

ECE 574 – Cluster Computing

Lecture 16

Vince Weaver

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

`vincent.weaver@maine.edu`

21 March 2023

Announcements

- HW#7 was be posted
- Don't forget project topics due Thursday!
- Midterms handed back, average was 90



Pi 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>
- Try not to use up too much disk space
- Also note the SD card is sorta slow, which with the network affects scaling a bit.
- Use slurm
- The batch scripts I gave you have a timeout of 5 minutes



per job. Last time some people's code went crazy and ran forever and other people's jobs never ran

- Use `sinfo` or `squeue` to see cluster and job stats
- Use `scancel` to cancel a job
- If things going poorly, contact me
- Did update PAPI on all nodes which should be working



Raspberry Pi Cluster Notes

- A lot of sysadmin work maintaining a cluster
- OOM. If cause crash let me know
Old Pi2s, so even though cluster has 24GB of RAM,
each node only 1GB (256MB/core)



Raspberry Pi Cluster Construction

- Imaging disks is slow. SD-card takes 40 minutes or so to write a 4GB image.
- Uses DHCP instead of hard-coding IP address in image. Why? Allows one common disk image for all nodes.
- NFS filesystem: for MPI to work you need to have an identical file layout (including the executable) on all nodes. Using a cluster filesystem makes this easier.
- ganglia: provides cluster stats via web-browser. Having a huge issue trying to get it working.



Pi 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



MPI and slurm

- HW #SBATCH `--tasks-per-node=4`
- `-N` = number of nodes
- `-n` = number of tasks, default is one task per node?
- `N=4 tasks-per-node=4, 16`
`N=4 tasks-per-node=4, sbatch -n 8, 16 (N=nodes, n=tasks)`
`N=4 tasks-per-node=4, sbatch -N 8, 32`



nothing, sbatch -N 8, 32

nothing, sbatch -n 8, (8, 2 nodes * 4 each)

nothing, sbatch -N 8 -n 8 (8, 8 nodes * 1 each)



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



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

- Reprogrammable 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



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-precision FP calculations (8 and 16-bit)
- Unlike GPU has no rasterizer or texture processor



- Some recent NVIDIA GPUs have tensor units



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

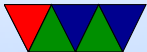


Accelerator Options – Xeon Phi (Obsolete)

- 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



Graphics and Video Cards / History



Old CRT Days

- Electron gun
- Horizontal Blank, Vertical Blank

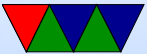


LCD Displays (sic)

- Crystals twist in presence of electric field
- Asymmetric on/off times
- Passive (crossing wires) vs Active (Transistor at 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)



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)



Busses

- DDC – i2c bus connection to monitor, giving screen size, timing info, etc.
- PCIe (PCI-Express) – most common bus in x86 systems
Original PCI and PCI-X was 32/64-bit parallel bus
PCIe is a serial bus, sends packets
Can power 25W, additional power connectors to supply
can have 75W, 150W and more
Can transfer 8GT/s (giga-transfers) a second
In general PCIe is limiting factor to getting data to GPU.



Connectors

CRTC (CRT Controller) Can point to same part of memory (mirror) or different.

- 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 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



GPUs

- Display memory often broken up into tiles (improves cache locality)
- Massively parallel matrix-processing CPUs that write to the frame buffer (or can be used for calculation)
- Texture control, 3d state, vectors
- Front-buffer (written out), Back Buffer (being rendered)
Z-buffer (depth)
- Originally just did lighting and triangle calculations. Now shader languages and fully generic processing



GPGPUs

- Interfaces needed, as GPU companies do not like to reveal what their chips do at the assembly level.
 - CUDA (Nvidia)
 - OpenCL (Everyone else) – can in theory take parallel code and map to CPU, GPU, FPGA, DSP, etc
 - OpenACC?



Why GPUs?

- Newer example:
 - Cascade Lake, 1 TFLOP (64-bit floating point)
 - NVIDIA 3090 36 TFLOPs
- Older example
 - Raspberry Pi, 700MHz, 0.177 GFLOPS
 - On-board GPU: Video Core IV: 24 GFLOPS



Key Idea

- using many slimmed down cores
- have single instruction stream operate across many cores (SIMD)
- avoid latency (slow textures, etc) by working on another group when one stalls



Latency vs Throughput

- CPUs = Low latency, low throughput
- GPUs = high latency, high throughput
- CPUs optimized to try to get lowest latency (caches); with no parallelism have to get memory back as soon as possible
- GPUs optimized for throughput. Best throughput for all better than low-latency for one



GPU Benefits

- Specialized hardware, concentrating on arithmetic. Transistors for ALUs not cache.
- Fast 32-bit floating point (16-bit?)
- Driven by commodity gaming, so much faster than would be if only HPC people using them.
- Accuracy? 64-bit floating point? 32-bit floating point? 16-bit floating point? Doesn't matter as much if color slightly off for a frame in your video game.
- highly parallel



GPU Challenges

- optimized for 3d-graphics, not always ideal for other things
- Need to port code, usually can't just recompile cpu code.
- Companies secretive.
- serial code
- a lot of control flow
- lot of off-chip memory transfers



Older / Traditional GPU Pipeline

- In old days, fixed pipeline (lots of triangles).
- Modern chips much more flexible, but the old pipeline can still be implemented in software via the fancier interface.



3D Graphics Rundown

- Rasterization (traditional 3d cards)
 - Send vertices to card
 - Triangles, normals
 - Project to 2d screen
 - Broken up to pixels and shaded/textured
 - Clipping, depth
- Ray-tracing
 - Light is traced to eye (or the reverse)
- Ray-casting



Older / Traditional GPU Pipeline

- In old days, fixed pipeline (lots of triangles).
- Modern chips much more flexible, but the old pipeline can still be implemented in software via the fancier interface.



Older / Traditional GPU Pipeline

- CPU send list of vertices to GPU.
- Transform (vertex processor) (convert from world space to image space). 3d translation to 2d, calculate lighting. Operate on 4-wide vectors (x,y,z,w in projected space, r,g,b,a color space)
- Rasterizer – transform vertexes/vectors into a grid. Fragments. break up to pixels and anti-alias
- Shader (Fragment processor) compute color for each pixel. Use textures if necessary (texture memory, mostly



read)

- Write out to framebuffer (mostly write)
- Z-buffer for depth/visibility



GPGPUs

- Started when the vertex and fragment processors became generically programmable (originally to allow more advanced shading and lighting calculations)
- By having generic use can adapt to different workloads, some having more vertex operations and some more fragment



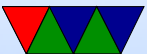
Graphics vs Programmable Use

Vertex	Vertex Processing	Data	MIMD processing
Polygon	Polygon Setup	Lists	SIMD Rasterization
Fragment	Per-pixel math	Data	Programmable SIMD
Texture	Data fetch, Blending	Data	Data Fetch
Image	Z-buffer, anti-alias	Data	Predicated Write



Shader Programming

- There are competitions. Also see shadertoy.com
- Vertex Shader
 - Vertex transform
 - Object space to clip space
 - Compute colors, normals, texture co-ords
 - Can displace/distort (move vertices: wave flag)
 - Can animate (move vertices: move fish)
- Fragment Shader
 - Compute and color



- Get data from vorteces and textures
- Can make better materials. Glossy, reflections, bumpy, shadows



GLSL Shader Programming

- Similar to C code
- Based on OpenGL
- vertex
 - Each time screen drawn main() called once per vertex
 - Massively parallel
 - Have vars. Can get positions
- Fragment
 - Each time screen drawn main() called once per pixel
 - Can get x/y

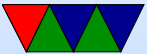


Example Shader 3.0 (DX9) Capabilities – Vertex Processor

- They are up to Pixel Shader 5.0 now
- 512 static / 65536 dynamic instructions
- Up to 32 temporary registers
- Simple flow control
- Texturing – texture data can be fetched during vertex operations
- Can do a four-wide SIMD MAD (multiply ADD) and a scalar op per cycle:



- EXP, EXPP, LIT, LOGP (exponential)
- RCP, RSQ (reciprocal, r-square-root)
- SIN, COS (trig)



Example Shader 3.0 (DX9) capabilities– Fragment Processor

- 65536 static / 65536 dynamic instructions (but can time out if takes too long)
- Supports conditional branches and loops
- fp32 and fp16 internal precision
- Can do 4-wide MAD and 4-wide DP4 (dot product)

