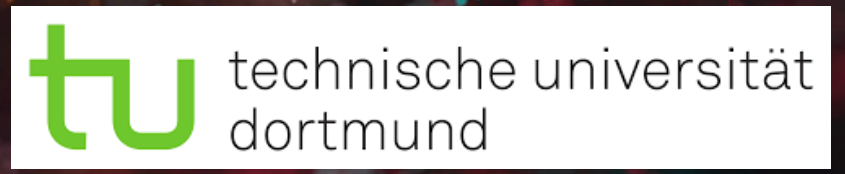
GPU PROGRAMMING: EXERCISES

Alessandro Scarabotto
TU Dortmund, Germany
Email: alessandro.scarabotto@cern.ch

Fast and Efficient Python Programming School
August 2024
Aachen, Germany



EXERCISES

- Login (or register) into VISPA cluster: https://vispa.physik.rwth-aachen.de
- You can find the exercises in the **examples page** of VISPA
- To be able to use a GPU you need to submit a job to the cluster:
  - **submit –f –M 2000 python example.py**
  - **submit –f –M 2000 ./example** (you need to create the executable first)
- You can open them in the file editor
- Running either from terminal or built-in command line in file editor page

EXERCISES

- Exercises order:
  - Hello World:
    - Running a first CUDA kernel
    - Testing on CUDA and PyCUDA
  - Vector addition:
    - Making a first threaded loop and strided-loop
    - Profiling your kernel to measure speed and GPU stats
  - Matrix multiplication:
    - How to speed a matrix multiplication improving the parallelization
    - How to exploit at best the GPU memory layout

EXERCISE 0: GPU STATUS

- Check the status of the GPU assigned to you with the Nvidia system management interface:
  - **submit –f –M 2000 nvidia-smi**
- You get GPU name, driver and CUDA version
- Get more information running the **device_properties.cu**
  - **.cu** is the extension of CUDA accelerated files
  - **nvcc** is used a compiler, use **–o** to specify output file and **–arch** to indicate the architecture (use native to checkout the current available GPU, otherwise check CUDA docs)
  - Then run the compiled output program
- Example:
  - **nvcc –arch=native –o device_properties device_properties.cu**
  - **submit –f –M 2000 ./device_properties**

EXERCISE 0: GPU STATUS

- You can get from here many info from the GPU you are using:
  - Grid, block and warp size
  - Global and shared memory
  - Number of streaming multiprocessors

- If you want to make sure no one is using your GPU simultaneously, you can run:
  - submit –f –M 2000 ./run-exclusive.sh ./device_properties

EXERCISE 1: HELLO WORLD

- Go on and modify **hello_world/hello_world.cu**
- You see the CPU calling of the printf: you can add the GPU version and kernel call
- The compile with nvcc and run the code which requires as input n_blocks and n_threads
- Example with 1 block and 1 thread:
  - **submit –f –M 2000 ./hello_world 1 1**
- Try to change n_blocks and n_threads and see what the code prints out

- Try to run hello_world in PyCUDA
- Copy your __global__ function into hello_world.py
  - **submit –f –M 2000 python hello_world.py**

EXERCISE 2: VECTOR ADDITION

- Perform vector addition allocating first memory in the GPU, perform the addition and finally return the output to the host
- Code takes 3 arguments: size of vectors, n_blocks and n_threads
  - Example: **./vector_addition 36 6 6**
  - You can vary n_blocks and n_threads, if you have written a correct strided-loop, you should get the correct answer
- The vector content is stored into host variable and transferred to global memory in the device, labels _h and _d are used to distinguish host and device variables

PROFILING

- Use the Nvidia profiler to profile and check performance of your code
- Example:
  - nvprof –s ./vector_addition 36 6 6
- Try to check: name of your kernel application, how many times it run, how long did it run for
- You can try to print other info, explore nvprof --help

EXERCISE 2: VECTOR ADDITION

- Try to run vector addition in pyCUDA
- Find vector_addition.py and modify it to make it work
- Define vectors with numpy with a specific type

EXERCISE 3: MATRIX MULTIPLICATION

- Using squared matrices to simplify the code, A x B = C
- All elements of C can be calculated independently → let's parallelize it
- Try to compile and run matrix_multiply.cu taking as input argument the matrix size:
  - ./matrix_multiply 512
- You can try to parallelize the work over threads in 2 dimensions threadIdx.x and threadIdx.y
  - Modify for loop and change n_threads
  - ./matrix_multiply_threads 512
  - Profile the application with nvprof and record the time taken by the processing
- Increase the parallelization adding more blocks (defined as a 2D grid) trying to make every thread calculating a single element of the final matrix C:
  - ./matrix_multiply_grid 512
  - Profile application and see if you increased speed
  - You can try to define loops over threads/blocks or make a strided loop

EXERCISE 4: SHARED MEMORY

- Use shared memory and tiling method to split data into memory
- Define the tile size (=32) at compile time
- All threads participate in loading tiles into memory, calculate partial result to registers and move to the next tile
- Try to compile and run matrix_multiply_shared.cu taking as input argument the matrix size:
  - ./matrix_multiply_shared 512
  - Profile the application and check time

EXERCISE 5: PRECISION

- It is important to understand the precision needed by your algorithm
- Requiring less or more precision both in the arithmetic or memory could impact performance
- We can test using storage as double, float and half precision and similarly for arithmetic (modifying storage_T and arithmetic_T): make all possible combinations
- Try to run setting a threshold = 0.01:
  - ./matrix_multiply_precision 512
  - Which threshold do you need to make each combination pass?
  - Run profiler to get timing of each combination

BONUS EXERCISE: LHCB TRIGGER

- You can try to git clone the LHCb software handling the trigger: Allen
- The documentation explains how to write a small kernel performing reconstruction of tracks from particles: https://allen-doc.docs.cern.ch/index.html
- You can define a quick kernel which reconstruct tracks from a simulated sample and counts them with an AtomicAdd

RESOURCES FROM EXERCISES AND EXTRA INFORMATION

- The exercises are adapted from: https://gitlab.cern.ch/dcampora/tcsc-gpulab
- You can find there also more exercises and in depth explanations in a jupyter notebook
- Many more exercises and examples carefully explained in the Nvidia docs: https://docs.nvidia.com/cuda/cuda-c-programming-guide/
- Examples for PyCUDA: https://wiki.tiker.net/PyCuda/Examples/