Fiche Introduction to multiprocessor architecture

Pierre Colson

Saturday 30 October

Contents

Markdown version on *github*

General

- $P = CV^2f$ P = **Power**; V = Operating voltages; f = Operating voltages; C = Capacitance
- **Dennard's Law** Voltages used to go down
- **Moore's Law** is the observation that the number of transistors in a dense integrated circuit (IC) doubles about every two years. Moore's law is an observation and projection of a historical trend. Rather than a law of physics, it is an empirical relationship linked to gains from experience in production.
- Processor & core used interchangeably
- Cores run threads
- Each chip has multiple cores
- Each board can have multiple chips
- Each platform can have multiple boards

Parallelism

- **Amdahl's law** : if you speed up only a small fraction of the execution time of a computation, the speedup you achieve on the whole computation is limited!

$$speedup = \frac{1}{\frac{Fraction_{enhanced}}{Speedup_{enhanced}} + (1 - Fraction_{enhanced})}$$

Coherence

- Cache coherence makes sure all copies of one address have the same value
- Cache set = cache line
- Cache block : number of data in a line
- **Compulsory** misses Access data for the first time
- **Capacity** misses when the cache is not big enough
- **Conflict** misses when two addresses map to the same set and way of the cache, evicting one of them
- Naïve Solution – Valid and Invalid Protocol

- **MSI**
  - Modified (Dirty) This cache has updated the value and holds the authoritative copy
  - Shared (Valid) One or more caches hold the value, read permission only
  - Invalid
- **MESI** protocol reduces bus contention by eliminating BusInv messages
- **Directory Based Protocol** his directory to keep track of the status of all cache blocks, the status of each block includes in which cache coherence "state" that block is, and which nodes are sharing that block at that time, which can be used to eliminate the need to broadcast all the signals to all nodes, nd only send it to the nodes that are interested in this single block.

Optimization

- **Coherence** misses when a block is removed due to coherence messages from another core
- **True sharing** when processors modify a same cache block (same data); To optimize we can :
  - Divide up input data in advance
  - Privatize results when possible, reduce communication
- **False sharing** Processors updating different data which happen to be in the same cache block
- **The Locality Principle**
  - Problem: Bad page replacement policies led to swapping and unusable machines
  - Solution: Design memory to prioritize the "working set" of the currently executing applications

Consistency

- Types of Memory Dependences
  - Read After Write (RAW)
  - Write After Read (WAR)
  - Write After Write (WAW)
- **Load Store Queue** (LSQ)
  - Hold all store operations until they retire
- **Store buffer** buffer for store operations ^^
- **Sequencial consistency** : Exucute things in program order
- **Optimized sequencial consistency** : Allow peek
- **Processor consistency** (PC) : Reads can bypass write (used today by x86)
- **Weak consitency**
  - Only synchronization operations have any ordering
  - Data ops. have no order enforced among themselves
  - Weak Consistency is used in ARM
  - Synch. instructions are called Fences
    - Enforces program order for operations before/after
- Basic consistency models: SC, PC, Weak

Synchronzation

- Mutual exclusion
  - **Locks**
    - Test and set (**TS**): need to write and read in one operation to acquire the lock ⇒ bus overload
    - Test and Test and Set (**TTS**) : Do a read before doing the TS ⇒ no trafic while waiting
    - Exponential back off : add a delay before re-testing
    - Load lock & Store Conditional (**LL/SC**) ⇒ simple way to build a test and set ⇒ relies on atomicity
- **Lock** Caracteristics
  - Low latency
  - Low traffic
  - Scalability
  - Low storage cost

  - Fairness
- Point to point synchronization
  - Flags, **barriers** (uses lock, counter and flag)
- Sowftare methods
- **Coarse grained lock** ⇒ lock an entire list for example
- **Fine-grained lock** ⇒ a lock by element
- **Hardware Lock Elision**
- **Atomicity**
  - Upon transaction commit, all writes take affect at once
  - On transaction abort, no writes take effect
- **Isolation** : not other processor can observe writes before commit
- **Serialazability**
  - Transactions seem to commit in a single serial order
  - The exact order is not guaranteed though
- **Transactional memory** HLE whitout lock (software atomic section)

Multithreading

- Out of order pipeline ⇒ Misses do not block pipeline
- **Vertical waste** : whole cycle empty, nothing issued
  - Most common after long latency events
- **Horizontal waste** : unable to use the full issue width
  - Software not exposing enough ILP for hardware
- **Blocked (Coarse Grain) Multithreading** (CGMT) : Switch to a new thread on a long latency event
- **Fine grained Multithreading** (FGMT) : Cycle between threads periodicall (every cycle for ex)
- **Simultaneous Multithreading** (SMT) : Instructions from multiple threads in same cycle
  - Thread with fewest instructions in pipe has priority
- $IPC = \frac{\#instructions}{\#cycles}$

GPUs

- **GPU** has Processing clusters
- **Processing cluster** has Streaming Multiprocessors
- **Streaming multiprocessor** has Thread block
- **Thread block** has wrap
- **Wrap** has threads
- CUDA programming : Create a Kernel, copy memory, invoke, copy back
- **Control flow Divergence** : Thread in a wrap might execute different path (branch condition for ex)
- Memory System
  - **Global memory**
    - Shared by all threads
    - GPU main memory
    - Hight bandwith
  - **Shared memory**
    - Shared within a thread block
    - Manage by programmer
    - very fast
    - Inter thread communication
    - Several bancks (ideal one per thread)
  - **Registers**
    - Stack variable declared in kernels
    - Fastest access to data
  - Constant and texture : mainly for graphic
- Highly divergent code leads to poor performance

- Optimizing code
  - **Coalesced memory access** :
    - multiple memory access in one memory transaction
    - Threads in a wrap access the same cache line
  - Eliminate wrap divergence
  - For memory intensive code, reduce excess instructions
  - Consider work division per thread
- Optimization
  - Reduce thread divergence by grouping thread from the same wrap in condition
  - Shared memory
  - Thread indexing $\Rightarrow$ threads that are working are now in the same wrap
  - Sequencial mapping of accesses to the banks guarenties that we don't have bank conflict ??
  - Reduce during load from global memory