ECE 574 – Cluster Computing
Lecture 4

Vince Weaver

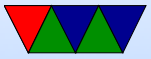
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

31 January 2017

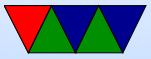
Announcements

- Don't forget about homework #3
- I ran HPCG benchmark on Haswell-EP machine.
  Theoretical: 16DP FLOP/cycle * 16 cores * 2.6GHz = 666 GFLOPS
  Linpack/OpenBLAS: 436 GFLOPS (65% of peak),
  HPCG: 0.7 GFLOPS (0.1% of peak)



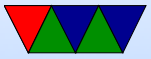
Running Linpack

- HPL solves linear system of equations, Ax=b. LU factorization.
- Download and install a BLAS. ATLAS? OpenBLAS? Intel?
  Compiler? intel? gcc? gfortran?
- Download and install MPI (we'll talk about that later). MPICH? OpenMPI?
- Download HPL. Current version 2.2?
  Modify a Makefile (not trivial) make sure links to proper

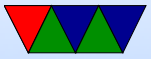


BLAS. make arch=OpenBLAS

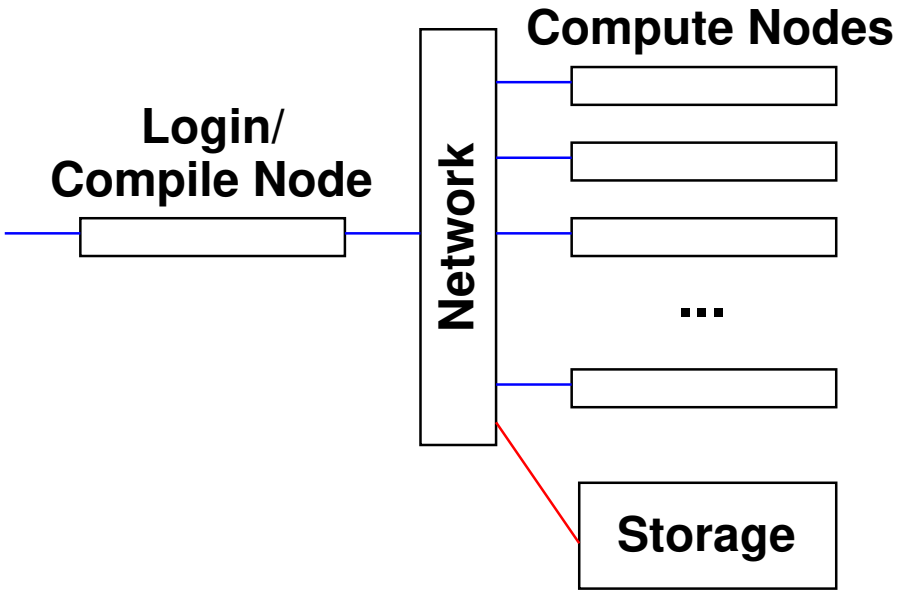
- Above step, might need to create a link from hpl in your home directory to actual location for reasons
- Create a a bin/OpenBLAS with default HPL.dat file
- Run it `./xhpl` Or if on cluster `./mpirun -np 4 ./xhpl` or similar.
- Result won't be very good. Need to tune HPL.dat
- N is problem size. In general want this to fill RAM. Take RAM size, squareroot, round down. NxN matrix. Each N is 8 bytes for double precision.
- NB block size, can be tuned



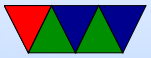
- PxQ, if on cluster can specify machine grid to work on. Linpack works best with as square as possible.
- Fiddle with all the results until you get the highest.



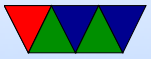
Commodity Cluster Layout



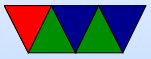
- Simple cluster like the pi-cluster, or older ones I've made
- Commodity cluster design is a combo of ECE331/ECE435 more than anything else



- Why have a head node?
- What kind of network? Ethernet? Inifiniband? Something fancier?
- Operating system? Do all nodes need a copy of the OS? Linux? Windows? None?
- Booting: network boot, local disk boot.
- Network topology? Star? Direct-connect? Cube? Hyper-cube?
- Disk: often shared network filesystem. Why? Simple: NFS (network file system). More advanced cluster filesystems available.

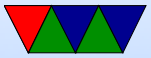


- Don't forget power/cooling
- Running software?

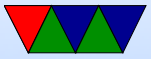


Job Schedulers

- On a big cluster, how do you submit jobs?
- If everyone just logged in to nodes at random, would be a mess
- Batch job scheduling
- Different queues (high priority, long running, etc)
- Resource management (make sure don't over commit, use too much RAM, etc)
- Notify you when finished?
- Accounting (how much time used per user, who is going

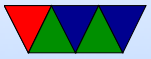


to pay?)

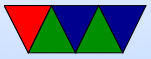


Scheduling

- Different Queues Possible – Low priority? Normal? High priority (paper deadline)? Friends/Family?
- FIFO – first in, first out
- Backfill – bypass the FIFO to try to efficiently use any remaining space
- Resources – how long can run before being killed, how many CPUs, how much RAM, how much power? etc.

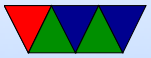


- Heterogeneous Resources – not all nodes have to be same. Some more cores, some older processors, some GPUs, etc.



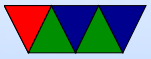
Common Job Schedulers

- PBS (Portable Batch System) – OpenPBS/PBSPro/TORQ
- nbs
- slurm
- moab
- condor
- many others



Slurm

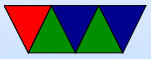
- `http://slurm.schedmd.com/`
- Slurm Workload Manager
  Simple Linux Utility for Resource Management
  Futurama Joke?
- Developed originally at LLNL
- Over 60% of top 500 use it



sinfo

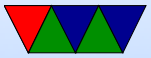
provides info on the cluster

```
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
debug        up   infinite      1   idle haswell-ep
general*     up   infinite      1   idle haswell-ep
```



`srun`

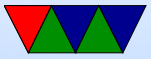
start a job, but interactively



sbatch

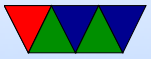
submit job to job queue

```
#!/bin/bash

#SBATCH -p general        # partition (queue)
#SBATCH -N 1                # number of nodes
#SBATCH -n 8                # number of cores
#SBATCH -t 0-2:00         # time (D-HH:MM)
#SBATCH -o slurm.%N.%j.out # STDOUT
#SBATCH -e slurm.%N.%j.err # STDERR
export OMP_NUM_THREADS=4
./xhpl
```



squeue

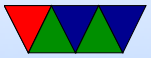
```
JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
   63   general time_hpl ece574-0 PD       0:00      1 (Resources)
   64   general time_hpl ece574-0 PD       0:00      1 (Resources)
   65   general time_hpl ece574-0 PD       0:00      1 (Resources)
   62   general time_hpl ece574-0  R       0:14      1 haswell-ep
```



scancel

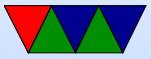
kills job

```
scancel 65
```

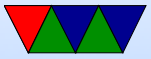


Some sample code

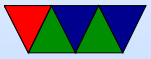
```
int x[8][8];

for(i=0;i<8;i++) {
    for(j=0;j<8;j++) {
        x[i][j]=0;
    }
}
```



```
    mov r0,0          ; i
i_loop:
    mov r1,0          ; j
j_loop:
    mov r3,0
    mov r4,x
    add r4,r4,r1,lsl#5
    add r4,r4,r0,lsl#3
    str r3,[r4]
    add r1,r1,#1
    cmp r1,8
    blt j_loop
    add r0,r0,#1
    cmp r0,8
    blt i_loop
```

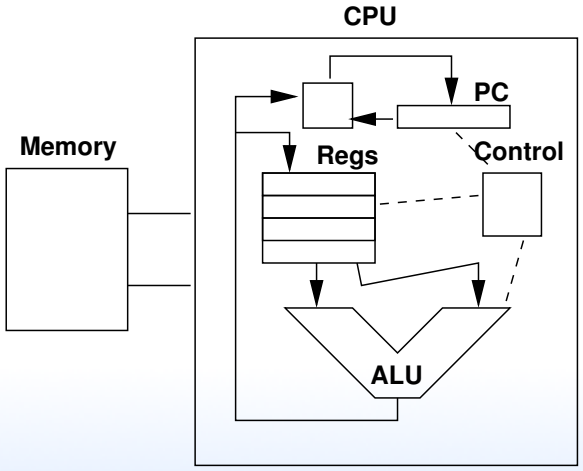


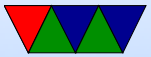
Parallel Computing – Single Core



Simple CPUs

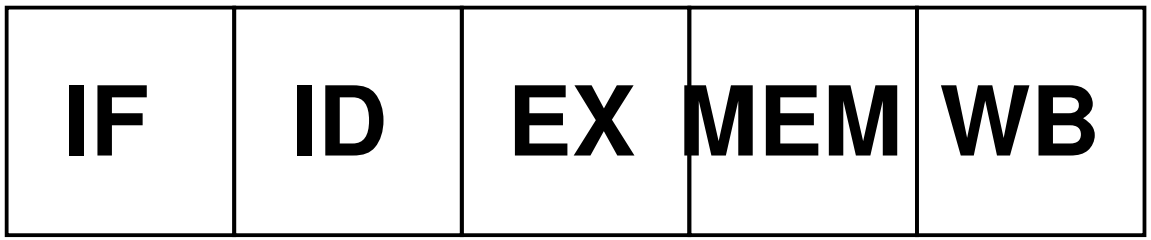
- Ran one instruction at a time.
- Could take one or multiple cycles (IPC 1.0 or less)
- Example – single instruction take 1-5 cycles?

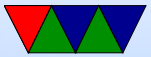




Pipelined CPUs

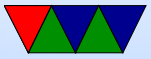
- 5-stage MIPS pipeline
- From 2-stage to Pentium 4 31-stage
- Example – single instruction always take 5 cycles? But what about on average?





Pipelined CPUs

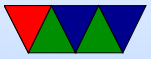
- IF = Instruction Fetch.
  Fetch 32-bit instruction from L1-cache
- ID = Decode
- EX = execute (ALU, maybe shifter, multiplier, divide)
  Memory address calculated
- MEM = Memory – if memory had to be accessed, happens now.
- WB = register values written back to the register file



Data Hazards

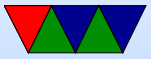
Happen because instructions might depend on results from instructions ahead of them in the pipeline that haven't been written back yet.

- RAW – "true" dependency – problem. Bypassing?
- WAR – "anti" dependency – not a problem if commit in order
- WAW – "output" dependency – not a problem as long as ordered
- RAR – not a problem



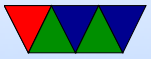
Structural Hazards

- CPU can't just provide. Not enough multipliers for example



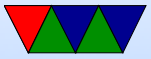
Control Hazards

- How quickly can we know outcome of a branch
- Branch prediction? Branch delay slot?



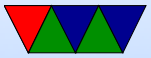
Branch Prediction

- Predict (guess) if a branch is taken or not.
- What do we do if guess wrong? (have to have some way to cancel and start over)
- Modern predictors can be very good, greater than 99%
- Designs are complex and could fill an entire class



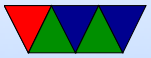
Memory Delay

- Memory/cache is slow
- Need to bubble / Memory Delay Slot



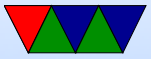
The Memory Wall

- Wulf and McKee
- Processors getting faster more quickly than memory
- Processors can spend large amounts of time waiting for memory to be available
- How do we hide this?



Caches

- Basic idea is that you have small, faster memories that are closer to the CPU and much faster
- Data from main memory is cached in these caches
- Data is automatically brought in as needed.
  Also can be pre-fetched, either explicitly by program or by the hardware guessing.
- What are the downsides of pre-fetching?
- Modern systems often have multiple levels of cache. Usuall a small (32k or so each) L1 instruction and data,



a larger (128k?) shared L2, then L3 and even L4.

- Modern systems also might share caches between processors, more on that later
- Again, could teach a whole class on caches