ECE 574 – Cluster Computing
Lecture 6

Vince Weaver

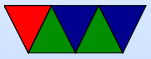
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

7 February 2017

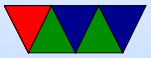
Announcements

- Homework #4 was posted. Can work in groups of two. Don't put it off until the last minute!

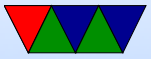


Homework #3 Review

- 2
  - 2a) 146 89 61 48
    2:26 1:19 1:01 0:48
  - 2b) Speedup: (to/tn) 1.6 2.4 3.0
  - 2c) Parallel effic: (Sp/p or T1/pTp) 82% 60% 40%
  - 2d) Yes, time decreases as you add cores.
    Not ideal strong scaling though.
  - 2e) No weak, didn't test with sizes constant
- 3



- 3a) dgemm kerenel (double-precision generic matrix-matrix multiply. algorithm kernel (core) not Linux kernel)
- 3b) vmovups (truncated) (memcpy?)
- 3c) skid



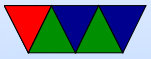
Homework #3 More

Why does it drop off at 8?

If ideal strong scaling, then parallel efficiency would be closer to 1. Not enough results for weak scaling.

What is the worst case parallel efficiency (i.e. a single-threaded program so adding more cores does not help?). Is this truly the worst-case?

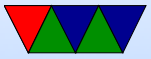
Perf record, make sure running it on benchmark, i.e. `perf record ./xhpl`
If you run on time, or the shell script you will still get



the right results because perf by default follows all child processes. However if you run on sbatch, it won't work, as sbatch quickly sets things up and notifies the scheduling daemon via a socket connection to start things, then exits. In that case perf will only measure sbatch and not your actual benchmark run.

`perf report` will show a profiling breakdown at the function level:

```
Samples: 688K of event 'cycles', Event count (approx.): 565000306359
Overhead  Command  Shared Object       Symbol
  70.74%  xhpl     xhpl                [.] dgemm_kernel
   9.77%  xhpl     xhpl                [.] HPL_lmul
   1.88%  xhpl     xhpl                [.] HPL_rand
```

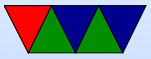


```
1.74%  xhpl      xhpl                  [.] HPL_dlaswp00N
1.29%  xhpl      xhpl                  [.] HPL_ladd
1.29%  xhpl      xhpl                  [.] HPL_pdlange
1.14%  xhpl      [kernel.vmlinux]      [k] entry_SYSCALL_64
```

Pressing enter or using `perf annotate` will show at the asm level:

```
0.02    c0:    vmovups (%rdi),%ymm1
0.72           vmovups 0x20(%rdi),%ymm2
0.07           vmovups (%r15),%ymm3
1.48           vmovups %ymm1,(%rsi)
0.02           vmovups %ymm2,0x20(%rsi)
0.02           vmovups %ymm3,0x40(%rsi)
```

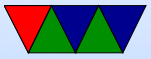
move unaligned packed 256-bits from memory to register (single precision?) memory copy? By default uses :ppp to reduce skid



Types of Clusters

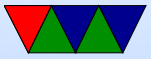
- Shared-memory: many CPUs, but one shared memory address space. Usually one copy of operating system. When write to memory, all CPUs can see it.

- Distributed: man systems spread across network. Each has own memory. For other CPUs to see data have to send message across network.

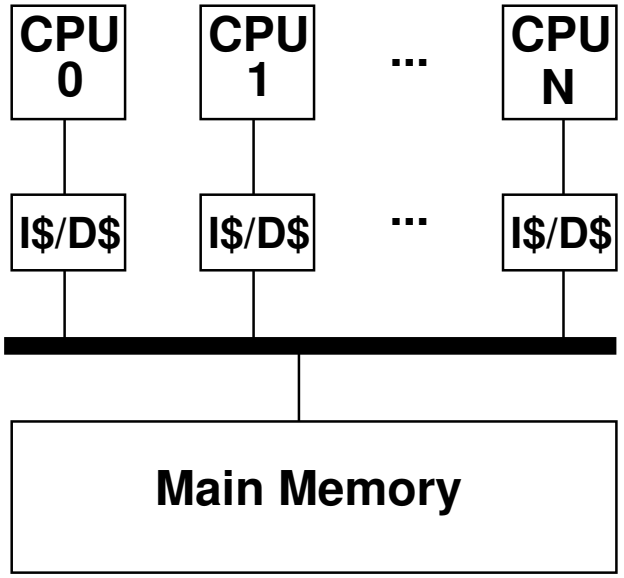


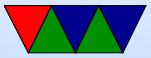
Multicore Systems

- CMP (Chip-multiprocessor) or SMP (Symmetric-multiprocessor)
- Multiple processors in system. More on this later.



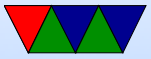
CMP Diagram





Hardware Multi-Threading

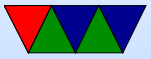
- Idea is to re-use a pipeline to execute multiple threads at once, *without* fully replicating the entire CPU (so less than multicore)
- You will have to replicate some things (program counter for each, etc)
- Usually they appear to the CPU as full separate processors even though they are not.
- Various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

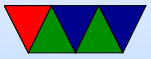
- Why do this? Often on superscalar running only one thread will leave parts idle, try to make use of these.

- Bad side effects?
  Can actually slow down code (especially if both threads

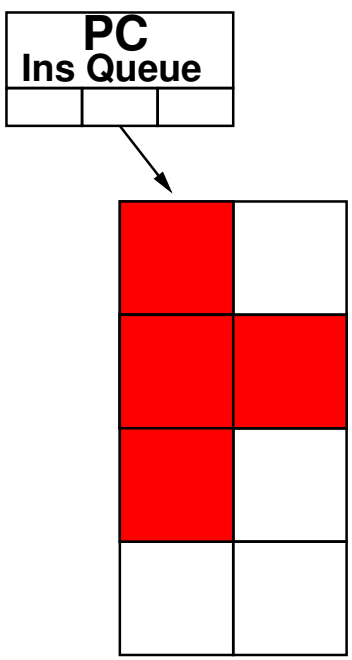


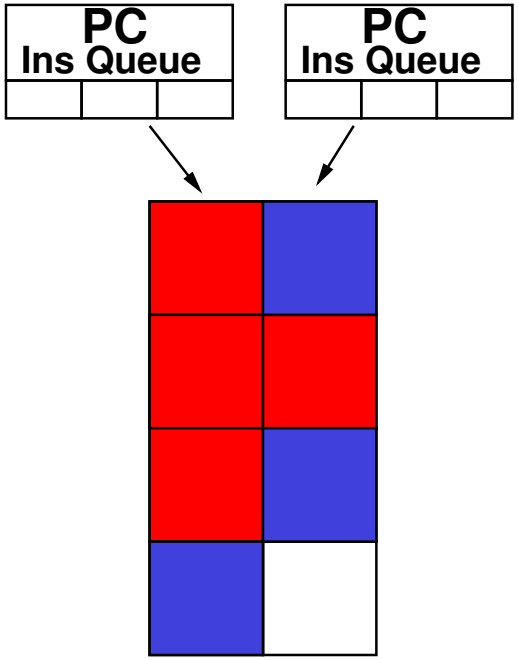
trying to use same functional units, also if both using memory heavily as cache is often shared)

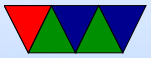
- Sometimes see it talked about as SMT (Simultaneous Multithreading), Intel Hyperthreading is more or less the same thing



SMT Diagram

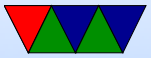






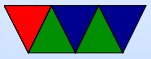
Cache Coherency

- 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
- Directory vs Snoopy



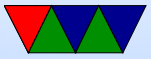
MESI

- Modified, Exclusive, Shared, Invalid



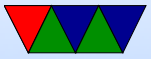
Barriers and Ordering

- On modern out-of-order execution, memory accesses can happen out-of-order
- Sequential consistency – all happen in order
- Strong consistency – stores
- Weak consistency – can be arbitrarily reordered, only barriers protect you
- 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



Ordering Example

```
    y1=0
    y2=0
    y1=3
    y2=4
Another core
    x1=y1
    x2=y2
```

What values of x1 and x2 can you get?

Strong:

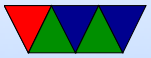
x1=0,x2=0

x1=3,x2=0

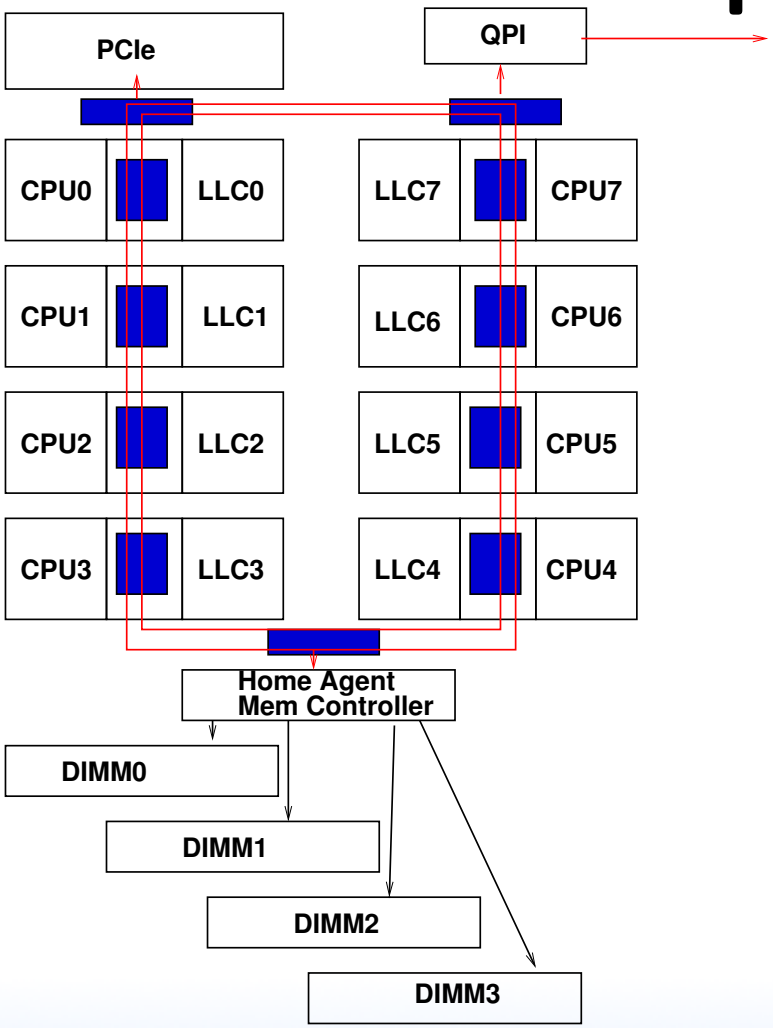
x1=3,x2=4

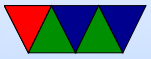
Weak:

x1=0,x2=4



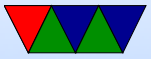
Haswell EP Setup



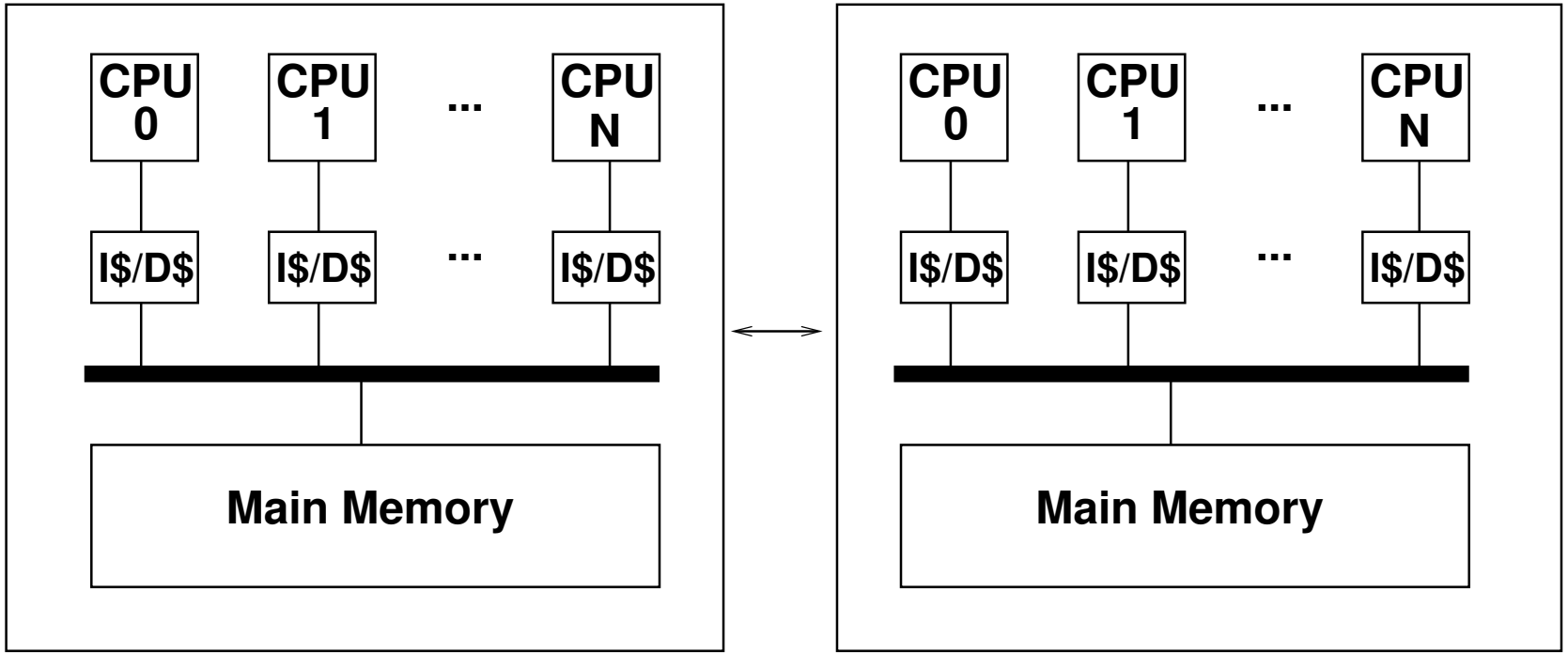


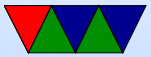
NUMA

Non-uniform memory access – some accesses will have to cross to other processors, causing extra delay. How can you optimize this?



Traditional NUMA Layout

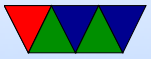




Parallel Programming!

- Simplest: process based. Just have multiple (maybe unrelated) processing running on system. Parallel make. Usually can scale as long as you have processes to run (but do most people ever have more than 2-4? Web-browser, virus scan, camera ap, virus?). Can communicate between separate processes but is slow, poor interfaces. But can run equivelent of distributed system on a shared-memory computer.

- Next: thread based. When you take a single program



and spread it across multiple cores.