

ECE 574 – Cluster Computing

Lecture 9

Vince Weaver

`http://web.eece.maine.edu/~vweaver`

`vincent.weaver@maine.edu`

23 September 2026

Announcements

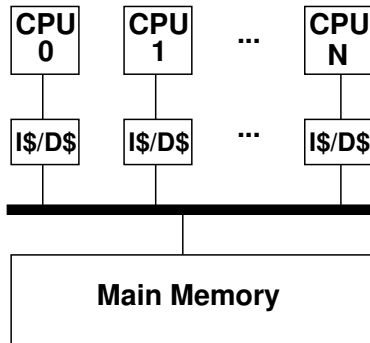
- Don't forget HW#3

More Computer Arch Review

Multicore / Shared-Memory Systems

- Moore's Law can't make systems faster, so the extra transistors are used for more cores
- Single Package: CMP (Chip-multiprocessor) or SMP (Symmetric-multiprocessor)
- Multi-package: Multiple CMP packages in system.
- These are called “shared memory” systems, one memory all cores write to, all cores can see reads/writes by all

CMP Diagram



Cache Coherency

- How do you handle data being worked on by multiple processors, each with own cache of main memory?
- Cache coherency protocols.
- Many and varied. MESI is a common one

Cache Coherency Implementations

- Directory – centralized piece of hardware that monitors memory accesses
- Snoopy – hardware that “snoops” the memory accesses to track where things are, often distributed to each core

MESI (Modified, Exclusive, Shared, Invalid)

- If just reading, all cores are fine (memory is same) so can live in shared state
- Writing is the problem. Before writing, need to request “exclusive” state, which will invalidate the copies on all other cores
- Once exclusive, can write which leads to “modified”
- Once write back to memory, can return to “shared” state
- This is all done magically in hardware so in theory software doesn't need to worry about it

Can Cache Coherency Affect Performance?

- If lots of threads writing to same location, can get ping-ponging where a cacheline bounces around between cores
- False Sharing – if a highly contested variable is on the same cacheline as unrelated data, accesses to the unrelated data might be slower than expected

Memory Models

- For performance reason both hardware and software might try to re-arrange loads and stores to be faster
- In theory this is allowed as long as the original program's behavior does not change
- Different processors might handle this differently
- See here for a good reference:
<https://arangodb.com/2021/02/cpp-memory-model-migrating-from-x86-to-arm/>

Language Memory Models

- For a while this was undefined, but things like C++11 have added standards for this
- Why would your compiler mess with loads/stores?
- One example:

```
x=1; x=2; x=3;  
printf("%d\n",x);
```

- Can the compiler tell nothing uses the first two writes, and remove them for performance reasons?
- What if the address of x is in MMIO memory region?
Then read and write might have side effects
Traditionally in C at least `volatile` or inline assembly is used to prevent the compiler doing this

Order That Loads/Stores happen

- Source Code Order
- Program Order
- Execution Order
- Perceived (hardware) Order
Impacted by caches, etc

Sequential Consistency

- Formalized by Lamport (1979)
- Simplest, pick arbitrary thread and execute next instruction
- Load/Stores seen are interleaved version of all threads running
- Easiest to logic about, difficult to implement on modern processors

x86-TSO (Total Store Ordering)

- x86 is not well documented but people have proposed x86-TSO as a model
- Each thread has store buffer, reads check this before accessing cache/main-memory
- Other threads not see store buffers until written to cache/memory
- mfence instructions flush out store buffers
- lock/atomic instructions take global lock across all threads, and flush before releasing
- stores can propagate any time when no lock held

Relaxed PPC/ARM

- A lot of RISC processors have relaxed ordering
- No limits at all, if you need a store to be seen need to manually have an instruction for it
- Much simpler hardware, higher performance, lower power, but more difficult to code
- Writes don't have to appear to all threads at once
- ARMv8 later versions are a bit stricter

Strong vs Weak Consistency Issues

- Code written on x86 might only work due to the implicit strong
- Might break on ARM
- Vendors that care about this, or doing emulation (Steam deck with emulated games, Apple with emulated software) must do extra work
- Apple M chips actually have an x86 compat mode they can switch to to make this easier

Barriers and Ordering

- A memory barrier instruction makes sure all previous loads/stores finish before moving on
- Most important for things like locks, as well as memory-mapped I/O

Hardware Multi-Threading

- Idea is to re-use a pipeline to execute multiple threads at once, *without* fully replicating the entire CPU (so less than multicore)
- You will have to replicate some things (program counter for each, etc)
- Usually they appear to the CPU as full separate processors even though they are not.

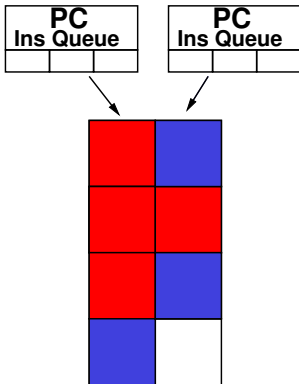
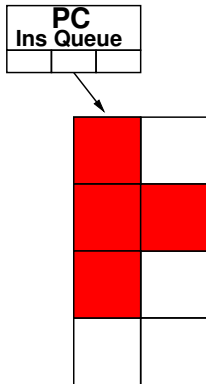
Hardware Multi-Threading 2

- Various ways to do this:
 - Fine-grained – rotate threads every cycle
 - Coarse-grained – rotate threads only if long latency event happens (cache miss)
 - Simultaneous – issue from any combination of threads, to maximize use of pipeline (have to be superscalar)

Hardware Multi-Threading 3

- Why do this? Often on superscalar running only one thread will leave parts idle, try to make use of these.
- Bad side effects?
Can actually slow down code (especially if both threads trying to use same functional units, also if both using memory heavily as cache is often shared)
- Sometimes see it talked about as SMT (Simultaneous Multithreading), Intel Hyperthreading is more or less the same thing
- Modern security issues, leak info between threads

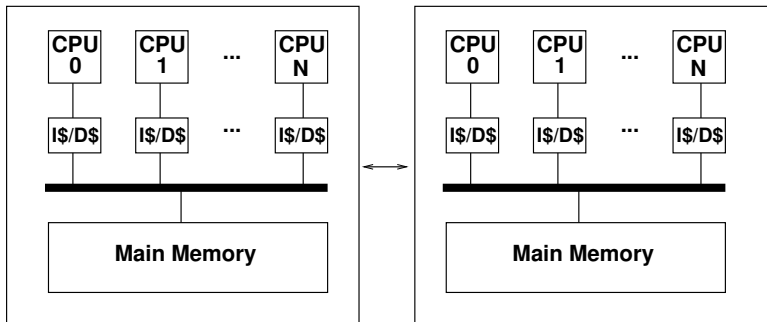
SMT Diagram



Non-Uniform Memory Access (NUMA)

- Random Access Memory – arbitrary memory accesses take same amount of time
- Is that true anymore/
- NUMA: some accesses will have to cross to other processors, causing extra delay
- How can you optimize this?

Traditional NUMA Layout



Who is responsible for NUMA Performance

- Operating system generally tries not to move threads around, but sometimes it will
- You can “pin” or “bind” a thread to a certain core on Linux
- You can also specify sometimes in the job control software

Haswell EP Info

- 1-U server, HP Proliant DL360 Gen 9
- Intel Xeon E5-2460v3 at 2.60GHz
- Two sockets, each package has 8 cores with 2 threads each
16 cores (32 threads)
- 80GB DDR4 RAM, 20MB L3 cache each socket
- NVIDIA GPU GP107GL (Quadro P400)

Maintaining Old Servers

- Due to Moore's law slowing, increasingly old servers are still being used
- There are three variants of x86_64: v1, v2, v3
- Some Linux distributions have been dropping support for the older versions
- haswell-ep is just barely v3 so safe for now, but eventually I'll have to move this class to newer hardware

Finding Detailed Stats on Modern CPUs

This is not always easy. There are papers, and manuals at the vendor websites, but these don't always present a clear picture.

One good resource is Agner Fogg's various optimization guides: <http://www.agner.org/optimize/>

Haswell EP Setup

