Exercise 5

To compile: unzip our uploaded code, and run `make` inside `code/`. The slurm scripts are stored inside `code/slurm/`.

To debug: run the debug outputs (`*.dbg`) and attach gdb to respective pids

5.1 Reading

The paper "Score-P - A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir" gives an overview of the Score-P suite, a performance measurement toolkit for high performance computing. The authors highlight the redundancies that come with using multiple specialized tools instead of a joint infrastructure especially when it comes to data acquisition. The proposed Score-P frameworks aims to fix these redundancies and provide ease of installation. The frame works bundles many different performance measurement libraries and tools (OpenMP, MPI planed support for CUDA) and instrumentation to automatically link and use them. Also The performance in comparison to previous and constituent frameworks is compared.

Avoiding code duplication by having a joint framework seems to be a very good idea as long as the framework is flexible enough to incorporate new specific features. We accept the premise of the paper that a unified data format and instrumentation would allow for less redundancy and better interplay. In our opinion a frame work like this should be as unopinionated as possible as not to be restrictive for further additions to the framework. As alway when introducing new standards there is a danger of creating yet another framework that has to be used in conjunction with others. However the project looks promising.

5.2 Heat Relaxation — Sequential Implementation

(see `code/`) Visualization in img/heat.gif

5.3 Heat Relaxation — Experiments

The Sequential Heat Relaxation Implementation of subsection 5.2 resulted in the values observed in Table 1. These Numbers resulted from 100 iterations per grid size, with two buffers.

Table 1: Resulting Numbers for a sequential Heat Relaxation

Grid Size	Time per iteration/s	Flops total	GFLOP64/s
128x128	0.0001	11290300	2.0623
512x512	0.0041	182784700	0.4470
1024x1024	0.0112	732570300	0.6544
2048x2048	0.2091	2933146300	0.1403
4096x4096	0.8034	11738317500	0.1461

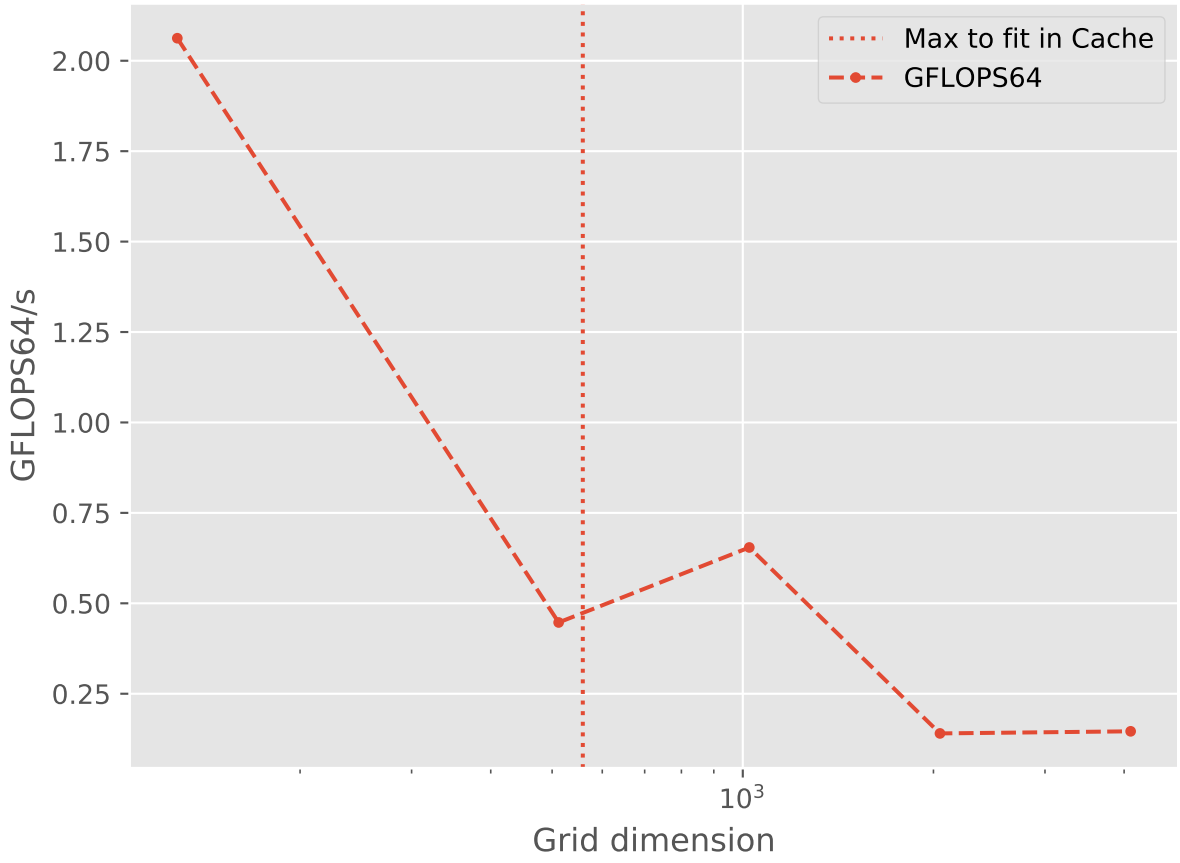

Figure 1: Resulting GFLOPS64 for increasing Dimensions, with the maximum dimension for fitting both buffers into cache marked

The resulting numbers are not surprising. The Size Limit for fitting both buffers into Cache[1], when reached will result in the effective GFLOPS plummeting.

$$\text{Max Grid Size} = \sqrt{\frac{\text{Cache Size}}{\text{Byte per Double} \cdot \#\text{ of Buffers}}} \tag{1}$$

$$= \sqrt{\frac{10\,\text{MB}}{8\,\text{B} \cdot 2}} \tag{2}$$

$$\approx 790 \tag{3}$$

But these Numbers represent the maximum possible dimension if nothing else has to be stored in the cache.

When increasing the dimension further, a further decrease in performance will be expected, due to an increase in cache misses.

5.4 Heat Relaxation — Pre-considerations for parallelization

Some considerations for applying MPI to the heat relaxation problem:

[1] 10MB for a Intel Xeon E5-1620

1. To de-compose the complete problem into subtasks would mean splitting the grid into parts for which a separate rank is responsible. This would require the ability to communicate between "neighboring" ranks.

2. For this grid either 1D or 2D partitioning is possible, e.g.: splitting the MxN Grid into 4 rectangles with one dimension equal to the original grid ($\frac{M}{4}$xN or Mx$\frac{N}{4}$) or splitting the complete Grid into 4 rectangles of the same aspect ratio as the original ($\frac{M}{2}$x$\frac{N}{2}$).

   The better partitioning for a given Problem would be the one, in which we are required to exchange fewer grid points (Volume-Surface Rule). Figure 2 demonstrates this rule: here all fully filled grid-points are calculated by a corresponding rank. The not filled outlines represent the points required by the same colored rank for its next iteration, which are calculated by a neighbor. When these surfaces are calculated they are exchanged with the corresponding neighbor.

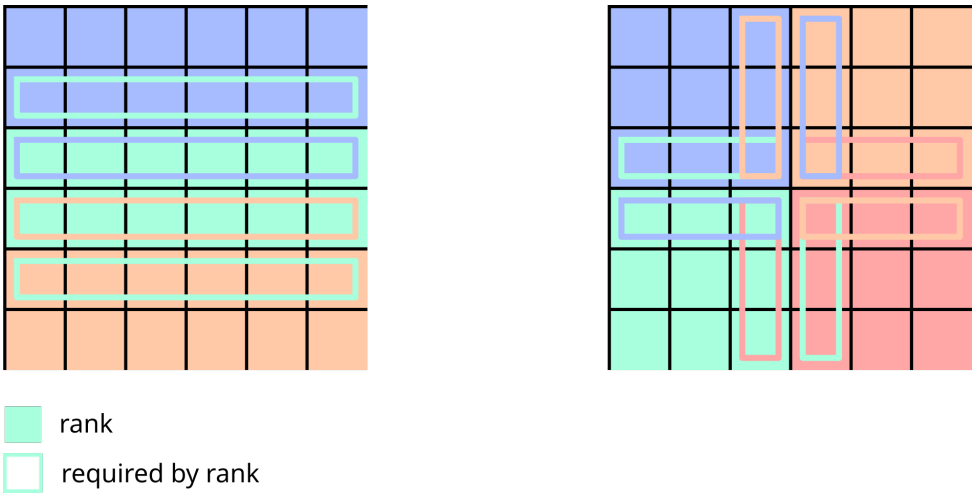

Figure 2: Example of Partitioning Schemes, 1D (NS) on the left, 2D (NEWS) on the right

   With the Example of before (4 ranks, MxN Grid, 1D (North-South) and 2D (NEWS) Partitioning), we would need to exchange the following number of points for each method:

$$\begin{aligned} \#\text{ exchanged points}_{\text{1D, NS}} &= N \cdot 2 \cdot \#\text{ partitions} - \#\text{ border partitions} && (4)\\ &= N \cdot 2 \cdot 4 - 2 = \mathbf{8N - 2} && (5) \end{aligned}$$

$$\begin{aligned} \#\text{ exchanged points}_{\text{2D, NEWS}} &= \sqrt{\#\text{ partitions}} \cdot 2 \cdot N \\ &\quad + \sqrt{\#\text{ partitions}} \cdot 2 \cdot M - \#\text{ border partitions} && (6)\\ &= \mathbf{4N + 4M - 4} && (7) \end{aligned}$$

   That means for our use case (big and nearly square grids) the 2D partitioning will always require less points to be exchanged.

3. We know that only the points near the surfaces need to be exchanged. To minimize delays due to these calculations, we should calculate the surface points ideally the moment their

required neighbors are available and send them off immediately. This is allowing the rest of the volume to be calculated in the meantime.

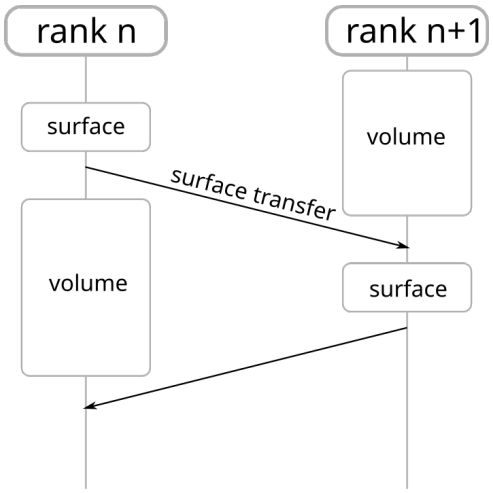

Figure 3: Transfer and Priority of Surface Calculations