Universidade de Aveiro

Mestrado Integrado em Engenharia Computacional

Computação Paralela

Lesson 4: GPU – Programming with CUDA

Academic year: 2019/2020 Nuno Lau/Manuel Barroso

The computational power of GPUs is, currently, much higher than that of general-purpose CPUs. In order to take advantage of that computational power, and using the latest programming capabilities of GPUs, several tools have been developed that allow the use of GPUs to solve general purpose tasks. One of those tools is CUDA (Compute Unified Device Architecture) which is a parallel computing platform and programming model developed by NVIDIA to be used with their GPUs.

1. In this exercise we're going to edit and execute small programs that demonstrate the basics about the CUDA programming model. Extract the archive file **`cp_cuda1.tgz`** and work on the tasks that follow.

   1.1. Open the file **`arrad.cu`** (using **`vim`**) and examine the CUDA program which adds 2 arrays of **`float`**. Identify which code will run on the host and which code will be executed in the GPU. Compile the program and execute it. Analyze the output results and check if they are correct.

   1.2. Change the program **`arradd2.cu`** in order to execute 2 threads per block. Compile the program. Execute it using **`cuda-memcheck`** to check for errors. Run the program and check the correctness of the produced results. Change again the program in order to execute 4 threads per block. Check the results.

   1.3. Change the program **`arradd3.cu`** in order to use two-dimensional blocks of 2 rows and 2 columns and a one-dimensional grid. Compile the program. Execute it using **`cuda-memcheck`** to check for errors. Run the program and check that the results are as expected. Change again the program to use two-dimensional blocks of 4 rows and 2 columns. Check the results.

   1.4. Change the program **`arradd4.cu`** in order to use two-dimensional blocks of 2 rows and 2 columns and a two-dimensional grid (assume that the variables A, B and C represent matrices with **`NROWS`** rows and **`NCOLS`** columns). Compile, test using **`cuda-memcheck`**, run the program and check the produced results.

   1.5. Increase the dimension of the matrix to 1024 rows and 8 columns (1024 x 8), keeping the block dimension as 2x2. Measure the execution time in the GPU and in the host, considering that:

      1.5.1. threads with consecutive ids represent adjacent values in the rows of the matrix;

      1.5.2. threads with consecutive ids represent adjacent values in the columns of the matrix;

   1.6. Increase the number of threads per block to 8x8 and repeat the previous exercise.

2. Implement a new program which identifies the prime elements of a matrix with 1024x1024 integer values. The result should be an integer matrix of 1024x1024 in which the positions

of the detected primes (in the first matrix) are set to the value 1 and the other positions to the value 0. Compare the execution times in the host and in the GPU.

3. Implement a program in CUDA to multiply matrices.
   Tip: consider that the number of threads to execute is equal to the dimension of the result matrix.