ENPH 479 Assignment #4: Parallelized 2D Heat Equation

Matt Wright[1,*]

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

(Dated: March 16, 2022)

The heat equation models the flow of temperature over time and space in a system. The partial differential equation can be generalized to higher dimensions and applied to a wide range of problems in engineering and mathematics but we narrow our attention to the 2-dimensional case. We discretize time and space and use finite differencing to translate the problem into a form more naturally solvable using numerical methods. Since large-scale applications would require an intractable amount of memory, we apply parallel computing to reduce the memory and computational burden on any one computer. This is accomplished by using the Message Passing Interface standard for Python and the Center for Advanced Computing's Frontenac cluster. We define initial conditions and boundary conditions so that we can solve the problem and visualize the evolution of the system. Then, we investigate how increasing the number of processes used to solve the problem influence the run time of the algorithm. We note that the amount of time spent computing the evolution of the system reduces by half as we double the number of tasks but the communication time remain static, defining a clear lower bound on run time gain of the parallel algorithm.

I. INTRODUCTION AND THEORY

The heat equation is a partial differential equation that is widely studied in physics and mathematics. As suggested by the name, the equation can be used to describe how heat diffuses through a given region. Due to this equations roots in physical phenomena, the 1-, 2-, and 3-dimensional versions of the equation are often studied but it can easily be generalized to N dimensions for applications in pure mathematics. Here, we focus our attention on the 2-dimensional case with,

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

where U is the temperate and D the thermal diffusivity, which is a property of the medium. We will assume the thermal diffusivity is constant throughout the medium that determines how fast temperature changes [1]. For numerical analysis, this can easily be discretized using finite differencing and the forward Euler scheme to evaluate the derivatives on a grid:

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

Solving this equation requires initial conditions (which are defined in the next section) and boundary conditions. We will choose boundary conditions in which the edge of the region remains at a constant temperature. This gives a standard boundary value problem that can be solved numerically. To do so, we will chose our spatial and time steps to ensure a stable approximation using the Courant Friedrichs Lewy condition, which requires:

$$C \geq D\frac{\Delta t}{(\Delta x)^2}, \tag{3}$$

where C is the Courant number, which we take to be $1/2$ here [2].

This method to numerical analysis can become intractable as the size of the simulation or the granularity of the analysis increases. To properly keep track of all the spatial points in a 2-dimensional grid, the memory requirements grow as $O(n^2)$, where n is the number of points used to discretize each dimension. This becomes even more apparent in the 3-dimensional case where one has to keep track of a all the points in a volume. A serial approach to solving the problem quickly becomes unrealistic for any one computer to solve large-scale applications, such as modelling the Earth's climate. This leads us to the technique of distributed computed and parallelized code.

Distributed computing uses a cluster of computers (or *nodes*) to share the workload of a large computational problem to reduce both the memory and number of computations required by each individual computer. This often requires some degree of communication between processes which is coordinated (in this work) using the Message Passing Interface (MPI) standard for Python, `mpi4py`, which defines the communication protocols. The cluster used here is the Center for Advanced Computing's[1] Frotenac cluster.

In this work we first define the specifics of the problem we are working on and how it is being discretized in Section II. We then implement the solution and solve the problem, displaying our results in Section III. We first visualize the time evolution of the system with a set of contour plots and then investigate the effect of increasing the number of number of tasks and nodes on the execution time of the algorithm.

II. METHODOLOGY

Armed with a discretized problem, a cluster of computers, and the software to orchestrate them, we must design

* matt.wright@queensu.ca

[1] https://cac.queensu.ca/

the code in such a way that allows us to take advantage of the cluster. We begin by choosing a dimension to parallelize along (arbitrarily choosing y here) and define our 2-dimensional array for U to have 1024 points for x and $1024/p$ points for y where p is the number of processes used for parallelization. However, it can be seen in Eq. (2) that the state at the next time step depends on the neighbouring point's state. Therefore, we actually define the U array to have two additional rows padding the top and bottom (for y being the vertical) to store the neighbouring states that are communicated by other processes. This allows us to implement a clean and efficient vectorized form of Eq. (2) that solves for the evolution of all interior points while maintaining the constant temperature boundary conditions chosen above. We initialize the temperature using the initial conditions:

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

where the units of temperature are Kelvin and r is the radius from the center of the slab. This is done within each process so an offset for y needs to be kept track of. From here, the system is evolved by first communicating the top and bottom boundary rows to and from the appropriate process before then solving for the next time step using Eq. (2).

We consider a system in which our spatial dimensions are in the range, $0 \leq x, y \leq 20.48$ (in mm), the diffusivity is $D = 4.2$ mm^2/s, and the boundary temperature remains at $U = 300$ K.

III. RESULTS

The system was solved for 1001 time steps using the specifications and approach defined above. The results of the evolution are shown in Fig. 1. The results are the same regardless of the number of tasks executing the job or even if it uses parallelized code or not, however, these results were obtained for 4 processes running on one node where all the slices of U were gathered by the root node and saved. It can be seen that the areas of high temperature begin to diffuse and the energy is spread to areas of lower temperature as predicted by the second law of thermodynamics. This is indicated by the sharp rings of high temperature beginning to morph into one fuzzy blob of a lower temperature.

We then investigated the relationship between run time and the number of individual processes being run. The results of the experimentation are shown below in Table III. It should be noted that there is considerable variance over different jobs but this was not estimated here. It can be seen that communication time contributes a small portion to the run time (on the order of 0.1 s or a few percent) for the single node case compared to the multi-node results, which takes on the order of a few seconds – even exceeding the computation time. While this is dependent on the job, the amount of variance in communication time for the multi-node configuration is much larger, ranging between times of 0.5 s up to over 6 s in some cases. It can also be seen that as we double the number of processes working to complete task, the computation time is essentially reduced by a half. This is exactly as we expect because the size of the computational problem reduces by a half. However, it is key to note that the communication time does not reduce and stays (roughly) static so the overall run time does not see such a significant reduction in run time. Therefore, the lower bound on the total run time to solve this problem on this cluster is on the order of 0.1 s as defined by the communication time.

Table I. Execution time on root process for various levels of resource allocation for 1001 time steps. Compares the time for total execution, communication between processes, and evolving the system against the number of nodes and tasks. All times are in units of seconds and the *Processes Configuration* column has node and task number pairs given to the cluster job.

Process Configuration	Communication (s)	Computation (s)
(1, 1)	0.0	22.47
(1, 2)	0.12	9.39
(1, 4)	0.56	4.05
(1, 8)	0.33	2.12
(1, 16)	0.21	1.05
(2, 8)	2.30	1.95
(4, 8)	2.08	1.91

IV. CONCLUSIONS

Solving the heat equation numerically using finite-differencing is relatively simple and effective approach to the problem but can become intractable for even the most powerful computers as the size or the dimension of the problem increases. The problem happens to be naturally parallelizable since communication along the boundaries is all that is required to evolve the system. We designed parallelized algorithm to solve the problem and investigate both the results of the simulation and the change in run time with different hardware allocations. We note that the communication time for the small-scale, 2-dimensional problem is quite small but certainly does act as a limiting factor on the overall run time as the number of processes increases beyond 16 processors. We also note that the communication time becomes significantly more restrictive as we add additional nodes to the cluster.

References

[1] "The 1D diffusion equation," .
[2] R. Courant, K. Friedrichs, and H. Lewy, "On the partial difference equations of mathematical physics," IBM Journal of Research and Development **11**, 215–234 (1967).

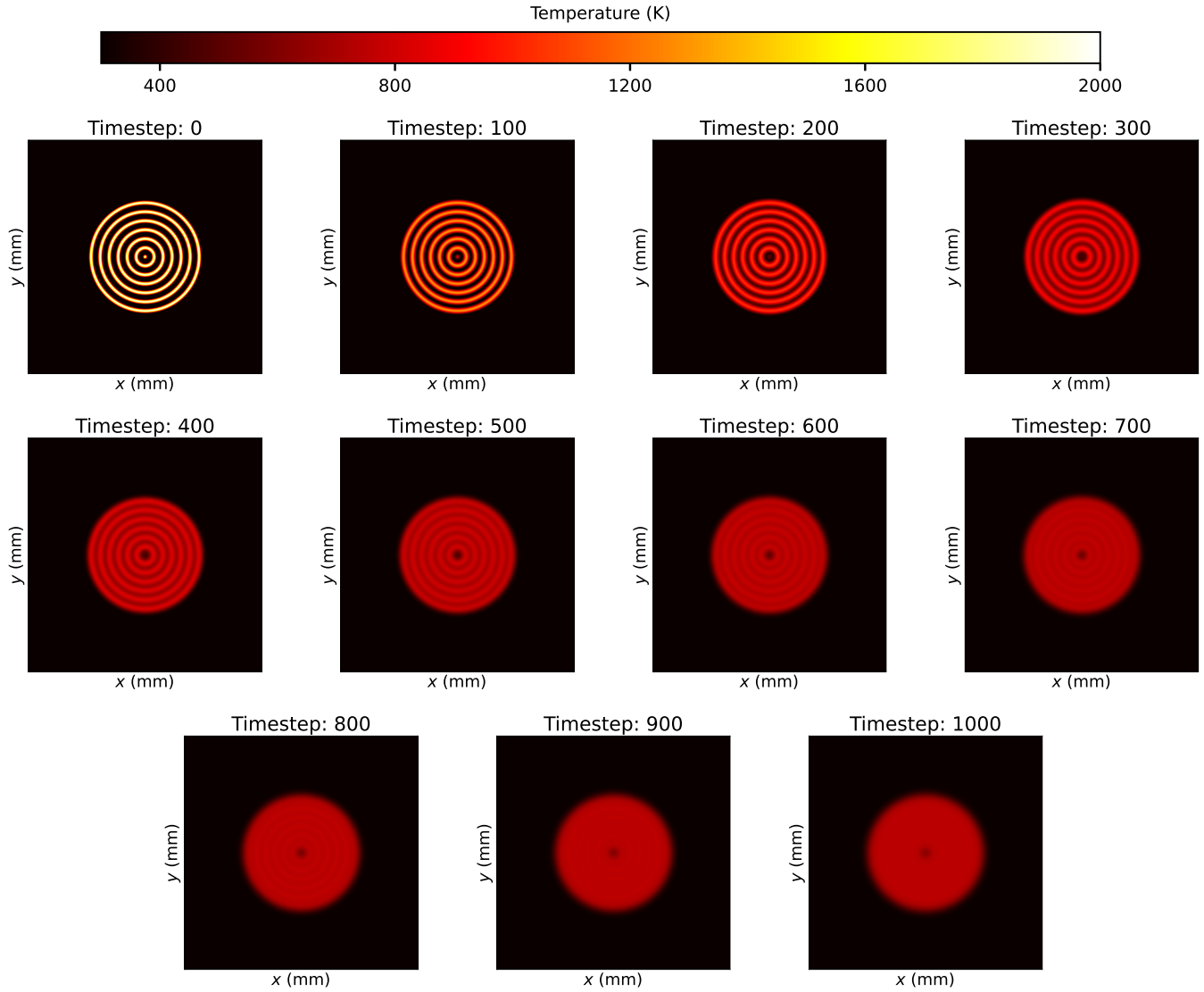

Figure 1. Time evolution of the 2-dimensional heat equation on a 20.48 mm by 20.48 mm sheet with diffusivity of 4.2 mm^2/s. The discretization uses 1024 spatial points and a time step size of 23.8 μs to ensure a stable approximation.