13X007: Assignment #3

Due on 08.11.2023

Parallelism

UNIVERSITÉ
DE GENÈVE

CHRISTOFOROU Anthony

Abstract

This report investigates the parallelization of a 2D heat equation solver using the Message Passing Interface (MPI). The Finite Difference Method with the Forward-Time Centered-Space (FTCS) scheme was applied across different domain sizes. Performance was analyzed by timing executions across a range of processor counts. The results demonstrate the power of parallel computing to significantly reduce computation time, particularly for large domain sizes, while also highlighting the trade-offs and challenges associated with parallelization, such as communication overhead and load balancing. The study provides insights into optimizing parallel computation strategies, contributing to the broader field of computational science.

Contents

1 Introduction

The 2D heat equation is a fundamental partial differential equation that describes heat conduction. It represents how heat diffuses through a given area over time. This report details the application of parallel computing techniques, specifically MPI, to solve the equation more efficiently. It discusses the methods used, the results obtained, and the insights gained from the process, highlighting the importance of parallelism in computational science.

1.1 Background

The heat equation is central to thermal physics and engineering, providing insight into the thermal behavior of materials and systems. Its solutions are crucial for designing heat management systems in various engineering disciplines, from electronics to aerospace.

1.2 Significance of Parallel Computing

Traditional serial computation methods become impractical as the complexity and size of the domain increase. Parallel computing addresses this by splitting the task among multiple processors, leading to substantial reductions in computation time and enabling the solution of larger and more complex problems.

2 Methodology

This section elaborates on the computational strategy adopted for solving the 2D heat equation. It details the numerical method employed and the approach to parallelizing the computation.

2.1 Finite Difference Method (FDM)

The core numerical method used is the Finite Difference Method (FDM), which approximates the continuous differential equations of the heat equation with a discretized grid. The grid spacing (Δx and Δy) and the time step size (Δt) were chosen in accordance with the stability criteria of the Forward-Time Central-Space (FTCS) scheme. This approach involves updating each grid point's temperature based on its current temperature and the temperatures of its immediate neighbors.

In the implementation, a two-dimensional grid of size `rows` $\times$ `cols` is used, represented by `kt::vector2D<double>`. The update formula for each interior grid point $T_{i,j}$ at a new time step is given by:

```
new_grid[i][j] = weightx * (grid[i - 1][j] + grid[i + 1][j] + grid[i][j] * diagx) +
    weighty * (grid[i][j - 1] + grid[i][j + 1] + grid[i][j] * diagy);
```

where `weightx`, `weighty`, `diagx`, and `diagy` are coefficients derived from the grid spacing and time step size, and `new_grid` stores the temperatures for the next time step.

Boundary conditions are set to maintain a constant temperature at the edges of the domain, mimicking a typical physical scenario. The temperature at the boundaries is set to 1.0, representing a fixed heat source:

```
        if (rank == 0) {
            for (int j = 0; j < cols; ++j) {
```

```
                grid[0][j] = 1.0; // Top edge
            }
        }
```

... [similar code for other edges] ...

2.2 Parallelization Strategy

To efficiently handle large-scale computations, the domain was decomposed into smaller subdomains, each assigned to a different processor. The MPI (Message Passing Interface) framework was used to facilitate communication between these processors. This approach allows for efficient data exchange necessary for calculating boundary conditions and iterative updates.

The domain decomposition is realized by dividing the grid into horizontal slices, with each MPI process handling one slice. Non-blocking communication operations (`MPI_Isend` and `MPI_Irecv`) are used to exchange boundary row data between adjacent processes:

```
if (rank > 0) {
    MPI_Isend(&local_grid[0][0], cols, MPI_DOUBLE, rank - 1, 0,
        MPI_COMM_WORLD, &top_request);
    MPI_Irecv(top_buffer.data(), cols, MPI_DOUBLE, rank - 1, 1,
        MPI_COMM_WORLD, &top_request);
}
... [Similar code for sending and receiving the bottom buffer] ...
```

This ensures that each process has the necessary data to update the temperature values at its boundaries, thereby maintaining the continuity of the heat distribution across the entire domain.

2.3 Computational Setup

The computations were carried out on a high-performance computing setup, consisting of a Ryzen 9 3900X CPU 12 Core Processor. This setup provided the computational power required to handle the intensive calculations involved in the FDM and parallel processing.

3 Results

The following subsections present the empirical data gathered from the simulations, highlighting the performance improvements and providing a detailed analysis of the results.

3.1 Execution Time Measurements

Execution times were recorded for domain sizes 64x64, 128x128, and 256x256 with varying processor counts. A general reduction in computation time was observed with the increase in processor count.

3.2 Graphical Representation

The data were plotted using line graphs, which clearly show the trend of decreasing execution time with an increasing number of processors. These graphs provide an immediate visual representation of the performance gains achieved through parallelization. This was made with a ver large number of iteration (10^5) as less than this would have been too fast therefore the communication between the processors would have been too important, because there would have been too many communications for a small amount of computation.

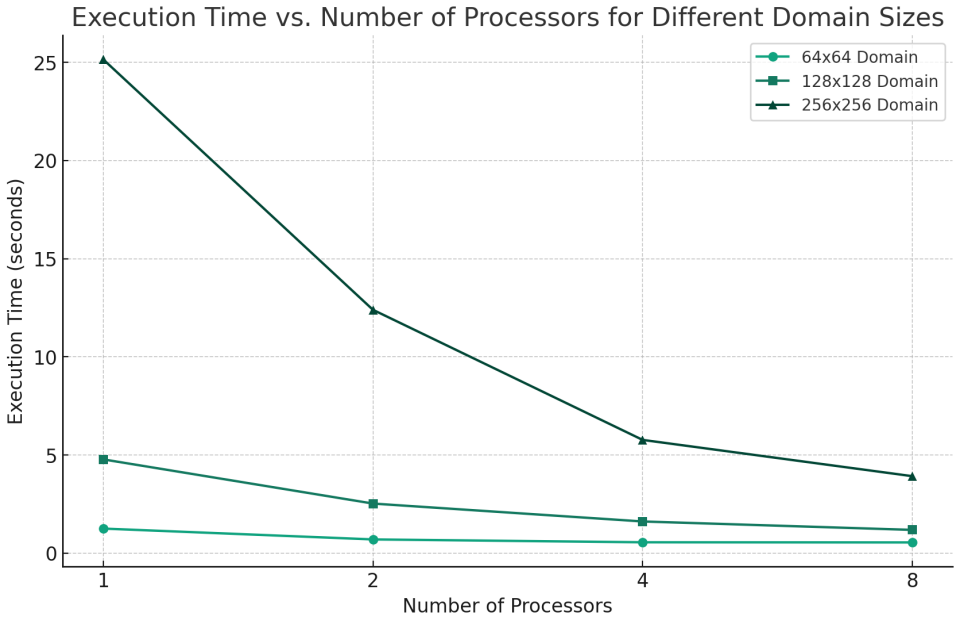

Figure 1: Execution Time vs. Number of Processors for Different Domain Sizes

3.3 Analysis of Results

The results indicate a trend consistent with Amdahl's Law; the speedup due to parallelization is substantial but tends to level off as the number of processors increases. This is particularly evident in the case of smaller domain sizes, where the communication overhead can offset the computational speedup.

For example, the solver's execution time for a 64x64 domain decreased from 1.25073 seconds with one processor to 0.545042 seconds with eight processors. Similarly, for a 128x128 domain, the time reduced from 4.77483 seconds to 1.18351 seconds, and for a 128x128 domain, from 25.1451 seconds to 3.91991 seconds with the same increase in processors.

We'll talk more about the implications of these results in the discussion section.

3.4 Case Study: 256x256 Grid

A detailed case study of the 256x256 grid provides a clear example of the solver's functionality.

3.4.1 Initial Heat Distribution

The initial grid represents the temperature distribution at the beginning of the simulation, where boundary conditions are set to 1 (representing a fixed temperature) and the interior points are

initialized to 0.

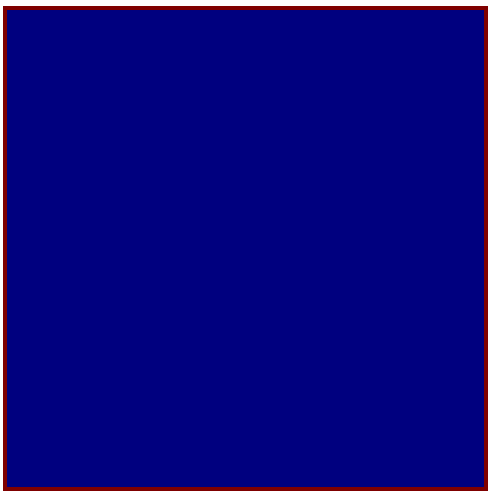

Figure 2: Initial temperature distribution on an 256x256 grid.

3.4.2 Evolution of Heat Distribution

Here you'll be able to see the evolution with the number of iteration of the heat dispertion. The heat has diffused from the boundaries into the interior according to the 2D heat equation.

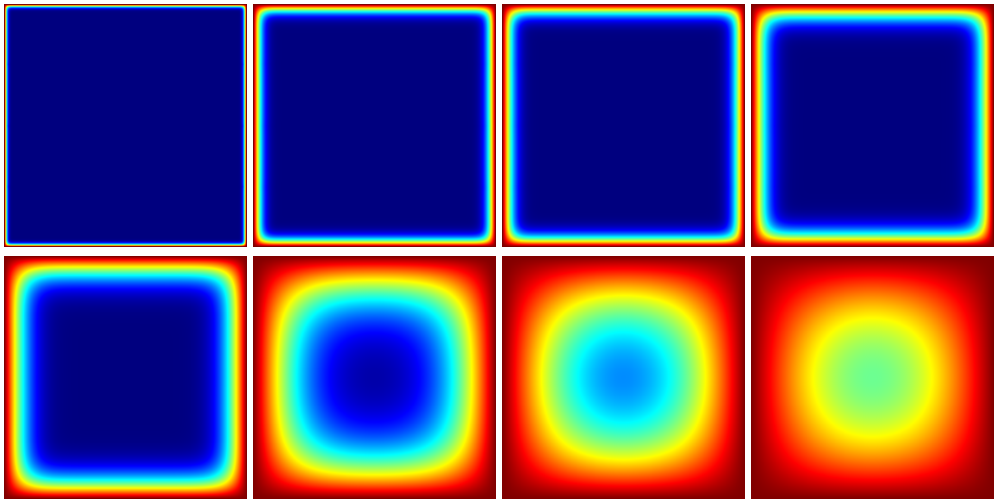

Figure 3: Temperature evolution on an 256x256 grid.

4 Discussion

The discussion assesses the results' implications, evaluating the benefits of parallelization and its constraints. The insights gained from this study illustrate the balance between computational speedup and parallel overhead, shaping the strategies for future parallel computing applications.

4.1 Performance Gains

The performance analysis revealed a marked improvement in computation times when using parallel processing. For instance, the solver's execution time for a 256x256 domain showed a reduction from 25.1451 seconds with a single processor to just 3.91991 seconds with eight processors, a speedup of approximately 6.42 times. This improvement emphasizes the critical advantage of parallel computing in handling large-scale problems, where the computational workload is significantly heavy. The ability to distribute this workload across multiple processors enables the handling of complex simulations that would be otherwise unfeasible in a reasonable timeframe with serial computing.

4.2 Challenges in Parallelization

However, parallelization introduces its own set of challenges, most notably load balancing and communication overhead. Load balancing refers to the equitable distribution of workload across all processors, preventing scenarios where some processors are idle while others are overloaded. In our simulations, ensuring a uniform distribution of the computational grid to processors was imperative for achieving optimal performance.

Communication overhead became increasingly pronounced with the addition of more processors. This is because each processor must exchange boundary information with its neighbors at each iteration, which can become a bottleneck, especially for small domain sizes where the ratio of computation to communication is low. For example, when operating on a 64x64 grid, increasing the number of processors from one to eight only halved the execution time, indicating significant communication costs.

To mitigate these challenges, we adopted strategies such as refining the granularity of domain decomposition and employing non-blocking communication to overlap computation with data exchange, thus reducing idle time.

4.3 Implications for Scalability

The scalability of our parallel solution was not linear and highlighted the nuanced nature of Amdahl's Law in practice. The law posits that the speedup of a parallel program is limited by the time needed for the sequential fraction of the task. In our context, despite the high parallelizability of the heat distribution problem, the speedup plateaued with an increasing number of processors, indicating the presence of a non-negligible sequential component in our algorithm.

The efficiency of the MPI implementation also played a significant role in scalability. The use of advanced MPI features, such as derived data types and collective communication, proved beneficial for reducing the complexity and duration of inter-process communication.

The computing system's architecture, specifically the interconnects between processors, determined the performance ceiling for our parallel application. A high-performance computing cluster with low-latency networking would likely exhibit different scalability characteristics compared to our Ryzen-based setup, potentially allowing for better performance with a higher number of processors.

Overall, the trade-offs highlighted in this study between computational speedup and parallel overhead underscore the importance of a balanced approach to parallelization, one that considers the specific characteristics of the problem, the efficiency of the parallel computing framework, and the underlying hardware architecture.

5 Compilation and Execution

5.1 Build and Compilation

The project uses a Makefile for compilation to simplify the build process. To compile the parallelized heat equation solver, navigate to the project directory and run the following command:

```
make
```

This produces an executable named `heat` in the build directory.

5.2 Running the Solver

The solver can be executed using MPI with the following command pattern:

```
mpirun -np [number_of_processors] ./build/heat [domain_size] [number_of_iterations]
```

For example, to run the solver on a 128x128 domain using 4 processors, the command would be:

```
mpirun -np 4 ./build/heat 128 1500
```

The terminal output for this command would resemble:

```
Simulation took 1.61275 seconds.
```

The results will be output to a bmp image named `T_[number_of_iterations].bmp`, containing the final temperature distributions.

6 Conclusion

The report concludes with an overview of the findings, emphasizing the significant performance gains achieved through parallelization. The importance of optimizing both the computational aspects and the communication strategies is highlighted as key to further performance improvements.