Simulating Heat Distribution in a Metal Plate Using Monte Carlo and MPI

Artur Sahakyan
Supervised by Bogdan Milićević
Erasmus+ WICT

January 8, 2025

1 Introduction

This seminar aims to simulate heat distribution (transfer) across a 2d grid (to resemble a metal plate). Simulation uses particles, which can then be visualized as pixels with temperature values. The process is stochastic, meaning particles move randomly from the heat source (Fig. 1). The farther away particles go from the source, the less heat they carry (depending on conductivity) [RHC+98]. The process is transient, and it will reach a steady state (in this case thermal equilibrium). The goal of the work is to demonstrate the effectiveness of parallel computation for heavy workloads through the experiment.

2 Methodology

High performance computing (HPC), according to Hager and Wellein, deals with algorithm implementation and related hardware [HW10].

Parallelism uses threads and processes may communicate in different ways taking into account the architecture constraints [ST98]. There are 3 main ways for communication - message passing, transfers through shared memory, direct remote-memory access [ST98].

The seminar implementation uses message passing techniques to solve the problem more efficiently [GLDS96]. According to Barker [Bar15], message passing interface (MPI) specifies the functionality of high-level communication routines and MPI's functions give access to low-level implementations that handle sockets, buffering, data copying, message routing, etc.

The 2D grid is partitioned among multiple processes to distribute the tasks into subtasks. Grid is divided into horizontal slices, with each MPI process solving for its own slice. When number of processes increases, it leads to potentially (potentially, since the process is random) faster execution time.

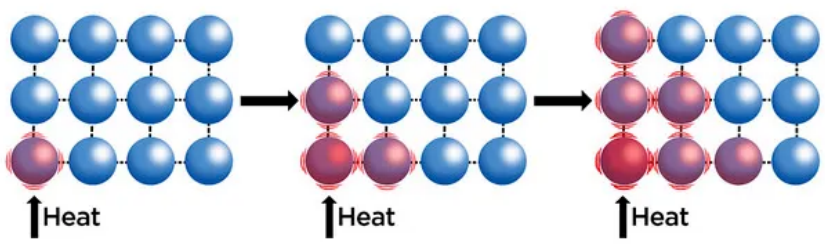

Figure 1: The higher temperature object has atoms with higher energy levels and they will move toward the lower energy atoms in order to establish an equilibrium. [Ber18]

	Parallel	Sequential
execution time (seconds)	0.26	0.25
memory usage (MB)	0.17	0.16
grid size	100	100
steps	100	100

Table 1: Results with small parameters

	Parallel	Sequential
execution time (seconds)	9.41	15.62
memory usage (MB)	0.63	0.61
grid size	200	200
steps	400	400

Table 2: Results with large parameters

3 Hardware

The CPU used during experiments is 13th Gen Intel(R) Core(TM) i9-13900H. The hardware architecture must be capable of parallel processing not to constrain the implementation [ST98]. i9-13900H has 14 cores and 20 threads, is built on Intel 7 manufacturing technology, with a max. frequency of 5400 MHz. The max load during execution for grid size 200 and steps 400 is 14%. It might be wise to utilize GPU (RTX 4050) of laptop as well.

4 Software

MPI for Windows 11 has been installed prior to installing mpi4py. The following paragraph focuses on key methods utilized in code.

MPI.COMM_WORLD is a communicator for representing processes in the MPI environment, comm.Get_rank() retrieves the rank (unique id) of the calling process within the communicator. It identifies which slice of grid a process should solve for, comm.Get_size() retrieves the total number of processes in the communicator, comm.Scatter(sendbuf, recvbuf, root=0) distributes data chunks from root process to other processes in communicator. comm.Gather(sendbuf, recvbuf, root =0) combines data slices after the simulation.

5 Results And Discussion

The results show that for smaller grid size (100) and steps (100) there is no big difference between the code with parallel processing (mpi4py) and the sequential processing (Table 1). However, once the grid size and steps increase the difference is obvious (Table 2).

Visually, there is a difference in number of particles based on grid size and steps (Fig. 2, Fig. 3).

It is also worth noting that due to the increase in grid size and steps memory consumption has slightly risen (+2MB) in HPC execution.

`mpi4py` is effective for large-scale distributed computing tasks and as seen on task with small parameters the sequential implementation has slightly better performance in execution time and memory (Table 1). In case of small parameters, communication overhead causes bottleneck and focusing on accelerating the numerical calculations instead would be better (one possible approach is to use `Numba`).

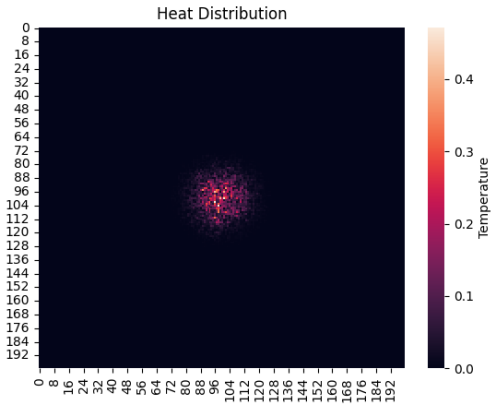

Figure 2: Heat distribution with larger parameters

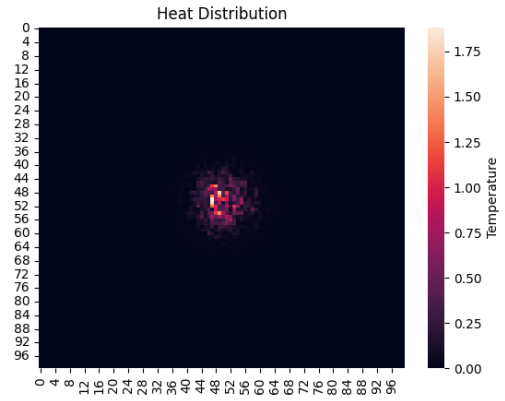

Figure 3: Heat distribution with smaller parameters

5.1 Future work

Given the implementation difficulty and benchmarks, it might be easier and more effective to utilize Numba framework. It has supported Numpy features and can seamlessly integrate with Python code by using decorators. The same experiment, but with Numba is on agenda.

Additionally Numba provides interface for CUDA GPU toolkit, which means processes can be sent to GPU ([Tea25]). The same experiment with GPU support is on agenda.

6 Acknowledgements

The seminar project was implemented thanks to Erasmus+ WICT, Bogdan Milićević lectures and supervision.

7 Conclusion

The seminar project modelled the heat transfer. Parallel, sequential computational techniques were applied and then compared by their parameters, results. It is important to estimate the computational load and based on that choose efficient techniques for faster execution. The goal of showing the effectiveness of parallel processing on larger-scale computational tasks has been achieved through experiments.

References

[Bar15] Brandon Barker. Message passing interface (mpi). In *Workshop: high performance computing on stampede*, volume 262. Cornell University Publisher Houston, TX, USA, 2015.

[Ber18] Brian Bergmann. Heat transfer – 3 types, 2018. Accessed: 2025-01-09.

[GLDS96] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. A high-performance, portable implementation of the mpi message passing interface standard. *Parallel computing*, 22(6):789–828, 1996.

[HW10] Georg Hager and Gerhard Wellein. *Introduction to high performance computing for scientists and engineers*. CRC Press, 2010.

[RHC+98] Warren M Rohsenow, James P Hartnett, Young I Cho, et al. *Handbook of heat transfer*, volume 3. Mcgraw-hill New York, 1998.

[ST98] David B Skillicorn and Domenico Talia. Models and languages for parallel computation. *Acm Computing Surveys (Csur)*, 30(2):123–169, 1998.

[Tea25] Numba Development Team. *CUDA Programming with Python — Numba Documentation*, 2025. Accessed: 2025-01-09.