ECE 574 – Cluster Computing
Lecture 17

Vince Weaver

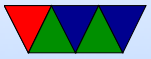
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

23 March 2023

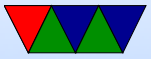
Announcements

- HW#8 (CUDA) will be posted.
- Project topics were due.
- News: NVIDIA GTC, new H100 NVL GPU for large language models. Two H100 PCIe cards stuck together, spans 4 slots, 700W. HBM3 memory. 94GB/die.

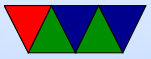


Raspberry Pi Cluster Notes

- 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)
- Issue with scp -r, a lot of this is because I have old pis need to update, but time consuming and things will break
  also probably need larger SD cards
- Problem is “scp” is deprecated for reasons and so defaults

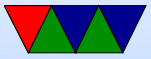


to the “sftp” protocol instead, but older versions of sftp don’t understand the concept of home directories. Fix is to use `scp -O -r` where `-O means` use old scp instead



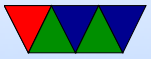
HW#7 Notes

- Extended until Monday
- Try not to break cluster over the weekend
- Provided a solution. Don't usually like doing that
- Working on grading HW#6, debugging other people's code is extremely time consuming
- Debugging skills. How would you debug a program like this?
  - Write in small chunks, testing along way. Easier than throwing together big mass of code and then giving up

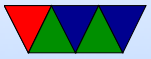


when it doesn't work

- That's why I tried to have you do things like
- Test on 1-rank and be sure that works before moving onto more ranks
- Dump intermediate output, be sure sobelx works before worrying about sobely or combine/
- Print your ranges and make sure they make sense
- "The output looks the same", but it isn't. Try flipping between them. There might be binary diff tools to actually show you what's different, though that's more difficult if it's an off-by-one error.

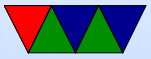


- If it crashes, usually it means you're going off the edge of a buffer, double and triple check the values that are going into array accesses (or even worse, pointers)
- People annoyed the tests I give don't work, but they're unit tests, known good inputs/outputs for verifying code works
- Some code is tricky, like finding edge conditions on inputs. This is an important thing that happens often in programming. Coding isn't always cut+paste of ask an AI, someone has to write the original tricky code.



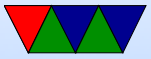
CUDA – installing

- On Linux need to install the proprietary NVIDIA drivers
- Have to specify nonfree on Debian.
- Debates over the years whether NVIDIA can have proprietary drivers; no one sued yet. (Depends on whether they are a "derived work" or not. Linus refuses to weigh in)
- Sometimes have issues where drivers won't install (currently having that issue on some of my machines)



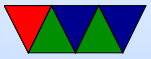
Question: how does Hardware Raytrace work

- NVIDIA: Optix Library
- You describe how rays behave
- Details are a bit hard to get

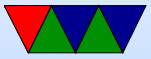


NVIDIA GPUs

- Quadro (Workstation) vs Geforce (Gaming) (note from 2023, they renamed these)
  - Quadro generally more RAM. higher Bus width
  - Fancier Drivers
  - Optimized for CAD type stuff and compute, rather than games
  - Higher reliability
  - Quadro better support for double-precision floats
  - More compute cores

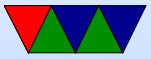


- Power limits



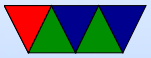
NVIDIA Generations

- Kepler
- Maxwell
- Pascal
- Turing (consumer)/Volta (pro)
- Ampere
- Lovelace/Hopper

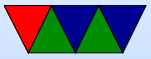


GPU hardware in my Lab

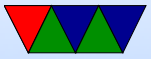
- Can use these for projects. I mostly get these for power measurement tests.
- NVIDIA RTX A2000 in Skylake
  - 6GB GDDR6, 192-bit, 288 GB/s
  - Ampere, PCIe4x16
  - 3328 CUDA cores, 104 tensor cores, 26 RT cores
  - 8 TFLOPS single-precision, RT 15.6 TFLOPS, Tensor 64 TFLOPS
  - 70W, DirectX 12.07, Vulkan 1.2



- NVIDIA Quadro P2000 in Skylake (old)
  - 5GB GDDR5, 160-bit, 140 GB/s
  - 1024 cores, Pascal, PCIe3x16
  - 75W, DirectX 12.0, Vulkan 1.0
- NVIDIA Quadro P400 in Haswell-EP
  - 2GB GDDR5, 64-bit, up to 32 GB/s
  - 256 cores, Pascal architecture
  - 30W, OpenGL 4.5, DirectX 12.0
  - Low-power for server, as runs in 1U rack
- NVIDIA Quadro K2200 in Quadro
  - So old the drivers don't want to support it anymore

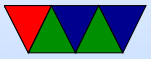


- 4GB GDDR5, 128-bit 80 GB/s
- 640 cores, Maxwell architecture
- 68W, OpenGL 4.5, DirectX 11.2



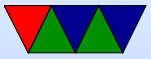
Programming a GPGPU (CUDA/OpenCL)

- Create a "kernel" which is a small GPU program that runs on a single thread. This will be run on many cores at a time.
- Allocate memory on the GPU and copy input data to it
- Launch the kernel to run many times in parallel. The threads operate in lockstep, all executing the same instruction in each thread.
- How is conditional execution handled? a lot like on ARM. If/then/else. If the particular thread does not



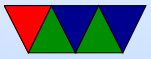
meet the condition, it just does nothing until the other condition finishes executing.

- If more threads are needed the available on the GPU, may need to break the problem up into smaller batches of threads.
- Once computing is done, copy results back to the CPU.



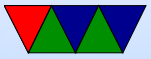
CPU vs GPU Programming Difference

- The biggest difference: NO LOOPS
- You essentially collapse your loop, and run all the loop iterations simultaneously.



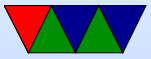
Flow Control, Branches

- Only recently added to GPUs, but at a performance penalty.
- Often a lot like ARM conditional execution



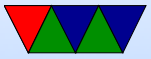
NVIDIA Terminology (CUDA)

- Thread: chunk of code running on GPU.
- Warp: group of thread running at same time in parallel simultaneously
  AMD calls this a "wavefront"
- Block: group of threads that need to run
- Grid: a group of thread blocks that need to finish before next can be started



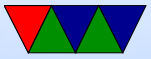
Terminology (cores)

- Confusing. Nvidia would say GTX285 had 240 stream processors; what they mean is 30 cores, 8 SIMD units per core.



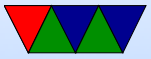
CUDA Programming

- Since 2006
- Compute Unified Device Architecture
- See the NVIDIA "CUDA C Programming Guide"
- Use `nvcc` to compile
- .cu files. Note, technically C++ so watch for things like `new`



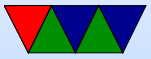
CUDA Coding

- version compliance – can check version number. New versions support more hardware but sometimes drop old
- nvcc – wrapper around gcc. global code compiled into PTX (parallel thread execution) ISA
- can code in PTX code directly which is sort of like assembly language. Won't give out *actual* assembly language. Why?
- CUDA C has mix of host and device code. Compiles the global stuff to PTX, compiles the <<< ... >>> into



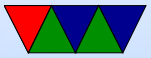
code that can launch the GPU code

- PTX code is JIT compiled into native by the device driver
- You can control JIT with environment variables
- Only subset of C/C++ supported in the device code



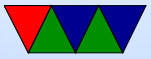
CUDA Programming

- Heterogeneous programming – there is a host executing a main body of code (a CPU) and it dispatches code to run on a device (a GPU)
- CUDA assumes host and device each have own separate DRAM memory
  (newer cards can share address space via VM tricks)
- CUDA C extends C, define C functions "kernels" that are executed N times in parallel by N CUDA threads



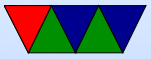
CUDA Programming – Host vs Device

- *host* vs *device*
  host code runs on CPU
  device code runs on GPU
- Host code compiled by host compiler (gcc), device code by custom NVidia compiler



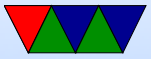
CUDA Programming – Memory Allocation

- cudaMalloc() to allocate memory and pointers that can be passed in
  `cudaMalloc((void **)&dev_a,N*sizeof(int));`
- `cudaFree()` at the end
- `cudaMemcpy(dev_a,a,N*sizeof(int),`
  `cudaMemcpyHostToDevice);`
- `cudaMemcpy(c,dev_c,N*sizeof(int),`
  `cudaMemcpyDeviceToHost);`



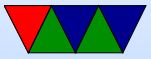
CUDA Programming – Pointers

- Note: result of a cudaMalloc() might look like a pointer, but it's not
- You can't dereference memory allocated with cudaMalloc() on the CPU, the memory area is completely separate
- There is work on newer GPUs allowing unified CPU/GPU memory but we're going to assume that's not available



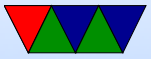
CUDA Hardware

- GPU is array of Streaming Multiprocessors (SMs)
- Program partitioned into blocks of threads. Blocks execute independently from each other.
- Manages/Schedules/Executes threads in groups of 32 parallel threads (warps) (weaving terminology) (no relation)
- Threads have own PC, registers, etc, and can execute independently
- When SM given thread block, partitions to warps and



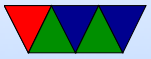
each warp gets scheduled

- One common instruction at a time. If diverge in control flow, each way executed and thread not taking that path just waits.
- Full context stored with each warp; if warp is not ready (waiting for memory) then it may be stopped and another warp that's ready can be run



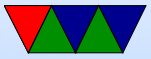
CUDA Threads

- kernel defined using `__global__` declaration. When called use <<<...>>> to specify number of threads
- each thread that is called is assigned a unique ThreadID Use threadIdx to find what thread you are and act accordingly

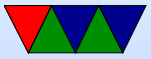


CUDA Programming – Kernels

- `__global__` parameters to function – means pass to CUDA compiler
- call global function like this `add<<<1,1>>>(args)` where first inside brackets is number of blocks, second is threads per block
- Can get block number with blockIdx.x and thread index with threadIdx.x
- Can have 65536 blocks and 512 threads (At least in 2010)

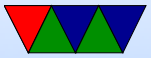


- Why threads vs blocks?
  Shared memory, block specific
  `__shared__` to specify
- `__syncthreads()` is a barrier to make sure all threads finish before continuing



CUDA Debugging

- Can download special cuda-gdb from NVIDIA
- Plain printf debugging doesn't really work



CUDA Example

```
__global__ void VecAdd(float *A, float *B, float *C) {
    int i = threadIdx.x;

    if (i<N)          // don't execute out of bounds
        C[i]=A[i]+B[i];
}

int main(int argc, char **argv) {
    ....
    /* Invoke N threads */
    VecAdd<<<1,N>>>(A,B,C);
}
```