a multi-core embedded system, using ARM processors like those found in these phones.

Literature


For further information

Email blipt01@swarthmore.edu