ECE 574 – Cluster Computing
Lecture 15

Vince Weaver

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

`vincent.weaver@maine.edu`

27 March 2025

Announcements

- HW#7 will be posted
- Don't forget project topics were due

Feedback on HW#5 – Results

- Coarse results
  - In theory, just two sections, so should scale to two
  - Moving to 4 probably won't increase performance
- Fine results
  - In theory should scale decently up to 16 at least
  - The biggest issue if not: you parallelized a loop with only three iterations (either the color loop or a matrix loop).
  - OpenMP can't split up a loop of 3 iterations across 16

cores

- Ways to fix this:
  re-arrange your loops so a big one is outer
  try to use the collapsed loops support
  you can re-write code to merge some of the loops together
- Even so you still might not scale well? Can vary on your code. Tracking down is the job of a performance engineer using tools like PAPI/perf. Can be a challenge.
- Also could be an issue of non-idle system

- Why did I have you print load/store time? Amdahl's law. Reduces overall potential speedup. Why does it vary so much? Not sure
- Static vs Dynamic. Setting up dynamic has a lot of overhead, and since our runs are quick and roughly same size shouldn't make a difference
  Some people did find dynamic was better? Interesting. Again, analyzing why would be interesting.

HW#5 General Code Quality Feedback

- Be sure your code compiles, and that it doesn't crash
- OpenMP is supposed to be about taking existing correct linear code and dropping some pragmas in to make it parallel. You shouldn't have to majorly re-write your code.
  - If you are checking your thread-id and doing things based on it, it's probably not doing things properly
  - Will low-level messing about always work? (If you have 8 threads and you ask for 2, are you guaranteed

they are 0 and 1?)

- If the homework says use sections, use sections. Don't use serial/tasks, or open-code your own sections implementation.
- Don't use nowait unless you know what you are doing
  - Great place for a code comment
  - "Because it crashes otherwise" is not a good reason
- Use of private vars.
  - Loop indices are always treated as private
  - Not sure how some of the solutions worked without declaring sum to be private.

  - Not sure if it's compiler optimizations, or just luck
- Where you put your for (before d 0..2 or x 1..xsize), how it interacts with static vs dynamic
- Putting parallelization in inner loops instead of outer? Complex how this works? Implementation dependent? Probably not recommended but seems to work. Maxes out at total number of threads. Possibly some overhead for starting parallel over and over
- Reduction: you can use a reduction, but only if you are summing up results from a loop. So if you're using a loop to do the sums

- Don't mix pthread and OpenMP code. It might work, but it's not necessary
- Let OpenMP set thread number for you, don't try to parse OMP_NUM_THREADS yourself

HW#7 Preview – Pi clusters

- HW#6 we did MPI, but on a shared memory system (which is inefficient, could have just used OpenMP instead)
- For HW#7 I have you fix up your HW#6 code, then run on an actual cluster
- I don't have any large x86 clusters anymore (big and power hungry) so we'll run on one of my Raspberry Pi clusters which are actual real clusters, but low-power usage

Old Pi2 cluster

- 1 head node (16GB SD card), 24 sub-nodes. One currently seems to be down (reliability!)
- Read up on the cluster here:
  `https://www.mdpi.com/2079-9292/5/4/61/htm`
- 32-bit ARM Cortex A7 with 4-cores each and 1GB RAM
  96 cores and 24GB RAM total
- 15.4 GFLOPS

Old Pi2 Cluster Power Usage

- It's not quite a commodity cluster as it has a fairly complicated power distribution system (ATX power supply to power boards to provide measured 5V to the USB power sockets)
  A bit time consuming to wire up all the cables.
- Power distribution issues
  An ATX power supply runs best when it has a PC-like power draw
  Drawing too much 5V without a 12V load and the 5V

line droops low enough that the Pis won't boot.

- Draws 90W at idle, which is 20W for ethernet switch, a few watts for fan/lights, and rest for the boards

Old Pi2 Cluster Power Downsides

- Only 1GB RAM each node
- Very slow ethernet (behind USB adapter), MPI runs are network limited
- Why such old nodes? Each time I bought new board for it, essentially within a week they'd announce better Pis. 1B to 1B+ to 2 to 3 when I gave up

New Pi4 Cluster

- Five Pi4 nodes, four core 64-bit ARM Cortex A72 (one is 8GB, others 4GB)
- Gigabit ethernet (Pi4s have PCIe and can handle)
- Only 5-nodes so manually updating IP addresses and password files
- Also haven't set up ssh-agent yet
- Set up slurm which can be a pain, especially getting a workable configuration file

New Pi4 Cluster Performance / Power Usage

- 50 GFLOPS
- Single node is 13 GFLOPS or so, so scaling reasonably
- Same performance as 10 year old macbook air
- Power over Ethernet
- 33W idle, 64W linpack, 0.863 GFLOPS/W
- Three times as fast as Pi2 cluster while using less power
- Note: in theory they have powerful GPUs too but as far as I know no one has good GPGPU support for them

yet.

Is there a Pi-5 cluster coming?

- Increasingly power hungry
- Not really a low-power cluster anymore, also takes more cooling

MPI and slurm

- A bit hard to get working, provided script for sbatch should work
- Use “-n” to specify number of cores
- Alternate use “-N” to specify number of nodes maybe in conjunction with - -tasks-per-node
- Not sure how OpenMP and MPI interact here

Why use slurm?

- Can set account to charge
- Can handle checkpointing
- Can set constraints (run on machine with gpu, certain proc type)
- Contiguous allocations
- CPU freq, power capping
- Licenses avail (things like Matlab etc)
- Memory avail

MPI and Linpack

- Running MPI on your own cluster can be a pain, especially making sure it is properly running on all nodes
- Essentially it uses ssh or similar to log into each node and launch your executable. Need to have a copy in the same place on each node (using NFS or similar helps)
- At MPI init it will set things up and communicate between them using sockets. (Details?)
- If running Linpack, you will need to set the P x Q values in HPL.dat to run on more than one nodei

HW#7 More Preview

- First step will be to get HW#6 running if you haven't already. If you had trouble, drop me an e-mail and I can help debug
- Do something to make things more fine grained
    - Parallel combine?

      Note if you do this, you can in theory skip the gather steps after sobelx and sobely and only gather after combine

      Alternately you can use GatherAll() after sobelx/sobely which will essentially broadcast the results back to all ranks after a gather

  - More advanced MPI functions?
  - Scatter at start rather than broadcast?

    Note if you do this, remember that the convolution needs an extra line before and after the rows of interest

- Finally run on Pi cluster

Accelerators

What if CPU power isn't enough?

- We've been mostly looking at ways to get the most performance out of CPUs
- What else is there?

Accelerator Options – ASIC

- hard-coded custom hardware for acceleration
- quite possibly the fastest, as custom made for your workload
- expensive to make, as one-off
- need to hire ASIC designers and get things fabbed
- found in BitCoin mining?

Accelerator Options – FPGA

- Reprogramable logic
- can have fast in-hardware designs but can re-program when workload changes
- Need to have someone who can write FPGA code
- There has been work for having OpenMP and such be able to handle FPGAs
- Tend to be expensive

Accelerator Options – DSP

- Digital Signal Processors
- Can be good at certain workloads
- Some supercomputers have had them

Google Tensor Processing Unit (TPU)

- For accelerating machine learning tasks
  - TPUs good at CNN (convolutional neural networks)
  - GPUs good at fully connected
  - CPUs good at RNN (recurrent)
- ISCA paper – In Datacenter Performance Analysis of a Tensor Processing Unit
- For high-volume low-prevision FP calculations (8 and 16-bit)
- Unlike GPU has no rasterizer or texture processor

Other TPU implementations

- Recently with the advent of AI people have been putting TPUs into everything
- Some recent NVIDIA GPUs have tensor units
- Apple M chips in laptops/cellphones have NPUs
- Intel “AI Boost” in Extreme chips have NPUs
- Intel’s AMX instructions add NPU acceleration

Accelerator Options – Cell Processor (Obsolete)

- Special IBM Power core that had many smaller helper cores
- Could be really fast if programmed well, hard to program
- In end people found it not worth the extra effort
- Was also in PlayStation 3
- Some groups would buy them up and make fast clusters with them. This annoyed Sony who eventually dropped Linux support

Accelerator Options – Xeon Phi (Obsolete)

- Intel, came out of the larabee design (effort to do a GPU powered by x86 chips)
- Large array of x86 chips(p5 class on older models, atom on newer) on PCIe card.
- Sort of like an internal mini cluster
- Runs Linux, can ssh into the boards over PCIe.
- Benefit: can use existing x86 programming tools and knowledge.
- Intel cancelled this

GPUs

- The most common accelerator these days are GPUs (graphics processing units)
- We'll look in detail how these are used

Graphics Background

- Graphics have often been of interest in HPC even outside GP-GPU
- Visualizing the results of experiments is important
- At SC they often have special presentations / awards for best graphics visualizations of HPC results

Graphics and Video Cards / History

Old CRT Days

- Electron gun
- Horizontal Blank, Vertical Blank

LCD Displays (sic)

- Crystals twist in presence of electric field
- Asymmetric on/off times
- Passive (crossed wires) vs Active (transistor each pixel)
- Passive have to be refreshed constantly
- Use only 10% of power of equivalent CRT
- Circuitry inside to scale image and other post-processing
- Need to be refreshed periodically to keep their image
- New “bistable” display under development, requires no power to hold state

Coding for CRTs

- Atari 2600 – only enough RAM to do one scanline at a time
- Apple II – video on alternate cycles, refresh RAM for free
- Bandwidth key issue. SNES / NES, tiles. Double buffering vs only updating during refresh
- Multibanks of graphics (VGA and older) another way to deal with lack of bandwidth

Old 2D Video Cards

- Framebuffer (possibly multi-plane), Palette
- Dual-ported RAM, RAMDAC (Digital-Analog Converter)
- Interface (on PC) various io ports and a 64kB RAM window
- Mode 13h
- Acceleration – often commands for drawing lines, rectangles, blitting sprites, mouse cursors, video overlay

Old 3D Video Cards

- At first only in high-end workstations (like SGI)
- 3dfx cards, with passthrough cable
- Became more mainstream

Modern Graphics Cards

- Essentially high-end linear algebra / 3D rendering supercomputers
- Can draw a lot of power
- 2D (optional afterthought these days)
- Can contain other hardware accelerators (such as Video decoders, TPUs, NPUs)

Interface – Integrated vs Standalone

- Integrated
  - Built into motherboard/chipset/processor
  - Can share memory (and bandwidth) with CPU
  - Traditionally less capable, but that is changing
- Standalone
  - Usually in PCIe slot, bandwidth constrained
  - Can draw lots of power
  - Can have multiple

Video RAM

- VRAM (old) – dual ported. Could read out full 1024Bit line and latch for drawing, previously most would be discarded (cache line read)
- GDDR3/4/5 – traditional one-port RAM. More overhead, but things are fast enough these days it is worth it.
- Confusing naming, GDDR3 is equivalent of DDR2 but with some speed optimization and lower voltage (so higher frequency)

- HBM – high bandwidth memory, stacked on top of GPU or else next to it on interposer. Much higher bandwidth (1024 bit wide vs 64-bit for DDR)

Busses Connecting GPUs to CPUs

- Ancient busses parallel – ISA
- Move to 32 bit big debate (EISA/VLB/MCA)
- Intel's PCI won: PCI and PCI-X were 32/64-bit parallel bus
- Around this time people stopped wanting parallel busses hard to keep so many lines in sync, hard to route on motherboard

PCI-Express PCIe

- PCIe is a serial bus, sends packets
- One lane: differential pairs, one in each direction (so 4 wires)
    - 1x, 2x, 4x, 16x lanes can be grouped together for speed
    - Your CPU only has so many lanes available which can limit how many slots on motherboard
    - Especially as PCIe lanes needed for things like storage, network, etc, not just GPUs

- Versions of spec 1.0 up through 7.0
  - Each one faster and more complex
  - 5.0 can do 32 GT/s
  - NVIDIA Blackwell first PCIe 6.0 device
  - Your CPU possibly still stuck at PCIe 5.0
- In general PCIe is limiting factor to getting data to GPU.

PCIe Power

- Can power 25W
- additional power connectors to supply can have 75W, 150W and more
- Issues where NVIDIA cards melting

Connectors

- DDC – i2c bus connection to monitor, giving screen size, timing info, etc.
- RCA – composite/analog TV
- VGA – 15 pin, analog
- DVI – digital and/or analog. DVI-D, DVD-I, DVD-A
- HDMI – compatible with DVI (though content restrictions). Also audio. HDMI 1.0 – 165MHz, 1080p or 1920x1200 at 60Hz. TMDS differential signaling. Packets. Audio sent during blanking.

- Display Port – similar but not the same as HDMI
- Thunderbolt – combines PCIe and DisplayPort. Intel/Apple. Originally optical, but also Copper. Can send 10W of power.
- LVDS – Low Voltage Differential Signaling – used to connect laptop LCD

Interfaces for 3D Graphics

- OpenGL – SGI (Khronos)
- DirectX – Microsoft (Direct3d)
- Vulkan (sort of next gen OpenGL. Lower level, closer to hardware)
- Metal – from Apple
- WebGL – javascript/web
- OpenGL ES – embedded subset