Heidelberg University
Institute of Computer Engineering
Computing Systems Group

Holger Fröning
Christian Alles

GPU Computing
Winter Term 2021/2022

Exercise 8

- **Return electronically until Monday, Jan 31, 2022, 09:00**
- **Include your names on the top sheet. Hand in only one PDF.**
- **A maximum of four students are allowed to work jointly on the exercises.**
- **When you include plots add a short explanation of what you expected and what you observed.**
- **Hand in source code if the exercise required programming. You can bundle the source code along with the PDF in a .zip file.**
- **Programming exercises can only be graded if they compile on the cluster head node. Make sure to document the commands which you used for the compilation.**

8.1 Reading

- Justin Holewinski, Louis-Noël Pouchet, and P. Sadayappan. 2012. High-performance code generation for stencil computations on GPU architectures. *In Proceedings of the 26th ACM international conference on Supercomputing* (ICS '12). ACM, New York, NY, USA, 311-320.

(5 points)

Goal of the exercise and possible flexibilities

The general goal of the exercise is for you to get to know OpenACC.
As such we give you the possibility to work on a different problem than the heat relaxation presented in the next section. In general you can pick a problem of your choosing: Something that you always wanted to port to a GPU, a common task that seems interesting to you or alternatively you can base your work on a more complex CUDA code which implements the conjugate gradient method using a 5pt stencil. This code will be provided to you on request.
The only requirement is that the general scope of your submission is of the same complexity as the following tasks. In particular your submission should include:

- An explanation of what for a problem you want to run using OpenACC.
- Serial CPU version of your code.
- OpenACC version without optimizations.
- OpenACC version with optimizations. Including a discussion of what you tested and the results you got.

Using OpenACC on the brook cluster

OpenACC is included with the Nvidia HPC SDK. As such you will need to load the HPC SDK instead of CUDA. This can be done on the head node with the following command:

```
1 module load nvhpc/21.9
```

For compilation you will need to use the *pgc++* compiler. A full command could look like this:

```
1 pgc++ -O2 -acc -ta=tesla,time -Minfo=accel -o <output_file> <input_files>
```

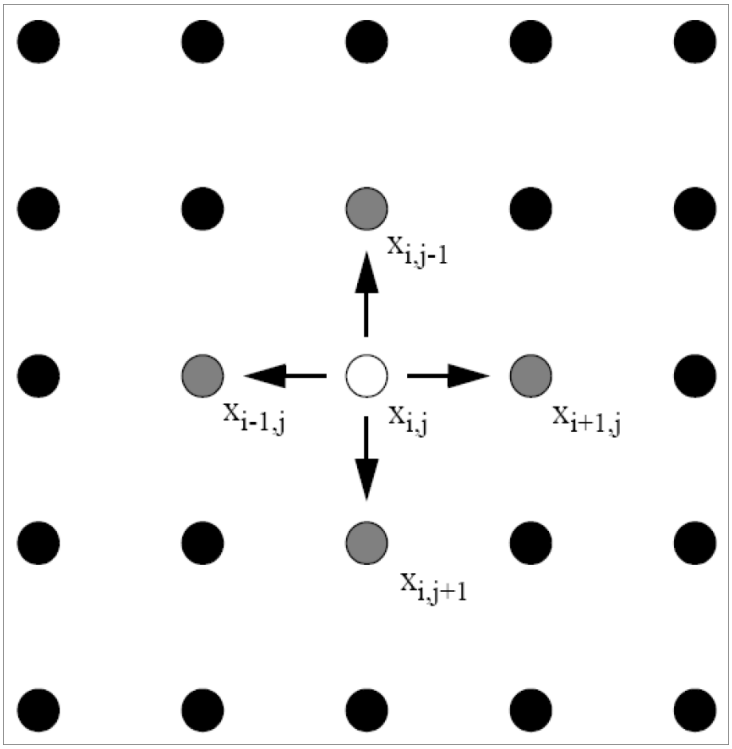
Here *-acc* instructs the compiler to use openACC, *-ta=tesla,time* instructs it to use the Nvidia GPU as an accelerator and to print out timing statistics for calls to the accelerator. Finally *-Minfo=accel* will print out information about how and if automatic acceleration was implemented during the compilation.
Important: We are currently experiencing problems with the HPC SDK on gpu08. Until this is resolved you should explicitly select gpu09 for your experiments (add *-w gpu09* to your srun call or sbatch configuration).

Heidelberg University
Institute of Computer Engineering
Computing Systems Group

Holger Fröning
Christian Alles

Heat Relaxation

Relaxation is a mathematical technique used for modeling or the simulation of dynamic processes (heat distribution, material yielding, etc.). In this technique, the solution of a multi-dimensional function is mapped to a discrete grid. The value of a given grid point is dependent on the values of the previous time step, usually of the point itself and its surrounding neighbors. Implement a program, which calculates the new value of grid points as average of the point itself and its four direct neighbors.



The value of a grid point for the next time step t+1 calculates as follows, note that i and j are the coordinates of this grid point:

$$x_{i,j}^{t+1} = x_{i,j}^{t} + \Phi \cdot ((-4) \cdot x_{i,j}^{t} + x_{i+1,j}^{t} + x_{i-1,j}^{t} + x_{i,j+1}^{t} + x_{i,j-1}^{t}) \qquad \Phi = \frac{24}{100}$$

- Dynamically allocate a grid of NxN single precision floating point values. The allowed value range is [0,127].
- Permanently inject heat in the topmost grid points (j=0), with i in between [N/4, 3N/4]. These gridpoints are statically set to 127. The value of all other grid points at the borders is statically set to 0.
- Additionally initialize a patch of medium heat (100) in the center of the matrix between [N/4, 3N/4] for i and j. This patch should be non-static and vanishes over time.
- The size of the grid N shall be configurable using a command line parameter.
- Automatically stop the program when the heat relaxation stabilizes, i.e. the maximum change between two iterations becomes small. A threshold of about 10^{-3} for the maximum change should be sufficient.

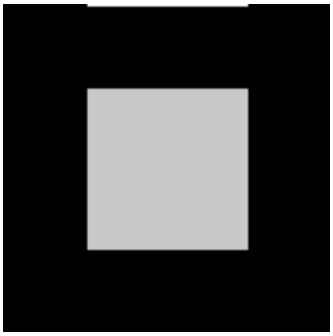


Figure 1: Initial setup of the heat relaxation (128x128 matrix)

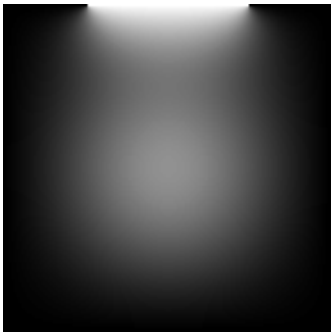


Figure 2: Heat relaxation after 1000 iterations (128x128 matrix)

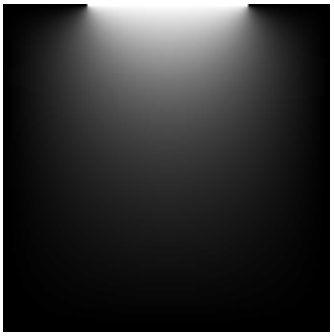


Figure 3: Heat relaxation after 9000 iterations (stabilized) (128x128 matrix)

8.2 Stencil Code – CPU implementation

- Write an unoptimized CPU program that performs the described calculation in a serial fashion.
- Visualize the output to check correctness.
- Plot how the solution stabilizes over the number of iterations.

(15 points)

8.3 Stencil Code – Naïve OpenACC implementation

- Extend your CPU implementation using OpenACC compiler pragmas.
- Automatically stop the computation once the heat relaxation stabilizes or after a fixed number of maximum iterations. Note that the number of iterations it takes to stabilize directly depends on the matrix size.
- Do not apply any optimizations just yet.
- Visualize the output to check correctness.

(10 points)

8.4 Stencil Code – Optimized OpenACC implementation

- Optimize the previous program using different techniques provided by OpenACC:
  1. Vary launch the kernel launch parameters (*vector_length* and *num_gangs* clauses).
  2. Let OpenACC use the shared memory (*cache* directive). Note: OpenACC may already use some of the shared memory by default. The *cache* directive is only a hint to the compiler. To find out how much shared memory was actually used you can profile your code with Nsight Compute.
  3. Optimize data movements between the host and device. Note: Depending on your naïve implementation there may be little to do here.
- Compare the performance of the optimized and non-optimized version for different matrix sizes.
- Also plot at which number of iterations the solution stabilizes for different matrix sizes. This should be the same for the optimized and non-optimized version of your code.

(20 points)

Holger Fröning
Christian Alles

8.5 Willingness to present

Please declare whether you are willing to present any of the previous exercises.
The declaration can be made on a per-exercise basis. Each declaration corresponds to 50% of the exercise points. You can only declare your willingness to present exercises for which you also hand in a solution. If no willingness to present is declared you may still be required to present an exercise for which your group has handed in a solution. This may happen if as example nobody else has declared their willingness to present.

- Reading
- Stencil Code – CPU implementation
- Stencil Code – Naïve OpenACC implementation
- Stencil Code – Optimized OpenACC implementation

(25 points)

Total: 75 points