Computer Architecture III Study Guide Additional Topics

Maximillien Courchesne-Mackie (mcour010@uottawa.ca)

I. Chip Multiprocessors (CMP)

Chip multiprocessors have several advantages over symmetric multiprocessors (SMPs). Seeing as they have multiple cores on a single chip, there is reduced latency. This also means that it is also good for resource sharing. Most CMPs have a NUMA or cc-NUMA architecture with shared L2 cache and private L1 caches. SMP systems, on the other hand, are better suited for specific tasks.

II. Core Multithreading

Multithreading in CMPs can be executed in hardware of software.

A. *Software Multithreading*

Software mlutithreading segregates processes and gives them a chunk of virtual memory and a process control block (to save its state in the event of a context switch). A process is considered active when it is ready for execution and inactive when it is not.

When blocks , the process is switched out by the processor and is put on a waiting list with other processes. The maximum amount of time a process can run is dictated by the *time quantum*.

In the event of a context switch, the following events must take place. The process is trapped, the pipeline is flushed, the process state is saved in its process control block and the state of another process is restored.

B. *Hardware Multithreading*

Hardware multithreading is more efficient than its software counterpart. The fact that it is implemented in hardware reduces latency. Multithreading of this kind can be separated into three categories which are discussed in the next section.

III. Types of Multithreading

A. *Block Multithreading*

In block multithreading, pipeline stages without states are shared dynamically. Stages with hard states (PC, registers) are replicated and those with soft states (TLB, caches) are partitioned.

When block multithreading is implemented in an out of order core, each thread has its own IFQ. The processor fetches instructions for the current thread until it blocks. When this happens, another thread takes over. The results are placed on the common data bus (CDB) and the second thread must block or finish for the first to resume execution.

B. *Interleaved Multithreading*

In interleaved multithreading several threads share the core at one time. Thread flushing must then happen at specific pipeline stages (instead of flushing the whole pipeline). The thread selector chooses a new thread every clock cycle.

Barrel processors are a hypothetical interleaved architecture which has a very large number of threads to choose from. This essentially hides almost all latencies. However, this architecture is currently infeasible because no system actually has enough work to warrant a huge amount of threads. This design is also expensive and the execution of each thread is still slow. However, the pipeline is always full and working.

C. *Simultaneous Multithreading*

Simultaneous multithreading (also called Hyperthreading by Intel) fetches all threads every clock cycle and implements them in several functional units. Every instruction has a TID (thread ID) so it can be tracked across the different pipeline stages and stage flushing has to be thread aware. Instruction scheduling must also be thread aware.

IV. CMP Classifications

A. *Network Type*

Bus-based CMPs use a central bus as the interconnection network between nodes. Each core shares L2 cache banks through this bus which is kept coherent with private L1 caches using snooping.

Ring-based CMPs are able to be clocked higher than bus-based solutions due to their point-to-point architecture. Cache coherence is kept using a snooping or directory protocol (more on this in the next section). An example of a ring-based chip multiprocessor is Intel's Core i7 series.

Crossbar-based CMPs are more scalable than bus-based or ring-based architectures. They use a directory cache coherence protocol.

B. *Cache Coherence with Shared Cache*

There are essentially two approaches to directory-based cache coherence protocols with shared-cache CMPs: presence flag vector protocol and L1-map directory protocol.

In presence flag vector protocol, $n+1$ bits are needed for the presence vector (one bit for each processor plus one dirty bit). One vector is needed for each line of L2 cache in the system. This is not a very efficient system because the directory spans the entirety of L2 when most values that are in L2 aren't in L1 anyway. Thus, there is a lot of wasted space.

In the L1-map directory protocol each L2 bank has a directory and its size is proportional to the number of lines in the L1 cache that maps to that particular bank. The number of bits needed per directory entry are:

$$(\#\text{L2 lines} - \text{L1 index size}) + 2$$

The two added bits represent the state of the block in the L1 cache.

V. CMP Programming Models

To fully take advantage of CMPs, software developers must adopt programming technique to leverage the different multhreading architectures.

A. Independent Processes

Independent processes have parallelism limited by user activity. However, the seeing as the threads are independent, the sharing benefits of CMPs are lost. Instead, processes compete for resources.

B. Explicit Thread Parallelism

Explicit thread parallelism is when one application is split into several threads which can be managed by the developer. A popular thread management API is the Pthreads library.

C. Transactional Memory

Transactional memory programming removes the need for data locking by using lock-free data structures. In the event of a conflict (two threads accessing the same datum in a structure), one of the transactions is rolled back and doesn't get executed. Seeing as transactions are atomic, this is not a problem. The thread can try the transaction again until it gets executed. Another benefit of this programming model is that parallel loads of a datum in a lock-free data structure is allowed as long as no process requests a store.

Conflicts (for example, two threads attempt a store at the same location) are detected according to the conflict detection policy: eager or lazy. Eager detection happens at the moment the conflict happens. Lazy detection happens after the conflict has arisen.

Two new instructions must be added to the ISA to support TM: *TBegin* and *TEnd*. TBegin will switch the processor into transactional mode by setting a state bit (called transaction_active) in that processor. TEnd resets that bit.