Practical Introduction to CUDA and GPU

Charlie Tang

Centre for Theoretical Neuroscience

October 9, 2009

Overview

- CUDA - stands for Compute Unified Device Architecture
- Introduced Nov. 2006, a parallel computing architecture to use NVIDIA GPU
- New parallel programming model and instructure architecture
- Support for C; FORTRAN, C++, OpenCL, DirectX Compute to come
- Low learning curve for C programmers
- See pg. 12 Programming Guide

CUDA

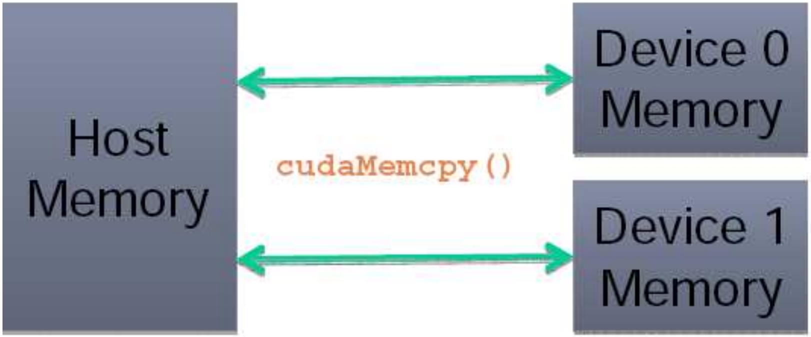
- "All the cool kids are using it!" Cuda Zone
- Support base: Cuda Forum
- OpenCV to OpenVIDIA Computer Vision - it's like saturday morning before all the football games starts (if you like football)

Performance Comparison

- See pg. 10 Programming Guide

CUDA Programming

- Three key abstractions: hierarchy of thread groups; shared memories; barrier synchronization
- nvcc (CUDA compiler) compiles a .cu file (C program with extended syntax) into host (CPU) code and device (GPU) code



CUDA Programming

- .cu file contains host code and device code
- Host code is run by the CPU, device code is run by the GPU
- Show pg. 19-22 gpu slide CS775
- Show pg. 15 Programming Guide
- Grid of thread blocks - show pg. 18 Programming Guide
- Mapping to hardware - show pg. 80 Programming Guide
- deviceQuery program

Execution

- Scalable array of multithreaded Streaming Multiprocessors (SMs)
- Host CPU calls a kernel, launches the kernel grid, with 2D layout of thread blocks
- Each thread block is allocated to 1 SM. (GTX 280 has 30 SMs)
- Each SM contains 8 Scalar Processor (SP) cores, 2 special math units, multithreaded instruction unit, on-chip shared memory
- Allows for fine-grain parallelism, e.g. each thread process one element of a matrix of a pixel in an image
- Each SM supports **__syncthreads()** function for barrier synchronization

Warps

- Each SM uses a Single Instruction Multiple Threads (SIMT) architecture
- Maps every thread onto a SP core
- Manages, schedules, and carry out thread executions in warps of 32 threads
- If threads within a warp diverge via data dependent branching, then the warp serially executes each branch path in turn, disabling threads on all other paths
- Therefore, max efficiency occurs when all threads in a warp agrees on execution path

Memory of SM

- (really fast) 32 bit registers
- (fast) shared memory cache - shared by all SP cores and therefore all threads in a block
- (fast, read-only) constant memory cache
- (fast, read-only) texture memory cache
- (slow, read-write) global memory - not cached
- (slow, read-write) local memory - not cached (used when variables are too big in kernel)
- See pg. 83 Programming Guide, pg 23. gpu slides

Specifics - C extensions

Function Type Qualifiers

- __**device**__: execute on device, callable from device only
- __**global**__: a kernel func, execute on device, callable from host only
- __**host**__: execute on host, callable from host only (default if a func has no qualifiers)

Builtin Variables

- **gridDim**: e.g. gridDim.x gridDim.y
- **blockIdx**: e.g. blockIdx.x blockIdx.y
- **blockDim**: e.g. blockDim.x blockDim.y blockDim.z
- **threadDim**: e.g.threadDim.x threadDim.y threadDim.z
- all are dim3, an integer vector type

Compiling

- 2 interfaces: C for CUDA (easier), and CUDA driver API (harder)
- **nvcc** compiles C or PTX language code (within .cu file) into binary code
- Splits .cu code into host and device code, host code is compiled by standard compilers (gcc), device code is compiled into PTX code or *cubin* objects
- *cubin* objects can be executed by CUDA driver API directly or linked together with host executable and lauched as kernels
- Full C++ support for host code, subset of C for device code, warrants splitting up the device and host code into separate .cu files
- -arch sm_13 flag for double precision, only on GPU with compute capability of 1.3 or later. Slow!! Fermi 8x faster

First Example!

- Show code, common.mk, and Makefile
- **cudaMalloc()**
- **cudaMemcpy()**
- **cudaFree()**

Shared Memory Example

- Show pg. 28 Programming Guide

Useful APIs

- CUBLAS - Basic Linear Algebra Subroutines (BLAS) implementation e.g. matrix-vector, matrix-matrix operations
- CULA - LAPACK for CUDA e.g. LU, QR, SVD, Least Squares
- CUFFT - 14x faster than Fastest Fourier Transform in the West (FFTW)
- CUDPP - CUDA Data Parallel Primitives Library e.g. parallel-prefix-sum, parallel sorting, sparse matrix vector multiply, parallel reduction

Example - Matrix Multiplication

Use CUBLAS - Basic Linear Algebra Subroutine implementation in C for CUDA

- Show matlab and .cu code
- Compare to Matlab
- Run example

CUDA debugging using CUDA-GDB

- nvcc -g -G first_example.cu -o first_example
- Newly introduced hardware debugger based on GDB!!

CUDA profiling

- Find out which kernel calls is dragging you behind
- Demo

Example - Sum

How do we perform MATLAB's sum() in parallel?

- This is a parallel reduction problem
- Show instructive white paper

Example - Wake Sleep algorithm for DBN

10x speed up over the fine tuning algorithm for Deep Belief Network

- Show matlab and .cu code
- Run example

Example - Backpropagation

Nonlinear Neighborhood Component Analysis learning algorithm

- Show matlab and .cu code
- Run example

Thank You

- Supercomputer in your desktop!
- No reason why you shouldn't attack algorithm bottelnecks with CUDA