ECE 574 – Cluster Computing
Lecture 15

Vince Weaver

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

vincent.weaver@maine.edu

27 October 2015

Announcements

- HW#5 review
- HW#6 Will be posted

HW#5 Review

- Interesting corner cases
  Not having proper private/shared constraints could cause a slowdown (cache-bouncing) but not correctness issues?

- Other performance issues when not having curly brackets around parallel section?

- Many corner cases here.

Notes on MPI for HW#6

Raspberry Pi Cluster Construction

- Imaging disks is slow. SD-card takes 40 minutes or so to write a 4GB image.

- 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 low and the 5V line droops low enough that the Pis won't boot.

- 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.

GPGPU

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

Interfaces

- OpenGL – SGI
- DirectX – Microsoft
- For consumer grade, driven by gaming

GPGPUS

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

Why GPUs?

- Old example:
  - 3GHz Pentium 4, 6 GFLOPS, 6GB/sec peak
  - GeForceFX 6800: 53GFLOPS, 34GB/sec peak
- Newer 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

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 Problems

- 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