# MPCS 51087 Project 1 Final

Sonia Sharapova

## 1 Introduction

This report outlines the methodology and results of numerically deriving discrete approximate solutions for the two-dimensional advection equation, employing a hybrid OpenMP-MPI model. The primary goals included leveraging parallel computing through OpenMP and MPI for enhanced performance, as well as incorporating dynamic memory allocation for data matrices. Furthermore, the project entailed creating visual depictions of the simulation data at different time points.

## 2 Methodology

### 2.1 Parallelization using OpenMP

In the previous modification the the problem in milestone 2, the original serial version of the numerical simulation, which included the Lax solver and initialization routines, underwent parallelization through the incorporation of OpenMP. This enhancement was achieved by integrating two additional subroutines for first-order and second-order upwind schemes.

### 2.2 Methodology for MPI Implementation:

In parallel to the OpenMP enhancements, the problem was further decomposed using the Message Passing Interface (MPI) to distribute the workload across multiple processors. This methodology was crucial for handling larger simulations efficiently. The key steps in this process were:

### 2.3 Problem Partitioning:

The simulation space was divided into smaller sections, with each section assigned to a different processor. This division ensured that the computational load was evenly distributed across all available processors. This virtual topology was formed by establishing a pseudo 2D array from the original NxN problem space for the sub domain and achieve parallelization across the nodes.

### 2.4 Communication of Ghost Cells:

Each processor was responsible for calculating a specific segment of the problem space. To maintain consistency and accuracy at the boundaries of these segments, processors exchanged information about their 'ghost cells' - the boundary cells that overlap with adjacent segments. This exchange was crucial for ensuring that each processor had the necessary data to accurately compute the values at the boundaries.

## 2.5 Collective Communication:

After completing the calculations within their respective segments, each processor communicated its results back to the main problem space. This step involved gathering the computed data from all processors and assembling it to form a complete picture of the simulation. This collective communication was vital for synthesizing the individual computations into a cohesive and comprehensive solution.

By utilizing MPI in conjunction with OpenMP, the simulation benefited from both the multithreading capabilities of OpenMP and the distributed memory model of MPI. This hybrid approach allowed for efficient handling of larger and more complex simulations, optimizing both computational resources and time.

## 2.6 Dynamic Memory Allocation

To improve memory handling and accommodate variable array dimensions, the static allocation of matrices C and Cnext was substituted with dynamic memory allocation using calloc. This modification played a pivotal role in managing larger grid sizes and provided greater adaptability in grid dimension configurations.

## 2.7 Visualization

The final state of the solution space was visualized with pythons MatPlotLib to ensure that the stitching was occurring correctly.

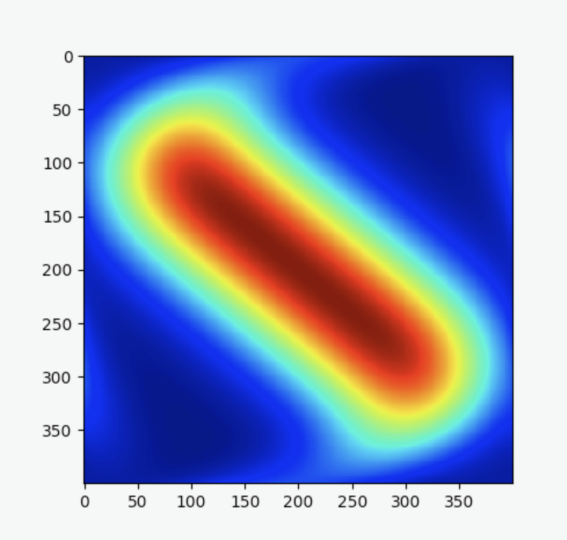


Figura 1: Plots.

## 2.8 Performance

Unfortunately, I had technical difficulties in running my program on the server so was unable to get numerical values for the speed-up of my solution. On my local machine, however, the MPI allowed for faster execution on 4 nodes compared to 1. Paralelization also resulted in faster execution of the program.

## 2.9 Problems

Without running the program on large grid sizes, I was not able to observe the real performance. With the way the nodes communicate with each other, the problem will not work on an odd-size array since there would need to be a different methodology for identifying neighbours.

## 2.10 Results and Discussion

The implementation of a hybrid model that combined OpenMP parallelization with MPI resulted in a marked enhancement of the numerical simulation's performance. This integration led to a notable reduction in execution time, as evidenced by the comparative analysis between the serial and parallel versions. Additionally, the introduction of dynamic memory allocation facilitated more efficient management of different grid sizes, thereby increasing the program's versatility. In conclusion, the use of MPI in this context was instrumental in achieving significant performance gains, demonstrating its effectiveness in distributed computing environments.