Georg-August-Universität
Institut für Informatik
Practical Course on Parallel Computing ∘ SoSe2016
Prof. Yahyapour ∘ Prof. Grabowski ∘ Prof. Quadt

Assignment Sheet #11
Submission deadline: 28.06.2016, 16:00

Organizational hints and related background were presented in the lecture and also can be found in the lecture notes (uploaded to Stud.IP). Send your solution files in a compressed ZIP archive to fzhang@gwdg.de. You can find the programs or templates referred to in the assignments in the directory in Stud.IP.

Prepare environment
(1) You can use your own CUDA machine. Here you can find whether your machine has a CUDA-enabled GPU:https://developer.nvidia.com/cuda-gpus. Here you can find the installation guide of CUDA toolkit:
docs.nvidia.com/cuda/cuda-quick-start-guide/index.html#axzz4BrcI0U10.
(2) In case that you do not have a CUDA-capable machine at your disposable, you can use the following login to develop and test your solutions:

ssh <uni-user>@c6.num.math.uni-goettingen.de

Please subsitute <uni-user> with your student account. If your student account are not recognized by this server, access the following server firstly, and then your account is registered.

ssh <uni-user>@login.math.uni-goettingen.de

If your account still does not work, you can go to the media room of the Institute of Mathematics(Bunsenstr. 3-5) to login your account on any computer there, and then your account is registered.

Exercise 1 - Matrix Addition (5 Points)

Read and understand the code in the file *e1.cu*. It implements the addition of two matrixes with N*N blocks.

1. How does each block index the element of the two matrixes? Finish the kernel function.
2. Implement the addition of the two matrixes with 1 block of N*N threads by changing the file *e1.cu*. Do not forget to change the element index method.
3. How about implementing the addition by combining two-dimensional threads with two-dimensional blocks? Assume each block has 4*4 two-dimensional threads, how to set your two-dimensional blocks? Implement it by changing the corresponding part of the file *e1.cu*.

Exercise 2 - Matrix Multiplication (4 Points)

e2.cu is an template to implement matrix multiplication (C = A * B). Finish the two following portions in this file.

1. Launch the kernel on two-dimensional blocks. Each block further has two-dimensional threads. You can decide the size of each dimension.

2. Finish the kernel function to implement matrix multiplication. Note: each thread calculate one element of matrix C.

Exercise 3 - 1-d Stencils (6 Points)

Implement a stencil function. A 1-d stencil operates on a linear array. It transforms the array into an array of the same length. Each output element is the sum of the neighbouring elements in the original array. Assuming a radius of r, each element will be the sum of *2*r+1* elements: the original element itself and its r neighbours to the left and to the right. The illustration below shows an example with $r = 3$.

1. Implement the stencil using C in a sequential host-version and in a parallel CUDA version. At the boundaries assume that there are elements with 0-entries.

2. Run the sequential version and parallel version with various array lengths and r. You should see a clear speedup when comparing both implementations. Plot your performance results into a figure.