ENPH 479 High Performance Computational Physics Parallelized Computing

Jane Cohen[1, *]

[1] *Department of Physics, Engineering Physics and Astronomy, Queen's University, Kingston, ON K7L 3N6, Canada*
In association with the Kingston Centre for Advanced Computing

(Dated: March 17, 2024)

This study explores the application of parallel programming, specifically using the Message Passing Interface (MPI), to solve the 2D heat equation. The initial conditions set a region of a steel plate at a high temperature, with the remainder at a cooler temperature, and the simulation tracks the heat diffusion across the plate. The work focuses on dividing the computational domain of the plate into discretized sections, enabling parallel processes to solve the heat distribution for each section over time. This parallelization is made possible through the use of MPI, which distributes portions of the grid to different processes and manages the necessary communication between these processes, especially for the boundary conditions between adjacent sections. The study also investigates the performance scalability of the parallelized solution, analyzing the speedup and efficiency against the number of processes and nodes used. Results from the scaling experiments demonstrate the non-linear relationship between the number of processes and the speedup. This result highlights the impact of communication time and serial portions of the code on parallel efficiency. This is particularly evident in a two node setup, where communication between nodes introduces additional latency. Despite the challenges in communication with increased computational scale, this study affirms the efficiency of parallel computing in addressing complex physical models.

I. BACKGROUND

A. Parallel Programming

Parallel programming and the use of SLURM (Simple Linux Utility for Resource Management) are critical for modern high-performance computing. Parallel programming, facilitated by the Message Passing Interface (MPI), allows for simultaneous computations across multiple CPUs, significantly enhancing the efficiency and capability of computational tasks. SLURM acts as the job scheduler on the cluster, managing and queueing the submission of jobs. It requires users to create job scripts that define the necessary resources, such as the number of nodes and processes, as well as other parameters like simulation time and output directives. This setup enables detailed control over the execution of MPI scripts, ensuring that computational jobs are efficiently allocated and managed.

B. 2D Heat Equation

The heat equation predicts how heat will diffuse throughout an area over time. The 2D heat equation specifically is the partial differential equation shown in Equation 1.

$$\frac{\partial U}{\partial t} = D\left(\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial x^y}\right) \tag{1}$$

Using the forward Euler scheme, the equation can be written in terms of finite differences, allowing us to solve for $U(x, y)$ over time. In two-dimensions, the domain over which the temperature is solved for is discretized into smaller 2D sections or points. Each point on the grid has its own unique temperature and depends on the previous temperature for that point and its neighbouring points.

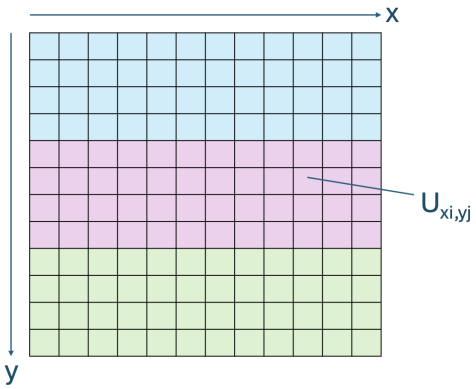

Figure 1. The domain of a plate discretized and split into strips. Each point on the grid has its own unique temperature. The domain is divided horizontally, resulting in each strip including all x points and a specific range of y points.

C. Solving the 2D Heat Equation Using Parallel Computing

To solve the discretized 2D heat equation using parallel processing, the domain is split into a grid with a defined number of points. The parallelization is done in one dimension so that each process solves for the temperature values of one "strip" at each time step. This division is shown in Figure 1.

To initialize the temperature of each point, the MPI framework is used to scatter each portion of the grid to a different process. The initial conditions for a specific scenario are handled by each process in parallel. The sim-

* 18jpcw@queensu.ca

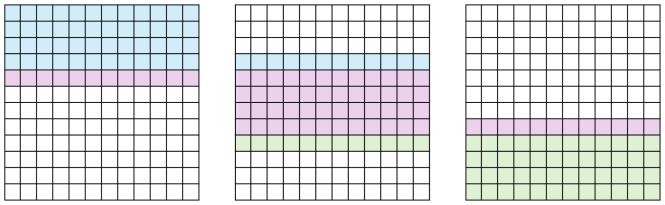

Figure 2. The boundary communication between strips of the plate domain. Each strip has buffer rows that facilitate the exchange. The top and bottom strips only exchange one row as shown.

ulation then proceeds in time steps, with each process updating the temperature values of its assigned grid portion based on the heat diffusion equation. Boundary conditions are applied to maintain the edges of the plate at the cooler temperature.

Communication between strips is necessary at each time step to accurately simulate heat diffusion across the entire plate, especially where the computational domain is divided among processes. Because each process is responsible for a portion of the grid, it must exchange boundary row information with adjacent processes in the form of 2D arrays to ensure continuity in the simulation. Specifically, a process sends its top most row to the process directly above (with a lower rank) and its bottom most row to the process directly below (with a higher rank). In addition, it receives a new top row from the process below and a new bottom row from the process above. This exchange is facilitated by MPI's send and receive operations. The processes with rank zero and the process with the highest rank are exempt from exchanging their top and bottom row respectively due to their position in the grid. These general and special cases can be seen in Figure 2 where the boundary communication is displayed visually. After the exchange, each process updates its local grid portion.

After each time step, the updated temperature grids from all processes are gathered back to the root process into the complete temperature grid. Based on user preferences, the script can also save snapshots of the plate at chosen timestamps and/or save the data as a text file for post-processing.

D. Test Case - Steel Plate

To observe the results of solving the 2D heat equation using the parallelized script described above, a test case of a 20.48 mm × 20.48 mm steel plate split into a grid of 1024×1024 points was used. Boundary condition of $U = 300K$ were implemented. For the rest of the grid, the initial conditions given by Equation 2 were used where r is the distance from the centre of the square domain.

$$U_0 = \begin{cases} 2000cos^4(4r) \text{ K} & \text{if r } < 5.12 \\ 300 \text{ K} & \text{if r } \geq 5.12 \end{cases} \tag{2}$$

The temperature distribution is initialized using these initial conditions such that circular regions at the center of the plate starts at a high temperature, with the rest of the plate set to a cooler temperature. Based on the dimensions and thermal diffusivity of the surface the heat distribution is then calculated over time using the parallelized process.

E. Scaling Experiments

The efficiency of a program can be determined by analyzing the time it takes to run the serial program compared to the parallel program. "Speedup" refers to the ratio of the serial and parallel runtime and is a helpful tool for comparing the efficacy of different number of processes. This ratio is given by Equation 3, where T_1 is the runtime of the serial program and T_p is the runtime for p processes.

$$S_p = \frac{T_1}{T_p} \tag{3}$$

The efficiency, another way of assessing the program, is found by taking the ratio of the speedup to the number of processes as seen in Equation 4.

$$E_p = \frac{S_p}{p} \tag{4}$$

We'd expect the runtime to half each time a process is added, resulting in a linear relationship between the number of processes and speedup. However, this is not the case in reality due to various factors such as portions of the code running in serial. In this case, considering the loop that runs in parallel for each time step, 0.23% of the code is serial. Amdahl's Law defines the consequence of having portions of the program run in serial. Given by Equation 5, this equation gives the maximum speed up that you can expect for a program running using a specific number of processes and percent of the program in serial. For this law, F_s is the fraction of the script that is serial and S_{max} represents the best speedup for that program and conditions.

$$S_{max} = [F_s + \frac{1 - F_s}{p}]^{-1} \leq \frac{1}{F_s} \tag{5}$$

The SLURM job script allows the user to modify the number of nodes and processes over which the computation will take place. To evaluate the speedup and efficiency of the program, various numbers of processes and nodes will be used and compared to Amdahl's Law.

II. RESULTS

A. Heat Distribution Across a Steel Plate

To determine if the parallel code correctly solved the heat equation, the evolution of the heat distribution across the plate was graphed. As seen in 3, the images were captured at different timestamps during the heat diffusion simulation. In the first frame at 0.0 ms, we see the initial condition with concentric rings of high temperature in the center

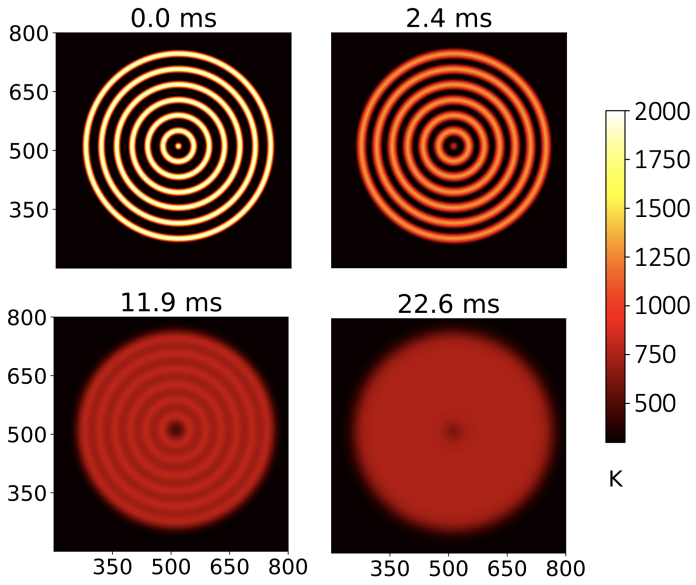

Figure 3. The simulation of heat diffusion across a 2D steel plate. The simulation was run with one node and four processes for 1000 time steps. Initially, the heat is distributed in defined circles based on initial conditions. Over time, the heat diffuses across the plate and the concentric circles begin to smooth out.

of the domain. The bright rings suggest high temperatures in the center, which fade as they radiate outward. As time progresses to 2.4 ms, the temperature starts to disperse across the plate, but the central pattern remains distinct. By 11.9 ms, the rings become more diffused, with the temperature gradient between them less pronounced. This diffusion continues until 22.6 ms, where we see the pattern has almost entirely faded into a uniform temperature distribution, indicating that the heat has spread throughout the entire domain and is approaching a state of equilibrium. The results of this test demonstrate the ability of the code to correctly represent the results of the physical problem being solved.

B. Program Scaling

Scaling experiments were conducted to determine how the performance of the parallel heat diffusion program scales with the number of nodes and processes used. For timing, all I/O and initialization was omitted including applying the initial conditions to the plate, plotting and saving data. For a serial script, solving the 2D heat equation for 1000 time steps took 19.29 s. For this same number of time steps, 1, 2, 4, 8 processes with a single node, and for 4, 8, 16 processes split over 2 nodes were timed. For each configuration, each individual process had its own runtime. While the difference in runtimes for one configuration only varied slightly between processes, it is interesting to note that they were not identical. Each configuration was tested three times and the average runtime was found. The results of these timing trials can be seen in Table I.

Figure 4 illustrates the speedup achieved through parallelization of the code as the runtime decreases with added

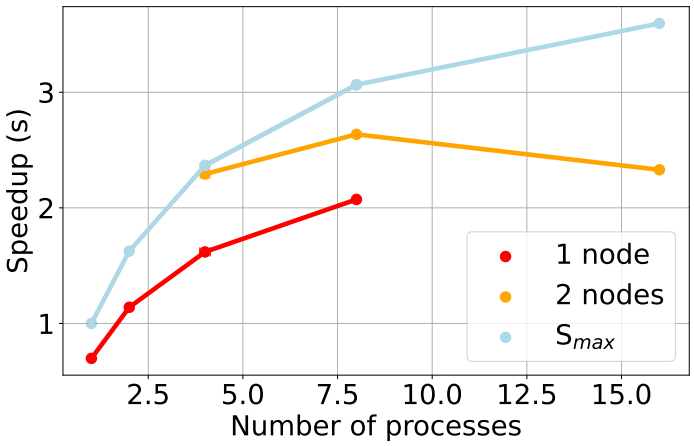

Figure 4. The speedup of the parallel program for different number of processes and nodes. For one node, the speed up increases as the number of processes increases but starts to plateau. For two nodes, a decline is seen in speedup eventually. The results of the maximum speedup theoretically possible is also displayed.

processes. This relationship is non-linear and the speedup curve starts to flatten as the number of processes increases. This phenomena is caused by bottlenecks from the serial portions of the code and communication between processes that do not allow for linear scaling. In addition, the two-node scenario not only shows a plateau, but slight decrease in speedup. This is likely due to inter-node communication that does not decrease as processes are added. This indicates that while parallelization enhances performance, the benefits can be impacted by the communication between nodes, especially in multi-node configurations.
The maximum speedup possible for each configuration was calculated using Amdahl's Law given by Equation 5. As seen in Figure 4, the actually speedup values never surpass this theoretical maximum.

In addition to speedup, the efficiency of the node/process configurations were found and are shown in Figure 5. The curves reveal that parallelization efficiency decreases as more processes are added, especially in the two-node setup. This trend is due to the fact that communication between nodes is slower than communication between processes in the same node.

Lastly, there are different technical aspects of the code that impacted its performance that are important to recognize. Initially, communication for sending and receiv-

Simulation Times			
Nodes	Processes	Runtime (s)	S_p
1	1	27.65	0.698
1	2	16.92	1.14
1	4	11.92	1.62
1	8	9.31	2.07
2	4	8.42	2.29
2	8	7.32	2.64
2	16	8.28	2.33

Table I. The runtimes for the parallel program for different number of nodes and processes and the speed up for each configuration.

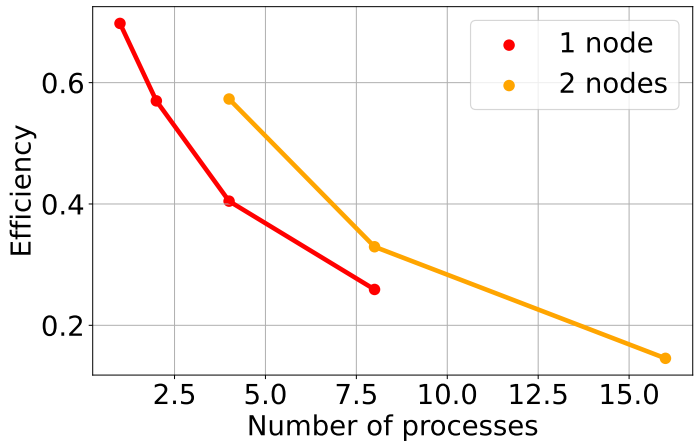

Figure 5. The efficiency of the program for different number of nodes and processes. The efficiency declines as the number of processes increases.

ing boundary conditions between processes was not being done using numpy array specific functions. Despite this, the code was able to work for a small number time steps and a small domain size. However, once the amount of data being processed increased, the script stopped working. This occurrence emphasizes the importance of data types and their corresponding functions when using MPI. In addition, the script initially had no tags being used to identify the arrays being sent between processes. For the small scale problem, this worked. However, if the project were to be scaled up or the complexity of the communication increased, the absence of tags could result in major communication problems for the program.

III. CONCLUSION

In conclusion, using parallel computing to solve the 2D heat equation proved to be an effective way in decreasing runtime and overall performance. By splitting a steel plate into discretized horizontal strips, parallel processing was used to solve the heat equation for portions of the plate simultaneously. The analysis of the speedup and efficiency of the code demonstrates the potential and limitations associated with parallelization. The scaling experiments revealed a non-linear speedup as more processes were added with an eventual plateau due to communication between processes and portions of serial computation. This is particularly evident when a two-node setup was used, where inter-node communication introduced additional latency. In addition, the results obtained from the heat distribution simulation confirmed the effectiveness of parallel computing in solving complex physical models. Therefore, despite the challenges in communication and data management that arise with increased scale, parallel computing was shown to be a promising method for solving complex computational physics problems.