Lecture Notes

Stephanie Chan

November 25, 2013

1 GPUs

1.1 Intro

Today, we'll talk about low level CUDA. Next time, we'll be looking at PyCuda and RCuda. GPUs are on Macs and Windows. You can use the one on your desktop or on the school servers: **lipschitz** and **pearson**

1.2 Background and History

GPU stands for graphical processing unit, used for billions of simple calculations. NVIDIA's CUDA programming language enables people to program GPUS. The other main manufacturer of GPUs is AMD, whose GPUs can be programmed using OpenCL (a language backed by Apple and other tech companies). At this stage, however, CUDA is preferred in the academic community.

CUDA is a C/C++ library, it is low-level (must do memory management and synchronity) but also has high level interfaces

2 Parallelism

2.1 Type 1. Task Parallelism

- Which tasks depend on previous tasks to do, and which can be done in parallel.
- Example: computing Multivariate Normal Density: after getting Cholesky decomposition, computing the inverse and the determinant can be done in parallel.
- Task parallelism is better for CPUs.

2.2 Type 2: Data Parallelism

- This is the same task with multiple pieces of data.
- Examples: matrix multiplication, row sum, column sum, numerical integration.
- Key role is **kernel** (the function that is applied to all data). Determine the appropriate kernel to do the task.

3 CPU vs GPU

- **CPU**: big cache, 4 ALUs
- **GPU**: little caches, lots of ALUs, little memory, so must use tons of stuff in parallel to be efficient

4 Definitions

- **kernel**: GPU program on thread grid
- **Thread hierarchy**: **Grid** -> **Block** -> **Warp** -> **Threads**
- **SIMD**: single instruction multiple data
- Grids and Blocks have three dimensions, but we do not need to use all three dimensions
- **Host**: CPU
- **Device** : GPU
- **Kernel**: function that runs on GPU
- **Thread**: series of calculations. Threads are cheap on GPU, so gains are made from launching large numbers of threads.

5 Basics

1. Allocate data in GPU
2. Transfer data
3. Launch Kernel
4. Transfer Results

Note: pinned memory can access data from CPU which saves on transfer. GPU must also not waste the transferring data time in order to be efficient.

6 Lecture Code Examples

6.1 Example0

- specify Grid and Block dimensions and threads $20 \times 2 \times 512$
- Block dimensions are tyically in multiples of 2 because of warp sizes.
- Get more threads than data 20480 threads for 20000 helloworlds
- **main**: check the number of threads is large enough and launch kernel

- **kernel**: must be global, here it has no argument or output. Global to know block, thread indices. Note: CUDA uses 0-indexing
- Make sure to check $n < N$ or else it will try to access memory it shouldn't.
- Use `nvcc` to compile code. `-arch=compute_20` for printing, it compiles to version 20
- The GPU prints to a buffer, so when it flushes, it will only flush what is in the buffer. Use `cudaDeviceGetLimit` and `cudaDeviceSetLimit` to change the buffer size.

6.2 Example1

- Each thread does multiple cosines, to keep all the threads busy Functions to use, cosf or __cos
- Check for CUDA device and what device
- What happens in main:
  1. Use malloc, allocate memory on CPU, fill in the memory on the CPU
  2. Allocate memory on the device. Make sure everything is properly allocated
  3. `cudaMemcopy` can copy both directions.
  4. lauch kernal.
  5. Free the memory when done.
- RCUDA and PyCUDA hide the memory allocation from you.

6.3 Example2

- In R, almost anytime you do linear algebra, it will call the BLAS or LAPACK linear algebra libraries to ensure it does the matrix algebra really fast. In GPUs, you would use `cublas_v2` to do basic linear algebra for the GPU.
- CUBLAS has `Handle` and `Status`.
- First, it sets up the device, and then it checks cublas is set up correctly.
- `cublasSetVector`, `cublasGetVector`
- Launch the same kernel as before, and then getting the sum of the absolute values.
- shut down cublas with cublasDestroy
- For production matrix algebra, use libraries and/or other expert GPU code, it will be faster than anything you can write.