Practical Work: Implementing GPU Kernels

Master CSMI
Compilation & Performance
Bérenger Bramas

November 25, 2020

1 Summary

In this work, you will create GPU kernels that are going to be emulated on the CPU.

2 Ressources

3 Practical work organization (always the same)

In the practical work, you will obtain the code from my repository and push it to your repository. Therefore, you will have to clone one branch per session and push it to your own repository. It is mandatory that you **commit and push** frequently (after each question and at the end of the session, at least) such that I can easily look at what you have coded at the end of the session, how you did progress (and potentially compare it with the latest version you will have).

It is required that you filled the report.md file to let me know what you did.

In the rest of the document, we consider you have a repository named *cnp-tp-2020* on *git.unistra.fr* that is private but that I can access in read.

3.1 Get the practical work

Consider you are in your project directory do the following:

```
# Clone my repo
git clone https://git.unistra.fr/bbramas/csmi-tp-2020.git --branch=TP6 csmi-tp6
# If you use SSH, use:
# git clone git@git.unistra.fr:bbramas/csmi-tp-2020.git --branch=TP6 csmi-tp6
# Go in the newly created directory
cd csmi-tp6
```

3.2 Add your repository as remote

You will push on your own repository:

```
# Rename my remote
git remote rename origin old-origin
# Add your own remote
git remote add origin https://git.unistra.fr/[YOU LOGIN HERE]/cnp-tp-2020.git
# If you use an SSH key:
# git remote add origin git@git.unistra.fr:[YOU LOGIN HERE]/cnp-tp-2020.git
# Push the current branch and active the tracking
git push -u origin TP6
```

3.3 During the session and while you work on the project

After each question or important modification push the current changes:

```
# No matter where you are in the project directory
```

```
git commit -a -m "I did something"
git push
```

3.4 When you are done

You have fully finished your work (at most D+14 H-2):

```
git commit -a -m "I did something"
git push
```

3.5 Important!

Do not forget that you have to fill the report.md file and commit it. Remember to commit regularly to keep track of your work and let me see a history of it if I need it. Do not share any code with someone else, as I am here to answer all questions and support all of you. Remember that you have questions to answer in the moodle before the end of the session and that you must push your branch at the end of the session too. Additional credits can be obtained if you make some modifications after the session to improve your solution. Changes can be made until two weeks after the session minus two hours, ie you must push before the beginning of the n+2 practical work. Do not remove code from the test functions as I use them to evaluate your code. Therefore, if you need you can add extra functions for your own testing/debugging. If some of them are showing interesting things about your code, simply leave a comment in the code and the report.md.

3.6 Compilation

To compile, we use CMake:

```
cd TP6
mkdir build
cd build
cmake ..
make # Will make all
make something # Will build only something
VERBOSE=1 make # Will show the commands used to compile (including the flags)
```

4 Reminder

GPUs are streaming architectures that have a design different than the CPUs, and thus they have to be programmed differently. In class, we saw that a programming model is here to help using this *large SIMD* architecture. In this current work, you will program a GPU kernel that will be executed on the CPU using an emulator (therefore, do not expect to have performance).

5 Distance from the center

Update the distance.cpp file to create a kernel where the threads will put in a 2D array the distance from the center of the matrix. The array should be of type *double* but you should print the resulting matrix using integer.

For instance, if N is set to 20, the output should be:

```
distance :
 14  13  12  12  11  11  10  10  10  10  10  10  10  10  10  11  11  12  12  13
 13  12  12  11  10  10   9   9   9   9   9   9   9   9   9  10  10  11  12  12
 12  12  11  10  10   9   8   8   8   8   8   8   8   8   8   9  10  10  11  12
 12  11  10   9   9   8   8   7   7   7   7   7   7   7   8   8   9   9  10  11
 11  10  10   9   8   7   7   6   6   6   6   6   6   6   7   7   8   9  10  10
```

```
11  10   9   8   7   7   6   5   5   5   5   5   5   5   6   7   7   8   9  10
10   9   8   8   7   6   5   5   4   4   4   4   4   5   5   6   7   8   8   9
10   9   8   7   6   5   5   4   3   3   3   3   3   4   5   5   6   7   8   9
10   9   8   7   6   5   4   3   2   2   2   2   2   3   4   5   6   7   8   9
10   9   8   7   6   5   4   3   2   1   1   1   2   3   4   5   6   7   8   9
10   9   8   7   6   5   4   3   2   1   0   1   2   3   4   5   6   7   8   9
10   9   8   7   6   5   4   3   2   1   1   1   2   3   4   5   6   7   8   9
10   9   8   7   6   5   4   3   2   2   2   2   2   3   4   5   6   7   8   9
10   9   8   7   6   5   5   4   3   3   3   3   3   4   5   5   6   7   8   9
10   9   8   8   7   6   5   5   4   4   4   4   4   5   5   6   7   8   8   9
11  10   9   8   7   7   6   5   5   5   5   5   5   5   6   7   7   8   9  10
11  10  10   9   8   7   7   6   6   6   6   6   6   6   7   7   8   9  10  10
12  11  10   9   9   8   8   7   7   7   7   7   7   7   8   8   9   9  10  11
12  12  11  10  10   9   8   8   8   8   8   8   8   8   8   9  10  10  11  12
13  12  12  11  10  10   9   9   9   9   9   9   9   9   9  10  10  11  12  12
```

Ensure it works for different number of threads/groups.

6 GEMV for square matrices

In file matmat.cpp add the code to compute the matrix/matrix product on GPU. You can copy part of the distance.cpp file to make things easier. The ouput of the kernel should be copied into A_from_CUDA as it is then use to test the accuracy.

7 Sum all the values of an array

Edit the sumvalues.cpp file in order to perform the summation of all the values in the array. The file already contains the allocations and copies, but you should provide the kernel and also the calls to the kernels. The strategy to use is the following. Since there is no cheap inter-group communication in a GPU, we will call the kernel multiple times. As long as the size of the array is larger than twice maximum number of threads, we use the maximum of threads. Then, when the array get smaller we try to maximize the number of threads but having them to compute at most one summation (having the maximum degree of parallelism). Therefore, we must have a loop that will call the kernel multiple time and obtain partial summation until it gets one value. Here is a possible output:

```
Run for size 1000 nb groups 5 nb threads 10 total threads 50
Run for size 50 nb groups 3 nb threads 10 total threads 30
Run for size 30 nb groups 2 nb threads 10 total threads 20
Run for size 20 nb groups 1 nb threads 10 total threads 10
Run for size 10 nb groups 1 nb threads 5 total threads 5
Run for size 5 nb groups 1 nb threads 2 total threads 2
Run for size 2 nb groups 1 nb threads 1 total threads 1
Correct! Sum found is : 499500
```

Here we made the choice to reduce the number of group rather than the number of threads per group, but the inverse is also valide.