Solving the 2D Heat Equation with Parallel Computing

Andrew Hayman[1, *]

[1] *Department of Physics, Engineering Physics and Astronomy, Queen's University, Kingston, ON K7L 3N6, Canada*

(Dated: March 16, 2022)

The heat equation is a famous partial differential equation that describes the evolution of heat over time in a system. It is a simple equation to solve numerically through space and time discretization and is an embarrassingly parallel problem. This makes it an excellent example for demonstrating the implementation and scalability of parallel computing. Here, we solve the 2D heat equation with parallel computing using MPI. We plot the results of one simulation of the 2D heat equation and show the expected diffusive behaviour. Then, we demonstrate the scalability of the parallelized program for different parts of the computation on one and two nodes on the Center for Advanced Computing cluster. We show that the main parallel computation speedup excluding initialization and communication scales linearly with the number of worker processes. The communication time and other serial tasks however, are shown to be bottlenecks that limit the parallelization of the program.

I. INTRODUCTION

The heat equation governs the rate of change of heat in a system with a set of partial differential equations. It has broad applications in areas such as chemical processes, random walks, and financial mathematics. Meanwhile, many computational problems are infeasible to run on single computer processes due to memory and computational requirements. As such, it is common to parallelize workloads across many processes using methods such as MPI or OpenMP.

Here, we investigate the scalability of parallel computing with MPI for solving the 2D heat equation. We start by demonstrating the results of one simulation to show the expected diffusive behaviour from the heat equation. Then, we show a scability analysis for the number of processes against working time for a single node and then two nodes on the CAC (Center for Advanced Computing) cluster.

II. THEORY AND EQUATIONS

A. The Heat Equation

The heat equation in 1D dimension is described by

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

where D is the thermal diffusivity. This is trivial to extend to multiple dimensions, and in this paper, we consider the extension to two dimensions where

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

To solve the heat equation numerically, we can discretize it in space and time with

$$\frac{u_{i,j}^{(n+1)} - u_{i,j}^{(n)}}{\Delta t} = D\left(\frac{u_{i+1,j}^{(n)} - 2u_{i,j}^{(n)} + u_{i-1,j}^{(n)}}{(\Delta x)^2} + \frac{u_{i,j+1}^{(n)} - 2u_{i,j}^{(n)} + u_{i,j-1}^{(n)}}{(\Delta y)^2}\right). \tag{3}$$

Appropriate boundary conditions and initial conditions must be applied. Finally, a time step of

$$\Delta t_{\max} = \frac{\Delta x^2 \Delta y^2}{2D(\Delta x^2 + \Delta y^2)} \tag{4}$$

gives the largest time step value before instability issues. We use this time step for our simulations.

B. Parallel Computing and MPI

MPI (message passing interface) is a popular method for parallel computing where each process works with its own memory and has the capability of passing messages between processes. Another popular method is OpenMP (open multiprocessing) which is different in that all processes share the same memory block, requiring no messaging between processes. MPI is particularly effective for running large simulations because it allows the usage of multiple distinct nodes over a network in a cluster environment as shown in Fig. 1. The main bottleneck is the communication time between nodes.

Here, we use the mpi4py package to implement MPI in Python for solving the 2D heat equation. To distribute the workload across multiple nodes, we divide the space discretized grid vertically over the number of worker processes, as shown in Fig. 2. Each grid requires a column of information from adjacent grids to solve for the next time step due to Eq. 3. This information is passed through MPI.

While most grid sub-divisions require communication from both adjacent grids such as the yellow or red grids from Fig. 2, the outer two only require communication from one adjacent grid such as the green and blue grids from

* 18arth@queensu.ca

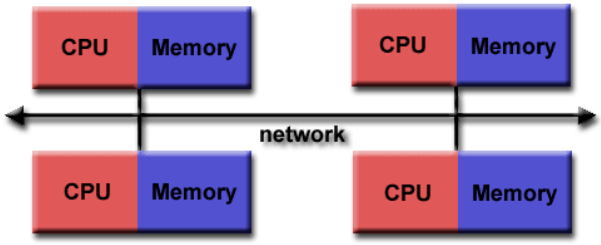


Figure 1. Diagram of MPI functionality, allowing communication between distinct nodes on a network. Image obtained from https://hpc-tutorials.llnl.gov/mpi/what_is_mpi/.

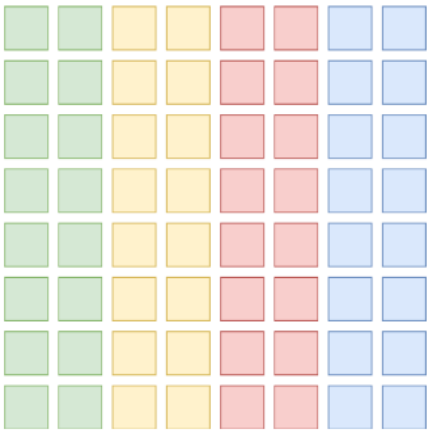


Figure 2. Example of dividing the 2D heat equation grid over 4 worker nodes. Messages must be passed between nodes containing the edge points of each colored block due to Eq. 3. For example, the yellow grid requires information from the right-most green column and the left-most red column to find the next time-step. Image obtained from class notes.

Fig. 2. This generates two special cases. In our program, both the initialization and calculation are parallelized with each process only having access to its own respective grid space.

III. RESULTS

A. 2D Heat Equation Problem

We consider the case of a grid where $0 \leq x, y \leq 20.48$ in mm. The initial condition of the system is a set of hot ripples surrounded by a cold region described by

$$\begin{cases} U_0 = 2000\cos^4(4r)\ K & \text{for } r < 5.12 \\ U_0 = 300\ K & \text{for } r \geq 5.12 \end{cases} \tag{5}$$

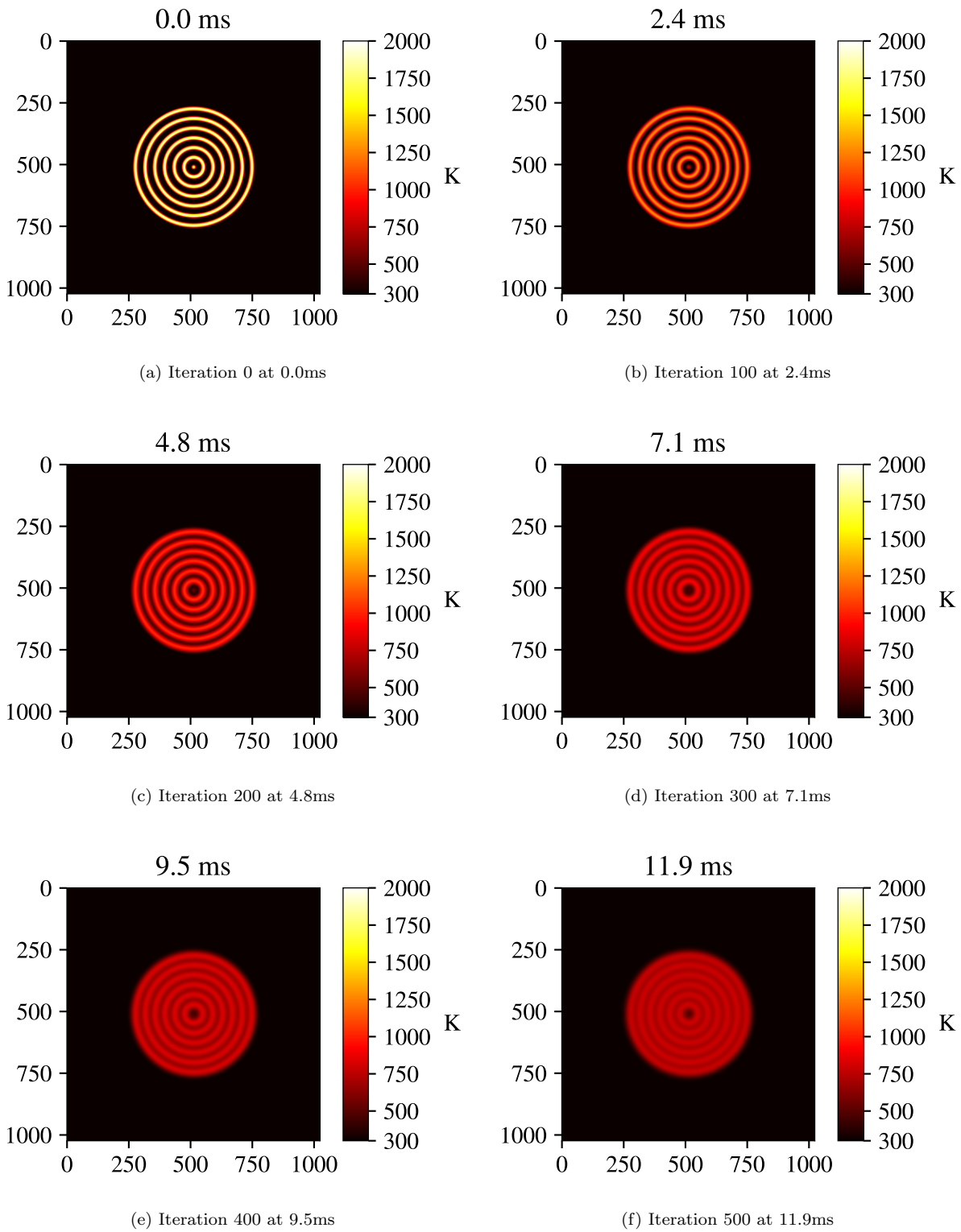
where r is the distance from the center of the grid. We apply boundary conditions of $U = 300$ K. In our program, we set this boundary condition implicitly by not updating the outermost points of the full grid. Finally, we use a thermal diffusivity constant of $D = 4.2$ mm^2/s and a grid size of 1024x1024 points.

Simulating this case generates the plots shown in Fig. 3. As expected, we observe the ripples washing out and cooling down over time.

B. Timing and Scalability

We examined the scalability of MPI by using different numbers of processes on a single node and on two nodes on the CAC cluster. The timing results are shown in Fig. 4 with the parallel computation time, the communication time, the full computational time (i.e. including parallel initialization, communication and parallel computation), and the full program time. As seen from the log-log plots, the parallel computation speedup scales linearly with the number of processes. The single and two node cases show the same time within 0.37s for a given number of processors, indicating good quality communication between different nodes in the cluster. The communication clearly demonstrates a bottleneck across jobs. However, the communication time was inconsistent between runs, and was sometimes negligible. Overall, this demonstrates that parallel computation can scale linearly, but is ultimately limited by other serial tasks and the communication bottleneck for MPI.

Note that only two nodes were available on the cluster, so the scalability to four nodes was not investigated. Also note that the maximum number of processors per node at the time of testing was only eight. As such, we have provided the most comprehensive testing possible for the given hardware.



(a) Iteration 0 at 0.0ms

(b) Iteration 100 at 2.4ms

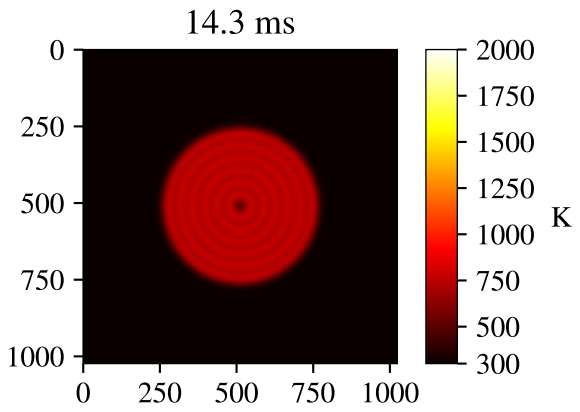
(c) Iteration 200 at 4.8ms

(d) Iteration 300 at 7.1ms

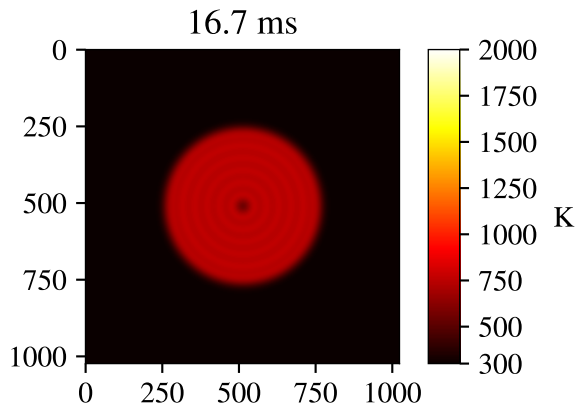
(e) Iteration 400 at 9.5ms

(f) Iteration 500 at 11.9ms

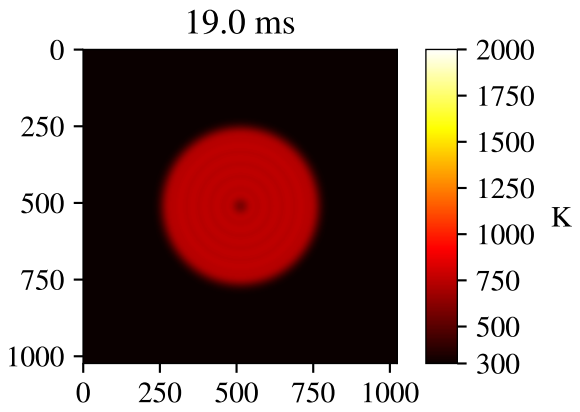
Figure 3. Evolution of heat equation, displaying relaxation over time.



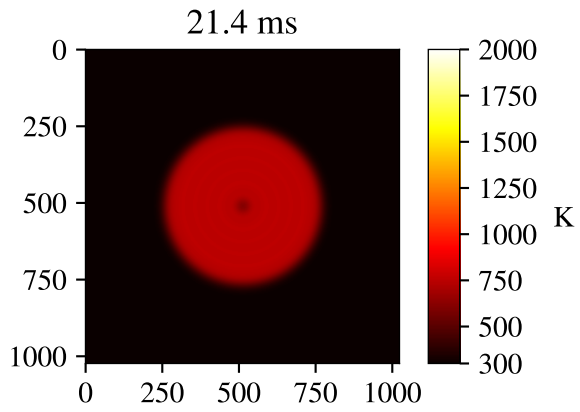
(g) Iteration 600 at 14.3ms



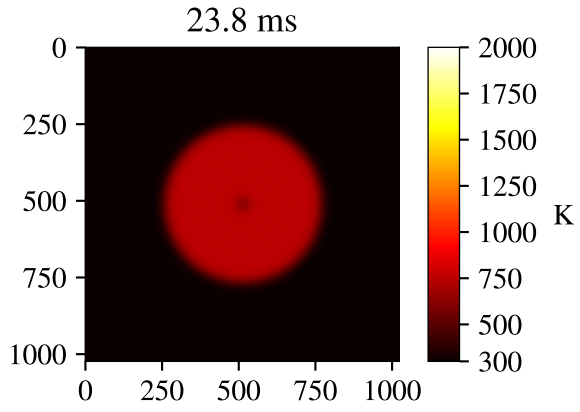
(h) Iteration 700 at 16.7ms



(i) Iteration 800 at 19.0ms

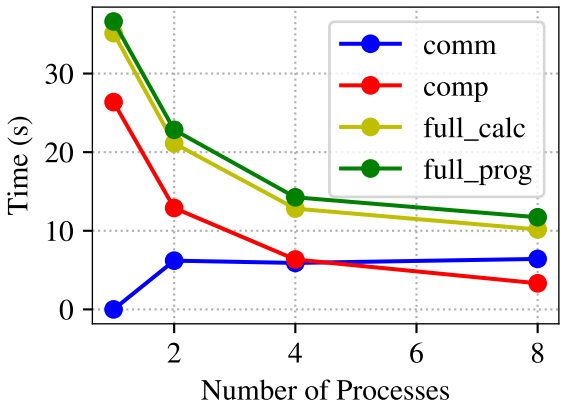


(j) Iteration 900 at 21.4ms

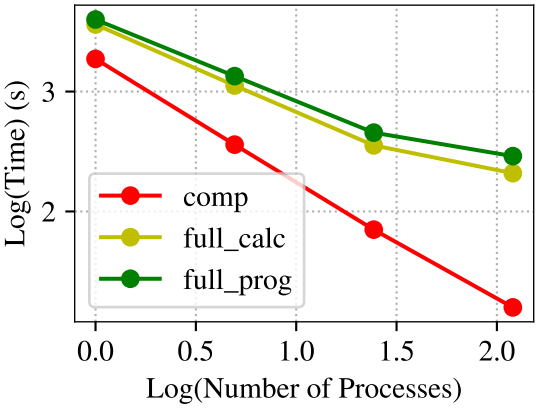


(k) Iteration 1000 at 23.8ms

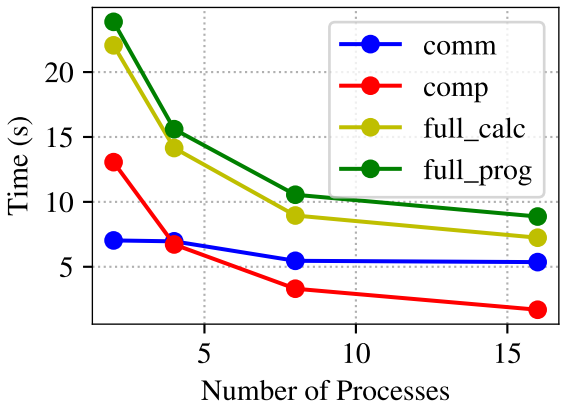
Figure 3. Evolution of heat equation, displaying relaxation over time (cont.).



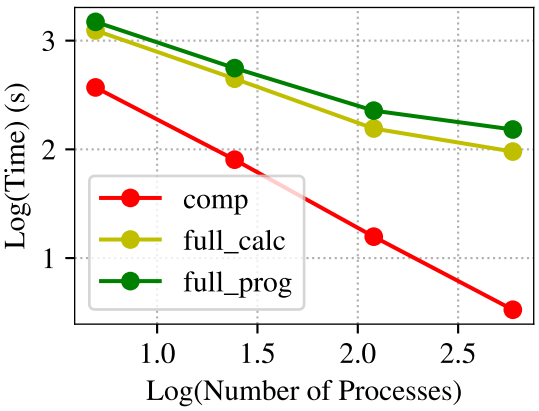
(a) CAC cluster with one node.



(b) CAC cluster with one node log-log plot.



(c) CAC cluster with two nodes.



(d) CAC cluster with two nodes log-log plot.

Figure 4. Timing results, showing a linear speedup for the parallel computation and non-linear speedup for all other parts. Note that comm refers to the communication part of the calculation, comp refers to the parallel calculation, full_calc refers to the full calculation (i.e. including parallel initialization, parallel calculation, and communication) and full_prog refers to the full program runtime. For the communication values, the maximum value from all processes was selected because it acts as the bottleneck for all other processes. The communication time is observed to be reasonably constant for different numbers of processes. Note that the communication time is excluded from the log-log plots due to the zero value of the first point for one process on one node.

IV. CONCLUSION

We demonstrated the scalability of parallel computing with MPI for solving the 2D heat equation. First, we demonstrated the expected dissipative behaviour of the heat equation for an initial condition of hot ripples surrounded by a cold region. Then, we demonstrated a linear speedup for the parallel computational part of the program and a non-linear speed up for all other parts. In particular, we found that the communication acts as the main bottleneck for parallelized program.