GPGPU Programming

Donato D'Ambrosio

Department of Mathematics and Computer Science
Cubo 30B, University of Calabria, Rende 87036, Italy
mailto: donato.dambrosio@unical.it
homepage: `http://www.mat.unical.it/~donato`

Academic Year 2020/21

Table of contents

Introduction

Introduction

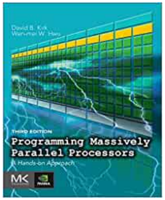
About the Course

The course illustrates the fundamentals concepts and algorithms of **GPGPU programming** by means of **CUDA.**

If possible, a brief overview of OpenCL will be also given, which represents a portable, non-proprietary alternative to CUDA for programming massively parallel processors.

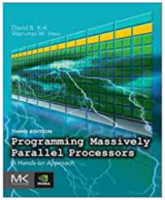
Books and other resources

- David B. Kirk, Wen-mei W. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach* (Third Edition). Morgan Kaufmann, 2017
- These slides (mainly based on the book above)
- CUDA Toolkit Documentation: `https://docs.nvidia.com/cuda/index.html`
- CUDA Unified Memory: `https://developer.nvidia.com/blog/unified-memory-cuda-beginners/`
- CUDA GPU Compute Capability: `https://developer.nvidia.com/cuda-gpus`
- CUDA Occupancy Calculator: `https://docs.nvidia.com/cuda/cuda-occupancy-calculator/index.html`



Syllabus

Chapters 1, 2, 3 of the textbook

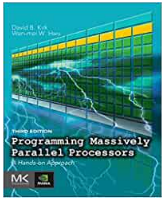


- Introduction and Development of GPU Computing
  - Architecture of GPUs
  - CPU-GPU Architectural Comparison
  - GPGPU and Heterogeneous Computing
- Data Parallelism and CUDA/OpenCL
  - Overview of CUDA (and OpenCL)
  - Programming model
  - A first example of application: Vector Addition
- Data-Parallel Execution Model
  - Thread Organization and Mapping to Data
  - A More Complex Example: Matrix Multiplication
  - Thread Synchronization
  - Thread Scheduling and Latency Tolerance

Syllabus

Chapters 4, 5, 7, 8 of the textbook

- Memory
  - GPGPU Memory Model
  - Reducing Global Memory Traffic
  - A Tiled Matrix Multiplication Kernel
- Improving Performance
  - Warps and Thread Execution
  - Global Memory Bandwidth
  - Using Shared/Local Memory and Registers/Private Memory
- Example of Application
  - Stencil
  - Convolution
  - Prefix Sum



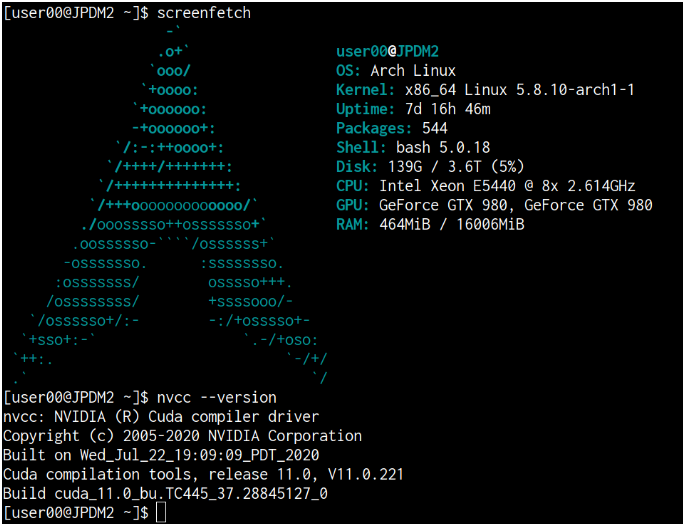
Exam

- The assessment consists of two parts
  - A **written test** (at most 1.5 hours), in which students must demonstrate knowledge of the basic concepts of the course.
  - A **project** to verify their design and development skill.
- Assessment criteria of learning
  - The written test aims at verifying the basic knowledge of the course and allows to access to the next phase regarding the evaluation of the project.
  - The project aims to verify the ability to design and develop massive parallel software on GPUs.
- Criteria for measuring learning and assigning the final mark
  - The **partial score** assigned to each test is between 0 and 30.
  - The **overall score** is obtained as the average of the two partial outcomes.
  - The ***laude*** is given in case both tests are considered particularly deserving.

CUDA Workstation @ DeMaCS - UniCAL

- Dual quad-core Intel Xeon CPU E5440 @ 2.83GHz, 16GB of RAM
- Dual **GeForce GTX 980** NVIDIA GPU
  - Compute Capability: 5.2
  - Architecture: Maxwell
  - Cores: 2048 (16 SMs, 128 cores per SM) @ 1126-1216 MHz
  - Memory: 4 GB GDDR5 @ 1750MHz
  - Bandwidth: 224 GB/s
  - FP32 (float) performance: 4.981 TFLOPS
  - FP64 (double) performance: 155.6 GFLOPS (1:32)
  - OpenCL support: 1.2

The JPDM2 Workstation



```
[user00@JPDM2 ~]$ screenfetch
                   -`
                  .o+`                 user00@JPDM2
                 `ooo/                 OS: Arch Linux
                `+oooo:                Kernel: x86_64 Linux 5.8.10-arch1-1
               `+oooooo:               Uptime: 7d 16h 46m
               -+oooooo+:              Packages: 544
             `/:-:++oooo+:             Shell: bash 5.0.18
            `/++++/+++++++:            Disk: 139G / 3.6T (5%)
           `/++++++++++++++:           CPU: Intel Xeon E5440 @ 8x 2.614GHz
          `/+++ooooooooooooo/`         GPU: GeForce GTX 980, GeForce GTX 980
         ./ooosssso++osssssso+`        RAM: 464MiB / 16006MiB
        .oossssso-````/ossssss+`
       -osssssso.      :ssssssso.
      :osssssss/        osssso+++.
     /ossssssss/        +ssssooo/-
   `/ossssso+/:-        -:/+osssso+-
  `+sso+:-`                 `.-/+oso:
 `++:.                           `-/+/
 .`                                 `/
[user00@JPDM2 ~]$ nvcc --version
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2020 NVIDIA Corporation
Built on Wed_Jul_22_19:09:09_PDT_2020
Cuda compilation tools, release 11.0, V11.0.221
Build cuda_11.0_bu.TC445_37.28845127_0
[user00@JPDM2 ~]$
```

The JPDM2 Workstation

- Students can use the workstation to practice CUDA, develop the project for the exam, and for performance assessment.
- The following accounts were created for you students: `user01, user02, ..., use19`.
  `user00` is for me!
  Disk quota is 5GB.
- Access the workstation as ($ is the shell prompt):

```
$ ssh user{01|02|...|19}@160.97.63.93
```

  Please, fill the **JPDM2 student accounts.xlsx** in the Files section of the channel **Lectures** in the course team to reserve your JPDM2 account.
  There you will find the intial passowrd for your account that are invited to change at the first access. Please, do not annotate the new password in the shared document!!!

The JPDM2 Workstation

- The access to the workstation has no limitations; more users can work at the same time.
- A student at a time can submit its program for execution to guarantya proper performance assessment. For this purpose, we use the **Slurm** workload manager. Time limit is fixed to **5 minutes**.
  - Compile your program as usual, e.g.:
    ```
    $ nvcc vecAdd.cu -o vecAdd
    ```
  - write a script called `run.sh` or similar:
    ```
    #!/bin/sh
    srun vecAdd
    ```
  - enqueue the job for execution:
    ```
    $ sbatch run.sh
    ```
- Slurm quick start guide: `https://slurm.schedmd.com/quickstart.html`