Getting Started with CUDA Programming

ECE 6397
Electrical and Computer Engineering
University of Houston
Dr. David Mayerich
Coding

• CUDA uses C/C++ syntax with some additional keywords reserved for GPU instructions
• Use standard C/C++ development tools, combined with the nvcc (nVidia compiler)
• Recommended development – standard to industry, academia
  • store your code on a shared resource
    • allows you to update the code from multiple locations (lab, home)
    • this resource should be backed up: Dropbox, BitTorrent Sync, NAS
    • consider using a versioning tool: Git, Subversion
      • git tutorial: https://www.atlassian.com/git/tutorials/
  • create a local build environment: CMake (https://cmake.org/)
  • compile/link/test your code
  • update source code as necessary
Development Tools

• Linux systems
  • Accessing Linux systems remotely
  • Generating and building a basic C/C++ application

• Windows systems
  • Creating a Visual Studio project
  • Compiling and running code
  • Debugging

• Cross-platform development
Linux Systems

• The cluster provided for this course is a Linux system
  • IP address is available through Blackboard

• In many ways, Linux is designed for development
  • libraries are easy to install and find
  • code is generally easy to compile
  • development tools are optimized for working remotely

• DEMO:
  • MobaXTerm (http://mobaxterm.mobatek.net/)
  • SSH basics
  • “Hello world!”, gcc (g++)
  • CUDA example
Windows Systems

• Install Visual Studio: [https://visualstudio.microsoft.com/](https://visualstudio.microsoft.com/)
  • Make sure to install for C++ development!


• Install CMake: [https://cmake.org/](https://cmake.org/)

• CUDA only supports 64-bit (Visual Studio default is Win32)
Creating a Build Environment

• CMake ([cmake.org](http://cmake.org)) – Multi-platform build tool
  • CMake will create a build environment using the OS and build tools specified by the user
    • Windows: Visual Studio Solution (ARM 64 – CUDA requires 64-bit code!!!!)
    • Linux: Makefile (Code::Blocks or Eclipse project)
  • Parameters are specified in a CMakeLists.txt file

```
cmake_minimum_required(VERSION 2.8) #Specify the minimum CMake version
project(hello) #Specify the name of the project
find_package(CUDA REQUIRED) #find the CUDA libraries
include_directories(${CUDA_INCLUDE_DIRS}) #Specify the CUDA include directories for the project
cuda_add_executable(hello main.cu) #create an executable from the given CUDA files

#specify any additional libraries here (CUFFT and CUBLAS can be useful)
target_link_libraries(hello
  ${CUDA_cufft_LIBRARY}
  ${CUDA_cublas_LIBRARY})
```
Running CMake

• **Source Code Directory** – location of all source files for the specified project
  • Often this is the location of a source *repository* (Git, SVN, etc.)
  • As a general rule: Keep this directory free of anything that changes during compiling/linking (building) or execution!

• **Build Directory** – location of the build environment
  • This will be where the executable (and all intermediate files) are created
  • Any data files that are required can be copied here using CMake

• **Keep your source code and build directory separate!**
  • If anything goes wrong with libraries, you can always completely delete your build directory and re-create it without any problems
  • Easily back up your source code by archiving all files in your source directory
Notes for Laptops

• Most laptops with nVidia GPUs use software called Optimus to switch between on-board and nVidia cards as required
  • (on Linux the consumer-driven software is called Bumblebee)
  • **If your code isn’t detecting a viable GPU – this is probably why**

• Your program should be set up to use the GPU
  • Download the newest nVidia drivers for your card
  • Open the nVidia Control Panel:
    
    Program Files/NVIDIA Corporation/Control Panel Client/nvcplui.exe

  • Manage 3D Settings -> Program Settings
  • Browse to your executable (*.exe file)
  • Under (2. Select the preferred graphics processor....) select your nVidia card
  • Apply settings
  • **You can also set up the GPU for all applications (probably not a good idea)**
Pointers

• Allocate a block of memory for an array of N floating point values:

```c
float* x = (float*) malloc(N * sizeof(float));
```

• Allocate a block of memory and split it into two 32-bit floating point arrays of size M and K

```c
size_t N = (M + K) * sizeof(float);
char* block = (char*) malloc(N);
float* A = (float*) block;
float* B = (float*) (block + M * sizeof(float));
```
Pointers

• The following statement will often give you an error or warning (depending on your compiler). Why?

```c
size_t N = 1024;
float* y = malloc(N * sizeof(float));
```

• malloc() returns a pointer of type `void*`
• it is good practice to pay close attention to your pointer types
Pointers

• What is wrong with the following code?

```c
size_t N = M * K;
float* block = (float*) malloc(N * sizeof(float));
float* A = block;
float* B = block + M * sizeof(float);
```

• pointer arithmetic is based on the pointer type
• if x and y are pointers of type T* and M is an integer:

```
y = x + M; will increment x by M elements of type T
```
Arguments

• Arguments are passed to your application by the operating system:
  `>> myprogram hello.txt 123 output.txt`

• These arguments can be accessed through the main() function:
```cpp
#include <string>
int main(int argc, char* argv[]){
    std::string command = argv[0];
    std::string infile = argv[1];
    int i = atoi(argv[2]);
    std::string outfile = argv[3];
}
```
Working with Files

• Load data from a binary file

```cpp
#include <fstream>
...
std::ifstream infile("filename.mtx", std::ios::binary);
int M;
infile.read(&M, sizeof(int));
float* x = (float*) malloc(M * sizeof(float));
infile.read(x, M * sizeof(float));
```
Homework 0

• Set up, compile, and run a CUDA program
• CMakeLists.txt and main.cu files are available online
• Use CMake to generate a build environment
• Compile and execute your code

  • Visual Studio: Open the VS Solution (*.sln) file and click

  • Linux: Go to your build directory, type “make” [ENTER], “./hello” [ENTER]

• The program I’ve provided will list the capabilities of the first CUDA-capable device on your system
  1) Change the code to output this data to a file
  2) Upload this file to Blackboard under Homework 0