

HPSC SS2003

June 22, 2003

Parallel Solving of the Heat Equation with MPI

Report

Harald Röck

hroeck@cosy.sbg.ac.at

Academic Supervisor Prof. Roman Trobec

Department of Scientific Computing

University of Salzburg

1 Introduction

The solution $u(t, x)$ of the heat equation (equation 1) describes the diffusion of heat in a bar with length L as a function of time t and position x . We will calculate a solution as described in [3].

The initial temperature is described by a function $f(x)$ at each point x on the bar at time zero. Now the bar is heated by different but constant temperatures α and β at both ends.

We describe this equation and this boundary conditions mathematically as follows:

$$\frac{\partial u}{\partial t} = c \frac{\partial^2 u}{\partial x^2}, \quad 0 \leq x \leq L, \quad t \geq 0, \quad (1)$$

with initial conditions:

$$u(0, x) = f(x), \quad 0 \leq x \leq L$$

and boundary conditions:

$$u(t, 0) = \alpha, \quad u(t, L) = \beta$$

The constant c is a positive constant and depends on the material of the bar.

Equations of this type are called *parabolic*, because the solutions are evolving toward a steady state. That means at a specific time the diffusion will only changing for a very small amount.

2 Discretization

The numerical solutions of a PDE discretize the derivations by approximation with finite differences. If we discretize both, time and space, then we use a fully discrete method. Another method would be a semi-discrete method. Such a method discretize only the space dimension with a final amount of points. This yields in a system of ODEs, which can then be solved by a standard method for solving such a system. This is also called the method of lines.

We will use a fully discrete method. So we have to substitute all derivations with finite differences:

$$\frac{u_i^{k+1} - u_i^k}{\Delta t} = c \frac{u_{i+1}^k - 2u_i^k + u_{i-1}^k}{(\Delta x)^2}, \quad i = 1, \dots, n \quad (2)$$

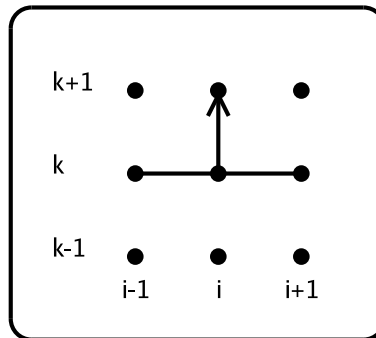


Figure 1: Stencil of the explicit method for the heat equation

We can rearrange this equation so that we get an equation for u_i^{k+1} :

$$u_i^{k+1} = u_i^k + c \frac{\Delta t}{(\Delta x)^2} (u_{i+1}^k - 2u_i^k + u_{i-1}^k), \quad i = 1, \dots, n$$

This seems to be naturally clear, because the temperature at the i th point in the next time step will be affected by the temperature of his neighbors and his own temperature.

From the initial conditions we get the values for $u_i^0 = f(x_i), i = 1, \dots, n$. This is the starting point from which we can march the approximate solution forward step by step in time. The boundary conditions providing the necessary values $u_0^k = \alpha$ and $u_{n+1} = \beta$.

This time stepping scheme is *explicit* because the approximate solution values at any given time step depend only on values that are available from the previous time step. The stencil is shown in figure 1.

The local truncation error of this scheme is $O(\Delta t) + O((\Delta x)^2)$, this means it is of first order in time and of second order in space.

This scheme is simply Euler's method applied to the semi-discrete system of ODEs derived for this problem using finite difference spatial discretization. The stability region of this method is defined by

$$\Delta t \leq \frac{(\Delta x)^2}{2c}.$$

So the time step size depends on Δx , if Δx is small Δt becomes very small. Therefore exists various implicit methods which would be unconditional stable. But we will use this explicit method, as we will also parallelize this.

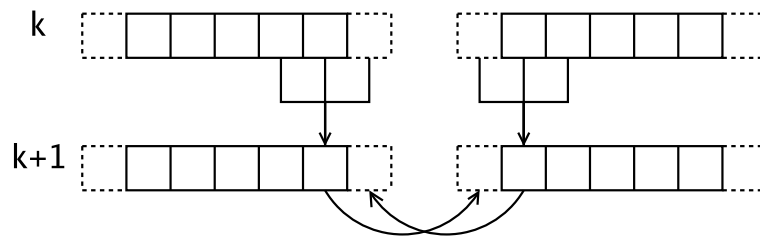


Figure 2: calculating and changing the new values

3 Implementation

As mentioned before and the title says we are looking for a parallel method to compute the heat equation. We implemented the method in C++ with *mpich* [1, 2].

The algorithm is relative simple. We have only to calculate

$$u_i^{k+1} = u_i^k + c \frac{\Delta t}{(\Delta x)^2} (u_{i+1}^k - 2u_i^k + u_{i-1}^k)$$

for each node.

The nodes are uniformly distributed to the available processors at the beginning. So each processor is responsible to a disjunctive set of nodes. As for each node the value for the next time step is calculated with the values of its neighbors, there must be some communication between the different processors. Figure 2 shows how this will work. Each processors has two additional stores where the values of the neighbors will be saved. This values have to be updated after each step.

As this equation is parabolic the solution converges to a stable state. So we have to include a convergence test. If the result doesn't change or only for a small amount the iteration will stop. To solve this we need some global communication. Each processor send the norm of the difference between the previous and the current solution to all others and sum all received norms. This will be done with `MPI_Allreduce` at each calculation step.

At the end of our calculation the results will gathered in to processor zero, and printed to `stdout`.

4 Results

I tested the program at a Linux Cluster with 16 1GHz processors, because one processor was broken there were only 15 processors available. First I run it on one

	1	2	4	8	15
init	0.0385	0.4270	1.0514	2.2763	4.2521
50000	18.646	9.852	4.743	2.912	31.237
100000	37.273	19.207	12.192	5.166	14.137
200000	73.365	37.557	21.355	12.478	11.567
500000	183.183	92.201	50.175	26.685	20.650
1000000	368.411	186.146	99.789	50.324	30.865

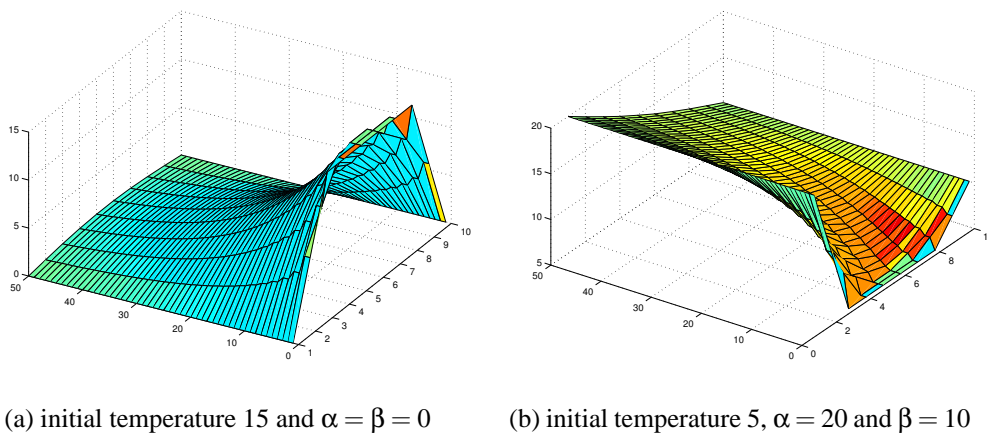
Table 1: results of the timing on a 15×1 GHz Linux cluster

Figure 3: Two results plotted after 50 steps

processor then with two, four, eight and finally with 15. The timing measurements can be found in table 1.

The init line on the table shows the time that is used to initialize all processors. In the first column you can find the problem size and on the following the time that was needed.

From the table follows that the speedup is nearly linear if the problem size is big. For a smaller problem size the communication time is dominant in comparison with the computation time. That means if there are too many processors involved they spend more time to transfer the data to their neighbors as to calculate the next step. We see also that for a problem size of 50 000 eight processors would be optimal and 15 would be to much.

At figure 3 you can find two plotted results. The solutions are as expected they converge to a linear interpolant between α and β .

References

- [1] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, September 1996. 3
- [2] William D. Gropp and Ewing Lusk. *User's Guide for mpich, a Portable Implementation of MPI*. Mathematics and Computer Science Division, Argonne National Laboratory, 1996. ANL-96/6. 3
- [3] Michael T. Heath. *Scientific Computing, An Introductory Survey*, volume 2nd Edition. 2002. 1