MPI-Implementation of the two dimensional heat equation

Parallel Computing I 2021 Project

Brian Zahoransky
Friedrich Schiller University Jena
Germany
brian.zahoransky@uni-jena.de

ABSTRACT

The project is part of the course Parallel Computing I at the Friedrich Schiller Univesity Jena. The goal of the project is to gain insight into development environments with multiple nodes and programming them by applying the Message Parsing Interface (MPI). This work has five parts. The introduction discusses the theoretical background of the thermal conduction and formulates a discretisation. Later on, it is shown how the approach can be implemented on a multiprocessor system using MPI. The developed program aims to gain information about how communication and computing times affect the time to solution. Its structure is described in the third section. The experiments on a local single processor machine showed the correctness of the algorithm. The time measurement on the ara cluster demonstrated the importance of uniform load distribution. The inner nodes which performed more computations due to the design of the program slowed it down noticeably. Thus, execution time may be shortened by choosing smaller ghost zones or realizing an approach with smaller sub-fields in the middle and huger ones at the edges and corners.

KEYWORDS

Heat Equation, MPI, Implementation

1 INTRODUCTION

1.1 Overview

The goal is to develop a mpi program of the heat equation and to analyze computing and communication times after running it on the ara cluster. The following chapter discusses the background of the heat equation and shows typical applications. Later on, the created program is described. The third chapter presents the conducted experiments and the next chapter shows the results. This chapter also tries to give explanations for the findings. The last section discusses the results and lists optimizations, which could be applied in further experiments.

1.2 Background

In wikipedia the heat equation is stated as follows.[1] "In mathematics, if given an open subset U of $\mathbb{R}^n$ and a sub-interval I of $\mathbb{R}$, one says that a function $u : U \times I$ is a solution of the heat equation if

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x_1^2} + \cdots + \frac{\partial^2 u}{\partial x_n^2}, \tag{1}$$

where $(x_1, ..., x_n, t)$ denotes a general point of the domain." Typically, t refers to time and $x_1, ..., x_n$ represent spatial variables. Thus, the right hand side is equal to the Laplacian and could be written compactly as

$$\frac{\partial u}{\partial t} = \Delta u. \tag{2}$$

Anyway in physics, especially in thermal conduction, it is often considered a cartesian coordinate system with two or three spatial variables. In such applications, commonly, a coefficient α is multiplied. It stands for the diffusivity of the medium. This project looks at the two-dimensional phenomenon, which leads to the following statement.

$$\frac{\partial u}{\partial t} = \alpha \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) \tag{3}$$

At this point starts the task description for the project. It defines a spatial discretisation through a $n \times n$ grid with an equidistant distance h for both spatial dimensions. Note that $n \in \mathbb{N}$ and $h = 1/n$. The discretisation of time Δt is equidistant as well. This allows to discretize u as below.

$$\begin{aligned} u_{t+1}(i,j) = u_t(i,j) + \alpha \frac{\Delta t}{h^2} \\ * [\, u_t(i-1,j) + u_t(i+1,j) \\ + u_t(i,j-1) + u_t(i,j+1) - 4u_t(i,j)] \end{aligned} \tag{4}$$

The variables $i, j < n$ describe the position in the grid while $t+1$ and t are the next and the current time step. This type of computation is known as five-point-stencil. The algorithm needs to compute the next time step of a point five points of the previous iteration, the point itself and its four neighbours.

1.3 Multiprocessor systems

The previous finding especially equation 4 allows simulating the process of on a single processor machine. In a multiprocessor system, however, communication also has been taken into account. To use more than one node, the field has to be split into parts. Each sub-field is stored on one processor. To compute the next time step of the margin values of such a sub-field, a processor needs the neighbour values which are stored at a different processor. These values have to be transferred each time step. Consider a system with 16 processors and a $n \times n$ field. To minimize the number of transferred values, the field is split into 16 square blocks and distributed as in table 1 shown.

Table 1: Distribution of the field. Processor zero manages the upper left block of the original $n \times n$ field. Each block has a size of n divided by four since there are 16 processors.

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Have a look at processor five. It communicates with its four neighbours every iteration. Since communications commonly come with overhead it appears to be reasonable to reduce its frequency of occurrence. This can be achieved by increasing the ghost zone size. If a processor just receives one column from its neighbours in the x-direction and one row from its neighbours in the y-direction, the ghost zone is one. If a processor receives two rows and columns, the ghost zone size is two. Note two things. First, the ghost zone size is equal to the number of local iterations which can be performed before further communication is required. Second, to perform multiple iterations locally, the corners of the diagonal neighbours are also required. To avoid diagonal communication, non-diagonal communication can be executed in an ordered way. First, the processors send their margins to the right and left neighbour. After receiving these messages, the processors have the needed data for their upper and under neighbours. Now, the processors can communicate in the y-direction.

2 THE PROGRAM

2.1 The project folder

The program folder contains a source directory, an include-directory, a cmake-script, a host file, and a run-script. The source directory includes two main programs, one with blocking send and receive operations, and a second version with non-blocking operations. The header files, stored in the include directory, provide the applied subroutines. Executing the cmake-script returns the two executables of the main programs. The host file allows starting any number of MPI processes on a local machine. The run script configures the slurm environment at a cluster and executes the main programs.

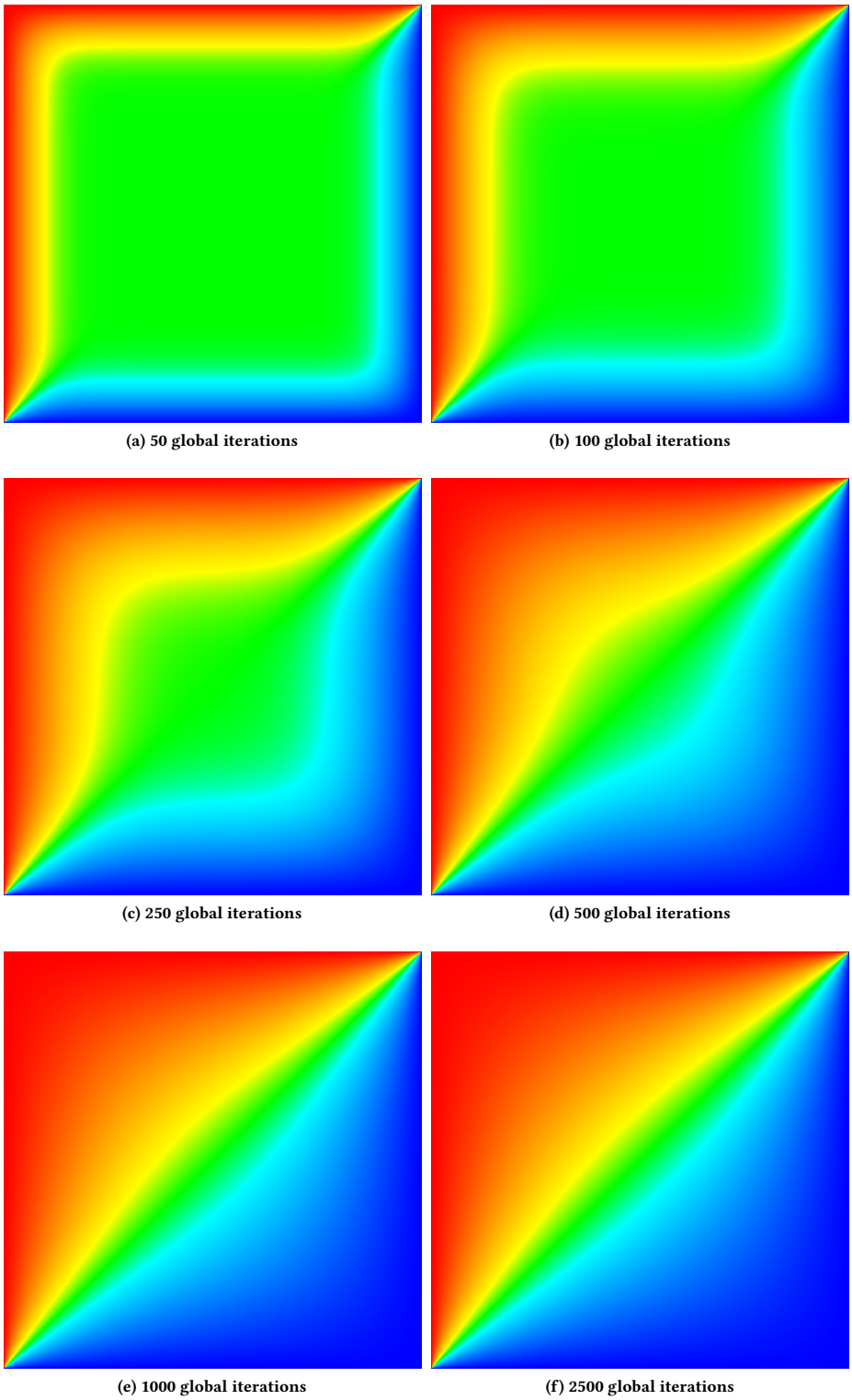
2.2 Preparation

However, the main program is consisting of three parts. The first one prepares the main loop and covers the following aspects:

- The user input is checked for compatibility. The program expects three input parameters field size, the number of local iterations or ghost zone size, and the number of global iterations. Thus, the overall amount of iterations is the number of local ones multiplied with global ones.
- Help variables such as x- and y-position of the processes are defined. Note that the number of processes has to be a square number. They will be sorted as a two-dimensional grid with the same number of processes in each dimension.
- The field is initialized and the memory for the communication buffers are allocated. Every process will return a sub-field of the same size. Due to this, it should be ensured that the number of points per dimension is dividable through the root of the number of processes.
- The communication parameters are declared.
- The time measurement is set up.

2.3 The main loop

The second part is the main loop. The loop begins with the processors send and receive the updates for their margin values in two phases. Consider the process grid, first, the communication operations follow the x-direction, then they follow the y-direction. This strategy enables sending margin values of processes which have a distance of one in x- and y-direction from the origin process. The diagonal communications are relevant when ghost zones exceed one. Now the stencil computation is performed for the number of the local iterations. Both, the communication and the computation are repeatedly carried out for the number of global iterations.



(a) 50 global iterations

(b) 100 global iterations

(c) 250 global iterations

(d) 500 global iterations

(e) 1000 global iterations

(f) 2500 global iterations

Figure 1: Evolved field after various numbers of global iterations. Further input parameter are set as listed. "512x512 points per field, 64 local iterations, 16 processes"

2.4 Returning results

The last part collects the results as well as the elapsed time on each process and generates the user output. In the command line, a table with the results of the time measurement appears. Additionally, a ppm file is created in the directory where the program was executed.

3 EXPERIMENTS

3.1 Local machine

To ensure the correctness of the code, experiments were executed on a local machine. Figure 1 presents the results. The local machine has four physical cores and supports hyper-threading. Thus, mpi starts not more than four processes per default. However, to check the code for inner nodes of the process grid, nine processes at least were required. A host file allows modifying the number of processes started through mpi. The program was locally tested with nine and 16 processes. The following page presents the results for 512x512 field produced by 16 processes with a ghost zone size of 64. Thus, the program start command for 500 global iterations was as follows.
mpirun –hostfile ./Hostfile ./non_Block.exe 512 64 500

3.2 Ara cluster

The performance tests were executed with 16 mpi processes. For a better comparison between computation and communication time, each process had its own node. Anyway, to determine a suitable set of parameters for testing, some trials were executed. Finally, the chosen size of the field was 4096x4096 points and the total number of iterations was 256,000. Now, the computation was executed with different combinations of local and global iterations.

4 CONCLUSIONS

4.1 Computation time

Table 2 shows the total times of all tests. It can be seen that the simulation with a ghost zone size of 64 and 4000 global iterations is the fastest version. This can be explained by taking the computation times and figure 2 into account. They became more unequal when the ghost zone size increases. An explanation might be that the ghost region size has more influence on the sub-fields of the inner nodes since they have four edges and not two as the corner nodes. Thus, the field size of an inner node is 2048x2048 while a corner node updates a 1536x1536 field when the ghost region is 512. Unfortunately, within the blocked version was used an older version of the stencil algorithm. Due to this, the computation time is not comparable to the other tests. If the newer and faster algorithm had been used, it probably would have delivered similar times as the non blocking version with the same parameters.

Table 2: Comparison of the different parameter combinations which were tested. The field size is defined by the points in each dimension. The number of local iterations is also the ghost zone size. The number of local iterations multiplied with the number of global iterations is equal to the overall iterations which are in each case 256'000. Only one test was performed with blocking communication. Note that this version has used a slower version for computing stencils. Thus, it cannot be compared directly to the non-blocking equivalent.

field size	local iter.	global iter.	blocking	total time (s)
4096x4069	512	500	non	1782
4096x4069	256	1000	non	1324
4096x4069	256	1000	yes	1621
4096x4069	128	2000	non	1276
4096x4069	64	4000	non	1258

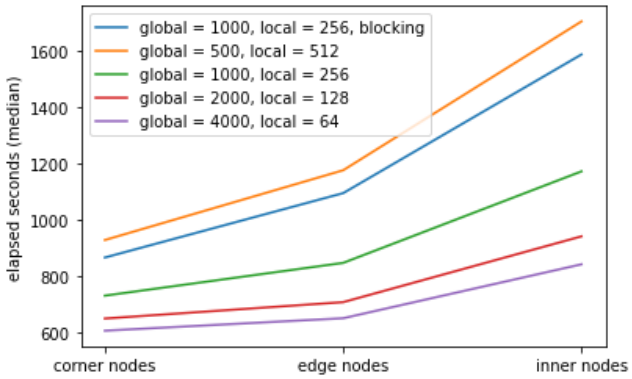


Figure 2: Computation time on the different types of nodes. This Figure presents the median time needed overall for computation. The various graphs visualize different combinations of parameters. The words 'global' and 'local' specify the corresponding number of iterations. Dividing the time by the number of iterations delivers the computation time of a single iteration. The number of iterations is equal to the number of local iterations multiplied with the global ones.

4.2 Communication time

Referring to figure 3, the communication time appears to be higher at the corners. This may be explained by the characteristics of the send and receive operations. The corner nodes complete their computation first. Then, they proceed with the next iteration. The corner nodes need the margins from their neighbours to end their communication phase. This edge nodes, again, need to finish communication with their neighbours in x-direction before they can

send the information to the corner nodes. Thus, the corner nodes have to wait until the computation of the inner nodes has finished. In summary, the corner nodes start the time measurement for communication then they wait until all data has arrived which needs time since the inner nodes will still be in the computation phase, and finally the corner nodes stop the time measurement. This means that the communication time of the edge and corner nodes includes probably a lot of idle time. Surprisingly, the blocking communication of the inner nodes is faster than the non-blocking version with same parameters. It may be that the communication is faster in the blocking version since the communication partners do not send their data concurrent while computing.

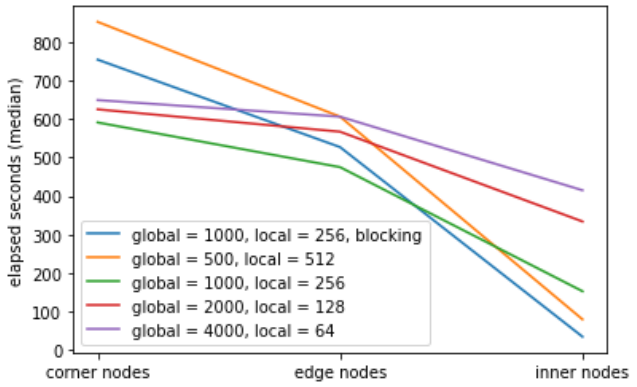


Figure 3: Communication time on the different types of nodes. The Figure looks at the overall median time required for communication. The various combinations of parameters are visualized in the same colours as in figure 2. As before, 'global' and 'local' stand for the corresponding number of iterations. The time has to be divided by the number of iterations to obtain the average communication time per single iteration. The number of iterations is equal to the number of local iterations multiplied with the global ones. Note that communication occurs only once per global iteration.

5 DISCUSSION

The tests came up with interesting results. The computation load per processor had more influence than expected to the total time. On the other hand, a final declaration is pending for faster communication in the blocking version.

Finally, suggestions for further tests are discussed. To optimize the time measurement for the communication time, inserting a MPI_barrier after each global iteration seems reasonable. Additionally, the time measurement should be saved as a CSV file or should have the form of a CSV file. It would allow evaluating the collected data much faster. Another point could be to execute the program with fewer local iterations. Applying this point may speed up the program since the fastest test was the one with the smallest ghost zone size.

Another approach to speed up the program could be realized by unequal sub-field sizes as presented in table 3. In this example, the corner processors have a larger block than the edge processors, and the inner nodes have the smallest sub-fields. When a ghost zone size is given, the algorithm could determine the block sizes that each node has to perform a similar number of computations per global iteration. Thus, the outer processors have no longer to wait for the inner nodes.

Table 3: A unequal distribution of the field. Processor zero manages the upper left block of the original $n \times n$ field.

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

REFERENCES

[1] 2021. Heat equation. In *Heat equation*. Wikipedia. https://en.wikipedia.org/wiki/Heat_equation