Heidelberg University
Institute of Computer Engineering
Computer Engineering Group

Holger Fröning
Bálint Soproni

Introduction to High-Performance and Distributed Computing
Winter Term 2023/2024

Exercise 5

- **Hand in via Moodle until 23:59 on Monday 4 December, 2023**
- **Include all names on the top sheet. Hand in a single PDF.**
- **Please compress your results into a single archive (`.zip` or `.tar.gz`).**
- **Please employ the following naming convention `hpc<XX>_ex<N>.{zip,tar.gz}`, where `XX` is your group number, and `N` is the number of the current exercise.**
- **A maximum of three students is allowed to collaborate on the exercises.**
- **In case an exercise requires programming:**
  - **include clean and documented code**
  - **include a Makefile for compiling**

5.1 Heat Relaxation II – Parallel implementation based on 1D-row partitioning

Use the program developed in exercise 5 to start with this exercise. The configuration is the same as before, but this time the goal is to parallelize this program for MPI.

- In this exercise, perform a 1D row partitioning. Thus, each process has to exchange only consecutive addresses (C has a row-based layout in memory) to its neighbor, simplifying the communication at the cost of additional communication per process.
- Make a new communicator with topology information attached. See *MPI_Cart_create*. The topology information helps to determine processes working on neighbor data domains. See *MPI_Cart_shift*.
- Ensure that you use two arrays for the current and previous grid. Do not exchange these two arrays by copying (after one iteration), instead just exchange the pointers to them. Saving one copy saves plenty of processor cycles.
- Try to maximize overlap by first calculating the borders and send them out in a non-blocking fashion.
- Write a suitable parallel program that performs the described calculation. Test your program extensively. Compare results against the sequential implementation.

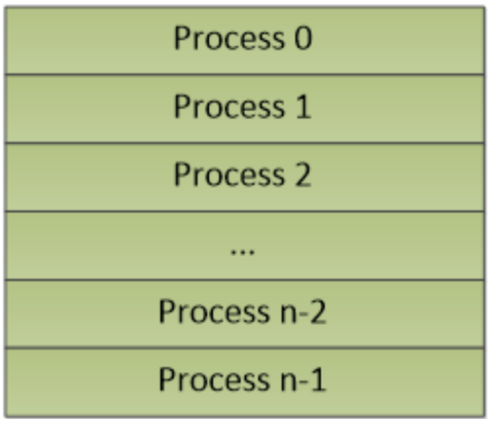

Figure 1: 1D Partitioning

(30 points)

Holger Fröning
Bálint Soproni

5.2 Heat Relaxation II - Experiments

- Measure the average time for grid sizes of 128×128, 512×512, 1024×1024, $2k \times 2k$, $4k \times 4k$. Report the average time of one iteration by performing for instance 100 iterations, measuring the time with a suitable function (e.g., *gettimeofday()* in Linux) and dividing by the number of iterations. Do not include time for initialization or output.
  Use compiler-specific optimizations to minimize the runtime.
- Increase the number of worker processes from 2 to 12 for these grid sizes.
- Report how mapping was done and why you chose it this way!
- Fill out the following tables and interpret results:

Time/iteration							
Grid size	Sequential	NP = 2	NP = 4	NP = 6	NP = 8	NP = 10	NP = 12
128×128							
512×512							
1024×1024							
$2k \times 2k$							
$4k \times 4k$							

Speedup (compared to sequential execution time)							
Grid size	Sequential	NP = 2	NP = 4	NP = 6	NP = 8	NP = 10	NP = 12
128×128							
512×512							
1024×1024							
$2k \times 2k$							
$4k \times 4k$							

Efficiency (compared to sequential execution time)							
Grid size	Sequential	NP = 2	NP = 4	NP = 6	NP = 8	NP = 10	NP = 12
128×128							
512×512							
1024×1024							
$2k \times 2k$							
$4k \times 4k$							

(30 points)

Note: Ensure that your program is running without problems even with different configurations, like more processes or larger grids.

Heidelberg University
Institute of Computer Engineering
Computer Engineering Group

Holger Fröning
Bálint Soproni

5.3 Heat Relaxation II - Tracing

In this exercise, you will be performing basic tracing on your parallel heat relaxation program.
Answer the following question in a couple of sentences (each):

1. Knowing the implementation of your program, what results would you expect from tracing?
2. Did the experiments you performed in the previous exercise (5.2) meet your expectations?
3. Try the different tools provided by score-P as explained below and find appropriate metrics. Check against your intuition. Does the profile confirm your expectations? Interpret the results.
4. Did tracing help you gain further insight into the actual run-time behavior of your program?

Trace multiple runs of your program with varying MPI configurations and problem sizes.

(20 bonus points)

5.4 Willingness to Present

- Parallel implementation based on 1D-row partitioning
- Experiments
- Tracing

(40 points)

Total: 60 points (+20 bonus points) + 30 (+10 bonus)

Quick Introduction to Score-P

Connect to the Octane Cluster and load the required modules

```
$ module load scorep-X.X
```

To produce an instrumented executable, compile your program using the Score-P executable
scorep [mpicc|mpic++] code_file, e.g.

```
$ scorep mpicc my_mpi_program.c
```

If you are using `make` just update your `CC/CXX` definition:

```
- CC = mpicc
+ CC = scorep mpicc
```

Now execute your instrumented application normally with `srun/sbatch`.
Once the job is completed there will be a new folder with the **scorep** prefix in your current working directory. (The execution will fail if this folder already exists. Therefore, you might have to manually rename it!) Inside the folder there should be a `profile.cubex` file.

For a basic overview, run `scorep-score -r <profile>`. This will give you Information on time spent per subroutine (absolute and percentage), number of visits, and the maximum buffer size.

More detailed information can be obtained from `cube_info`, `cube_score`, `cube_rank`, and `cube_calltree`. Please have a look at the command line tools' documentation here:
`https://apps.fz-juelich.de/scalasca/releases/cube/4.7/docs/CubeToolsGuide.pdf`.
Most of the aforementioned tools also have a `--help` option.