Performance enhancement using MPI in a simulation of heat diffusion

Gonçalo Lourenço
nº55780
gm.lourenco@campus.fct.unl.pt

Joana Faria
nº55754
js.faria@campus.fct.unl.pt

I. Introduction

This assignment aims to optimize a base code, written in `C`, that computes a simulation for heat diffusion. This time we intend to parallelize the computation using the Message Passing Interface (MPI).

To find the best performance we will explore different approaches, namely analyzing different patterns of communication and trying to overlap the communication with the computation.

We have available, for this assignment, two nodes, each one with 16 true CPUs. Given this information, we can have a maximum of 32 processes.

II. Results

We experiment with many versions of a parallel program using MPI, these versions are explained and analyzed below.

To establish a baseline for comparison we start by analyzing the execution time of the sequential version and we obtain an average of 153.979 seconds, with the following parameters:

```
// Width of the area
const int nx = 200;
// Height of the area
const int ny = 200;
// Diffusion constant
const float a = 0.5;
// h=dx=dy  grid spacing
const float h = 0.005;
// Number of time steps to simulate
const int numSteps = 100000;
// How frequently to write output image
const int outputEvery = 100000;
```

Unless otherwise stated, these will be the parameters used for determining the execution times.

III. Discussion

A. V1 version — Synchronous Communication

In version V1 we start with a simple parallel version where we distribute the lines of the original matrix by the number of processes and perform synchronous communication between neighbors processes to obtain the border necessary for the computation.

For the communication necessary to write the output we use the `MPI_Send` and `MPI_Recv` methods, where all process sends their results to process 0, which aggregates all the results and writes the output file.

In Figure 1 we can see the execution time for V1, comparing the performance of this version for different numbers of processes. From this graphic we can see the bigger improvement until 16 processes, these processes run on the same machine. For the sake of completeness, we test with 17 processes and we can observe a slight worsening, this is because only one process is running on a different machine, adding communication overhead. Is due to this communication overhead that the improvements aren't bigger when we use 32 or 48 processes.

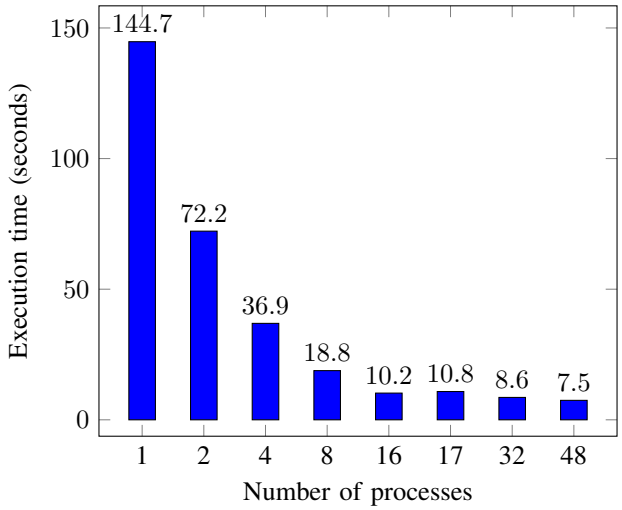

Fig. 1: Execution time of V1 varying the number of processes

B. V2 — Gather

In this version, we modify version V1 to use the method `MPI_Gather` for the phase of writing the output. The results are similar to version V1, and are presented in Figure 2.

C. V3 — One node for output

In V3 we assign node 0 the function of joining the output of all the other nodes, performing no computation concerning the heat equation. This version was intended as a first step to overlap computation with communication in situations where output is stored more frequently. In this version, we keep synchronous communication.

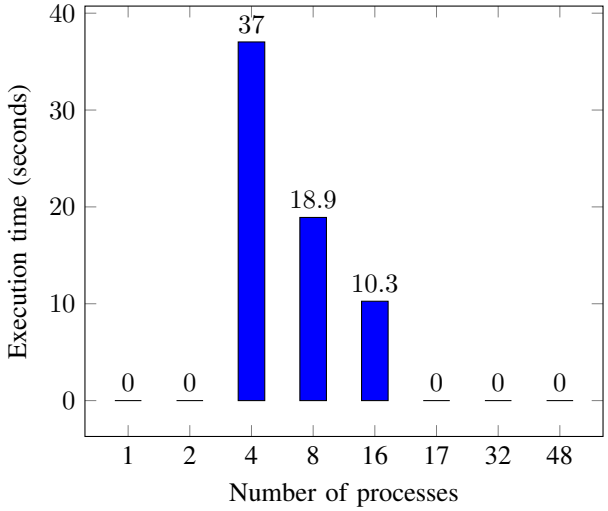

Fig. 2: Execution time of V2 varying the number of processes

IV. Compilation and Execution Instructions

Our best version, V2, can be compiled and executed with the following command, from inside the folder `/proj1`:

```
nvcc -o v2 v2.cu && ./v2
```

This will execute ten iterations to measure the average execution times and the results of the heat equation will be saved in the folder `images/v2`.