GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN
INSTITUT FÜR INFORMATIK

Practical Course on Parallel Computing · SoSe 2017

Prof. Yahyapour · Prof. Grabowski · Prof. Quadt

Assignment Sheet 8 · Submission Deadline: 12.06.2018, 3 pm

Organizational Issues

Organizational hints were already given in the lecture and can also be found as slides in StudIP. Please follow the following guidelines carefully:

- Solutions (including source code) must be send via E-Mail to christian.koehler[at]gwdg[dot]de **before** the deadline.
- Solution must be provided as either a **tar** or **zip** archive.
- If the assignment sheet contains questions (besides practical exercises), they must be answered in text form and provided as pdf inside the archive.
- Solutions must be submitted by groups of two students.

Hint: For resources aiding you in fulfilling these assignments, please refer to the lecture slides provided in Stud.IP. In particular, consult the *CUDA C Programming Guide*[1] for the practical exercises.

You can use your own CUDA hardware or the Scientific Compute Cluster[2] to run your code for assignments 2 and 3.

Assignment 1 – GPGPU with CUDA (*4 Points*)

a) What kind of performance challenges does recent CPU development face, i.e. why would one consider utilizing GPUs for general computing purposes in the first place? (1P)

b) Describe the GPGPU approach to parallel computing. Which steps have to be taken before code is actually run on the graphics device? (1P)

c) If a kernel is run on the GPU, how are its parallel threads organized? Pick a Compute Capability version of your choosing to explicitly discuss the constraints that are in place with respect to the number of threads an their memory requirements. (1P)

d) Under which circumstances can CUDA threads communicate without resorting to the global device memory? How can you ensure that in such a case reading and writing data happens in the correct order? (1P)

[1] `http://docs.nvidia.com/cuda/cuda-c-programming-guide/`

[2] `https://info.gwdg.de/dokuwiki/doku.php?id=en:services:application_services:high_performance_computing:start`

Assignment 2 – Data reduction (*5 Points*)

a) For an array `float arr[N]`, where N is a power of 2, start by summing its entries using host code exclusively. (1P)

b) Use CUDA to obtain the same numerical result using a kernel, but keep the pattern you used in a) to evaluate the sum. (2P)

c) Is there a better way to establish the sum with parallel code? Sketch the resulting data dependency between the array elements and their partial sums and compare the time this version takes compared to b). (2P)

Assignment 3 – Heat equation with fixed boundary (*6 Points*)

a) For $n = 512$, initialize a two-dimensional `float` array describing the $n \times n$-matrix A which is given in the following way:

$$A_{ij} = \cos\left(\frac{4\pi \cdot i}{n-1}\right) \cos\left(\frac{4\pi \cdot j}{n-1}\right)$$

if $i = 0$, $i = n-1$, $j = 0$ or $j = n-1$, otherwise $A_{ij} = 0$. (1P)

b) Apply the Jacobi iteration by replacing

$$A_{ij} \leftarrow (A_{i-1,j} + A_{i+1,j} + A_{i,j-1} + A_{i,j+1})/4$$

for all $0 < i < n-1, 0 < j < n-1$ for the entire array (so just change the interior, leaving the boundary given in a) intact). Repeat this process until the maximal difference per entry between the old and new array falls below a threshold $\epsilon = 0,01$. (1P)

c) Parallelize the initialization and iteration step with CUDA. You can initialize A either on the host (+copy) or directly on the device. (2P)

d) Visualize your result, for example with `gnuplot`. (1P)

e) The CUDA Toolkit supports the creation of events[3] which can be used with the functions `cudaEventRecord` and `cudaEventElapsedTime` to measure the time kernels need to run. Use this mechanic to compare the execution time of the serial and parallel solution. (1P)

[3] `http://docs.nvidia.com/cuda/cuda-c-programming-guide/#events`