Peachy Parallel Assignments (EduPar 2025)

H. Martin Bücker
Institute for Computer Science
Friedrich Schiller University Jena
Jena, Germany
ORCID: 0000-0002-5210-0789

Johannes Schoder
Institute for Computer Science
Friedrich Schiller University Jena
Jena, Germany
ORCID: 0000-0002-0771-3738

Xiaoyuan Suo
Dept. of Computer and Information Sciences
Webster University
St. Louis, MO, USA
ORCID: 0000-0003-2473-0705

David P. Bunde
Dept. of Computer Science
Knox College
Galesburg, IL, USA
ORCID: 0000-0001-6334-356X

***Abstract*—Peachy Parallel Assignments are assignments on parallel computing topics that have been tested in a classroom, are designed for easy adoption by others, and are "cool and inspirational" for students. They are published so that others can benefit from the authors' assignment-creation work, raising the overall quality of assignments given in our area while also saving instructors time. The assignments are selected competitively at the Edu* workshops. This article presents two assignments selected for presentation at EduPar 2025: one that has students use the MapReduce framework to resize images using the 2D discrete Fourier transform and one that has them parallelize an agent-based model to simulate a zombie outbreak.**

***Index Terms*—Peachy Parallel Assignments, Parallel computing education, Parallel programming, MapReduce, Big Data, Discrete Fourier transform, Parallel simulation**

I. Introduction

Assignments are an important part of any course. High-quality assignments solidify and reinforce lessons that students learn in class. Ideally, assignments also excite students about the material and give them a sense of accomplishment, encouraging them to dive deeply into the material and then continue pursing the subject. Excited students also provide positive word-of-mouth advertising for the field, potentially attracting others to join them.

Unfortunately, creating such assignments requires time, creativity, and hard work. There is also risk to creating a new assignment because sometimes there are unforeseen problems such as required student background or a brittle environment that cause assignments to be less successful than anticipated.

Peachy Parallel assignments are an effort to share successful, high-quality assignments on Parallel and Distributed Computing (PDC) topics. The goal is to improve the quality of assignments in this area, recognize the instructors who do the important work of creating such assignments, and simultaneously reduce the workload and risk for instructors who adopt the assignments.

With these goals in mind, Peachy Parallel Assignments are solicited from the community, selected competitively, presented at the Edu* (EduPar and EduHPC) workshops (e.g. [2], [6], [7]), and archived online for others to use (https://tcpp.cs.gsu.edu/curriculum/?q=peachy). The selection criteria are as follows:

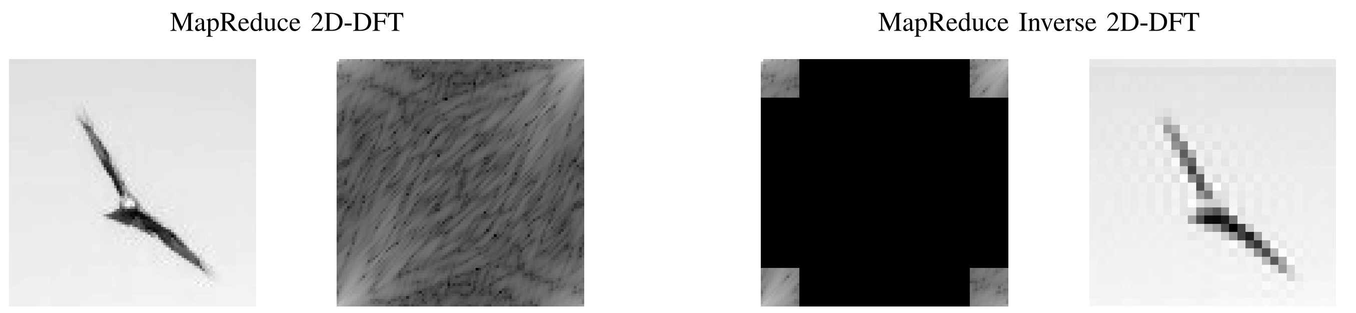
- Tested: All assignments have been tested with real students, helping to identify any issues and reducing risk for adopters.
- Adoptable: All assignments come with materials to facilitate adoption by others. More broadly, we seek assignments that will be useful to many others, covering common topics, relying on widely-used technologies, etc.
- Cool and inspirational: Assignments must motivate students by impressing them with the material's importance or teaching it an an attractive way.

This paper gives two Peachy Parallel Assignments which were selected for presentation at EduPar 2025. The first assignment has students use the MapReduce framework [5] to compute a 2D discrete Fourier transform, with image resizing as an application. The second assignment has students complete and then parallelize a simple agent-based simulation of a zombie outbreak as a visual introduction to the impact of parallel computing.

II. MapReduce for DFT and Image Resizing

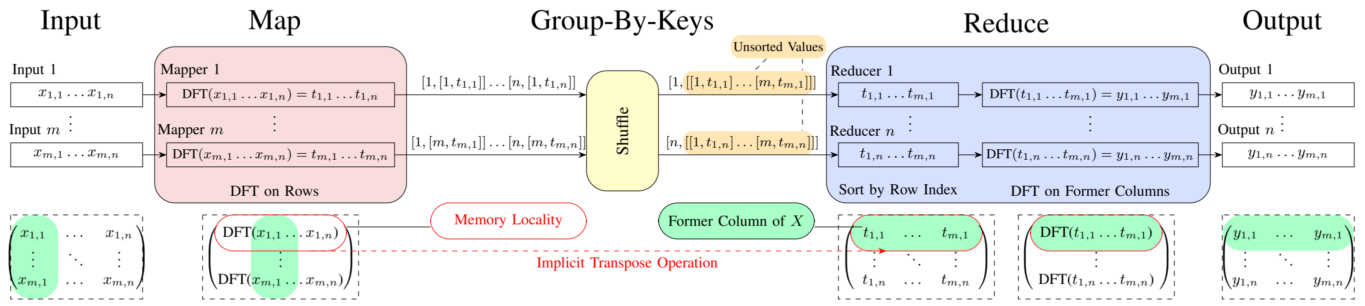
The widely-known MapReduce programming model is an integral part of processing massive datasets. It demands a high capability of abstraction to adapt existing algorithms to this specific programming paradigm. In our first assignment, students are asked to develop a MapReduce version of the row-column algorithm to compute the two-dimensional discrete Fourier transform. The most remarkable outcome of this assignment is that, by using MapReduce, an explicit matrix transpose operation can be encapsulated in MapReduce's group-by-keys phase.

The Ara cluster at Friedrich Schiller University Jena is supported by DFG grants INST 275/334–1 FUGG and INST 275/363–1 FUGG. The work on the first assignment is partly funded by the German Federal Ministry of Education and Research (BMBF), within project THInKI, project number 16DHBKI084.



i) Input Image (96 × 96) ii) Fourier Transform (96 × 96) iii) Low-Pass Filtered (30 × 30) iv) Output Image (30 × 30)

(a) The full pipeline for scaling down images by removing high frequencies and performing an inverse discrete Fourier transform.


Input
Map
Group-By-Keys
Reduce
Output
Input 1
$x_{1,1} \dots x_{1,n}$
Input m
$x_{m,1} \dots x_{m,n}$
Mapper 1
$\mathrm{DFT}(x_{1,1} \dots x_{1,n}) = t_{1,1} \dots t_{1,n}$
Mapper m
$\mathrm{DFT}(x_{m,1} \dots x_{m,n}) = t_{m,1} \dots t_{m,n}$
DFT on Rows
$[1,[1,t_{1,1}]] \dots [n,[1,t_{1,n}]]$
$[1,[m,t_{m,1}]] \dots [n,[m,t_{m,n}]]$
Shuffle
Unsorted Values
$[1,[[1,t_{1,1}] \dots [m,t_{m,1}]]]$
$[n,[[1,t_{1,n}] \dots [m,t_{m,n}]]]$
Reducer 1
$t_{1,1} \dots t_{m,1}$
$\mathrm{DFT}(t_{1,1} \dots t_{m,1}) = y_{1,1} \dots y_{m,1}$
Reducer n
$t_{1,n} \dots t_{m,n}$
$\mathrm{DFT}(t_{1,n} \dots t_{m,n}) = y_{1,n} \dots y_{m,n}$
Sort by Row Index
DFT on Former Columns
Output 1
$y_{1,1} \dots y_{m,1}$
Output n
$y_{1,n} \dots y_{m,n}$
Memory Locality
Former Column of X
Implicit Transpose Operation

(b) The various phases of the 2D-DFT in MapReduce with the transpose operation implicitly encapsulated in the group-by-keys phase.

Fig. 1: The assignment's application for image size reduction in Fig. 1a and the MapReduce job for the 2D-DFT in Fig. 1b.

The assignment is part of a course at Friedrich Schiller University Jena targeting master's students with a diverse background in various STEM disciplines. The course concentrates on big data processing. A previous publication already describes a selection of further assignments of this course [10].

The new assignment is built around the two-dimensional discrete Fourier transform (2D-DFT) [14]. Its main goal is to encourage students to design a MapReduce algorithm from a given serial algorithm. This conceptual task requires them to raise the level of abstraction at which they view and design algorithms serially.

In addition to a clear understanding of MapReduce, a prerequisite for this assignment is a basic knowledge of Apache Hadoop. The students are encouraged to solve the assignment with Python using Hadoop Streaming. While we are using the DFT as an example for designing MapReduce algorithms, the assignment itself is not intended to teach any principles of digital signal processing. Thus, the students are allowed to use Python libraries, making available the one-dimensional discrete Fourier transform (1D-DFT) upon which the given 2D-DFT algorithm is built. As the assignment is meant to provide a more intricate example of designing algorithms in MapReduce, its key objective is to establish the development of the mappers and reducers rather than benchmarking the performance on massive datasets.

The assignment comprises a short introduction to a simple yet powerful 2D-DFT algorithm, two programming tasks, and a discussion of an application. This application not only computes the DFT in the forward direction, but also executes the inverse DFT to demonstrate resizing of images as sketched in Fig. 1a. This more complex application is used as an illustrating example whose implementation is presented to the students. Students are required to implement the two programming tasks on the Ara system, the university's central compute cluster. The complete assignment, including its solution, is available in a public git repository [11].

A. Assignment's Structure

A brief introduction first addresses the fundamentals of the discrete Fourier transform. It takes approximately 30 minutes, depending on the background of the participants. It is mainly based upon small code examples included in a Jupyter notebook. The 2D-DFT transforms a discrete signal from the time to the frequency domain. For an arbitrary matrix $A \in \mathbb{R}^{m \times n}$, the symbol $\mathbf{a}_i$ represents its ith row and the notation $a_{i,j}$ denotes its element in row i and column j. Let $\mathrm{DFT}(\mathbf{a}_i)$ denote a 1D-DFT of a matrix row that is stored contiguously in memory (row-major order of A). Then, the first programming task of the assignment is to implement the serial row-column algorithm for the 2D-DFT by multiple 1D-DFTs first along all rows and then along all columns of a matrix X as described in Algorithm 1.

This algorithm takes into account the memory layout of matrices. It first accesses X in a row-oriented fashion computing the 1D-DFTs for all rows of X resulting in an intermediate matrix T. Since the code $\mathrm{DFT}(\cdot)$ assumes that its input is stored contiguously in memory, a transpose operation on T

```
Data: $X \in \mathbb{R}^{m \times n}$
Result: $Y \in \mathbb{R}^{n \times m}$
for $i \leftarrow 0$ to $m$ do
    $\mathbf{t}_i \leftarrow \mathrm{DFT}(\mathbf{x}_i)$ /* 1D-DFT(row $i$ of $X$)  */
$T \leftarrow$ Transpose($T$) /* explicit transpose  */
for $i \leftarrow 0$ to $n$ do
    $\mathbf{y}_i \leftarrow \mathrm{DFT}(\mathbf{t}_i)$/* 1D-DFT(row $i$ of $T$)  */
```

Algorithm 1: Row-column algorithm for the 2D-DFT

is needed. This operation results in excessive amounts of memory transfer, as almost all elements of the matrix have to be relocated in memory. After the transposition, the 1D-DFT is computed for each row of the transposed matrix resulting in the output Y of the 2D-DFT. Notice that, for the sake of simplicity, we omit a second transpose operation on Y and tolerate that the output $Y \in \mathbb{R}^{n \times m}$ is transposed compared to the input matrix $X \in \mathbb{R}^{m \times n}$.

The second programming task is to design a MapReduce algorithm following the row-column approach. This algorithm is summarized in Fig. 1b. Throughout the map phase, each mapper i receives the row $\mathbf{x}_i$ of the input matrix and computes the 1D-DFT of that row stored in the vector $\mathbf{t}_i$. Each element $t_{i,j}$ of this vector is then emitted in a key-value pair. More precisely, the key-value pairs $(j, (i, t_{i,j}))$ for $j = 1, 2, \ldots, n$ are generated, whose key is the column index j and whose value is the tuple $(i, t_{i,j})$ consisting of the row index i and the element $t_{i,j}$. These tuples are then grouped by their column indices j before being passed to the reducers. The task at a reducer j is to sort the elements $t_{i,j}$ by their row indices, as it cannot be guaranteed that they are passed to the reducer in their original order. These resulting ordered vectors now contain the former columns of the matrix, and the 1D-DFT of these ordered vectors is computed, taking advantage of the memory layout. The exploitation of data locality is managed and handled by the Python library NumPy and its DFT routines. The combination of the group-by-keys phase and the sorting in the reduce phase resembles the explicit transpose operation from Algorithm 1. However, in MapReduce, this operation is given implicitly. Following up on the programming tasks, this implicit transformation is discussed with the students. Lastly, as shown in Fig. 1a, in addition to the 2D-DFT, a second MapReduce step performing an inverse 2D-DFT can be added. This results in a basic image resizing algorithm. The code is included in the Jupyter notebook and given to and discussed with the students.

B. Experiences and Outlook

To evaluate this assignment, there was a small, optional, anonymous survey handed out to the students. Unfortunately, only four (out of thirteen) students completed the survey, leaving us with no statistically significant feedback. The responses indicate that the assignment seemed to be of reasonable difficulty. For students not familiar with the DFT, its introduction fell a bit short. However the assignment appeared to be generally well received.

A strength of the assignment is to illustrate the high expressivity of the MapReduce programming model. For the DFT, it allows one to implicitly execute a matrix transpose operation that otherwise needs to be specified explicitly. In future instances of the course, we plan to emphasize this aspect even more, as it constitutes the main learning goal of this Peachy Assignment.

III. Simulating a Zombie Outbreak

For our second assignment, we present a novel approach to teaching Parallel and Distributed Computing (PDC) concepts through an interactive, grid-based simulation of a zombie outbreak. The simulation uses agent-based modeling to simulate the spread of a zombie infection across a population, using parallel computing techniques to speed up the computation. By visualizing the effects of parallelism, students can observe the performance benefit from PDC, making abstract concepts more tangible and engaging.

PDC offers powerful tools for solving complex problems, but it can be difficult for students to grasp the real-world implications of parallelism [9], [13]. Interactive simulations provide an effective way to illustrate these concepts in an engaging and relatable way [1], [4].

This paper presents a zombie outbreak simulation that models infection spread using agent-based modeling on a 2D grid. The underlying concept draws inspiration from the work of Munz et al. [8], who used mathematical modeling to study the dynamics of zombie outbreaks. This assignment builds on their work, implementing a simple simulation using C++ 2D arrays to demonstrate how parallelism can accelerate computations in the context of agent-based modeling.

A. Methodology

The assignment was created for a CS2 course where students have a background in C++ programming. The goal of this assignment is to demonstrate the performance improvement achieved through multi-threaded programming in 2D array processing. We began by presenting an animation written in JavaScript to illustrate the zombie outbreak simulation (Fig. 2). The animation was designed with a slower speed, allowing users to visually observe the effects of parallelism and how the infection spreads over time. A slider feature enables users to adjust the number of threads. The animation is available in a GitHub repository [12]. In the 2D grid representation shown in Fig. 2, humans are initially represented by blue cells, while zombies are in red. As the simulation progresses, zombies infect neighboring humans, turning their cells red, with the infection spreading based on proximity.

Following this demonstration, we explained how the simulation works and introduced students to the core implementation details. Students are required to implement the simulation using C++ and representing the grid as a 2D array. To apply parallel computing, they must use OpenMP, a widely-adopted framework for parallel programming in C++ [3].

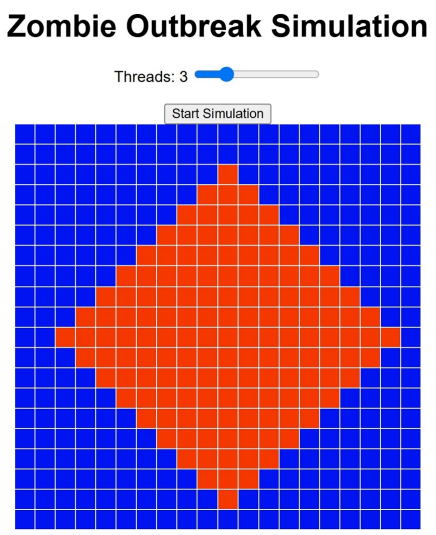
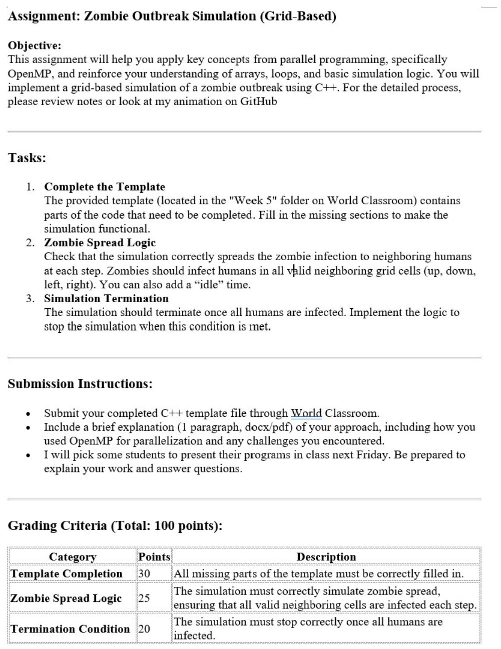


Fig. 2: Simulation Screenshot



Assignment: Zombie Outbreak Simulation (Grid-Based)

Objective:
This assignment will help you apply key concepts from parallel programming, specifically OpenMP, and reinforce your understanding of arrays, loops, and basic simulation logic. You will implement a grid-based simulation of a zombie outbreak using C++. For the detailed process, please review notes or look at my animation on GitHub

---

Tasks:

1. **Complete the Template**
   The provided template (located in the "Week 5" folder on World Classroom) contains parts of the code that need to be completed. Fill in the missing sections to make the simulation functional.
2. **Zombie Spread Logic**
   Check that the simulation correctly spreads the zombie infection to neighboring humans at each step. Zombies should infect humans in all valid neighboring grid cells (up, down, left, right). You can also add a “idle” time.
3. **Simulation Termination**
   The simulation should terminate once all humans are infected. Implement the logic to stop the simulation when this condition is met.

Submission Instructions:

- Submit your completed C++ template file through World Classroom.
- Include a brief explanation (1 paragraph, docx/pdf) of your approach, including how you used OpenMP for parallelization and any challenges you encountered.
- I will pick some students to present their programs in class next Friday. Be prepared to explain your work and answer questions.

Grading Criteria (Total: 100 points):

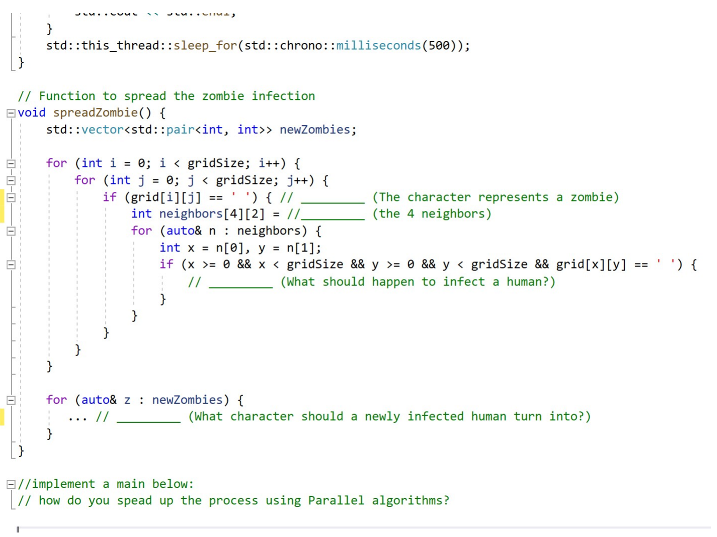
Category	Points	Description
Template Completion	30	All missing parts of the template must be correctly filled in.
Zombie Spread Logic	25	The simulation must correctly simulate zombie spread, ensuring that all valid neighboring cells are infected each step.
Termination Condition	20	The simulation must stop correctly once all humans are infected.

Fig. 3: Assignment details

1) Assignment Implementation Details: Students receive a C++ template with missing sections to complete. Key components include:

- A 2D array tracking infection status (human or zombie).
- OpenMP-based parallel computation to accelerate infection spread.
- An interactive user input that adjusts the number of threads used for parallel processing, affecting the speed of the infection spread.

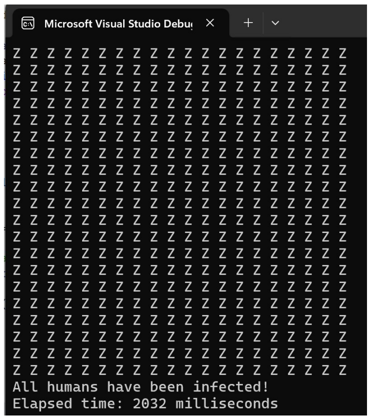
2) Results: Students’ submissions were tested with various configurations of threads, and the effects of parallelism was clearly observable. Students recorded the computational efficiency gained by utilizing multiple threads, reinforcing the



```cpp
            std::cout << std::endl;
    }
    std::this_thread::sleep_for(std::chrono::milliseconds(500));
}

// Function to spread the zombie infection
void spreadZombie() {
    std::vector<std::pair<int, int>> newZombies;

    for (int i = 0; i < gridSize; i++) {
        for (int j = 0; j < gridSize; j++) {
            if (grid[i][j] == ' ') { // _________ (The character represents a zombie)
                int neighbors[4][2] = //_________ (the 4 neighbors)
                for (auto& n : neighbors) {
                    int x = n[0], y = n[1];
                    if (x >= 0 && x < gridSize && y >= 0 && y < gridSize && grid[x][y] == ' ') {
                        // _________ (What should happen to infect a human?)
                    }
                }
            }
        }
    }

    for (auto& z : newZombies) {
      ... // _________ (What character should a newly infected human turn into?)
    }
}

//implement a main below:
// how do you spead up the process using Parallel algorithms?
```

Fig. 4: Screenshot of the program template provided to students

idea that parallelism can drastically reduce the time required for large-scale simulations. A sample output can be seen in Fig. 5.



```
Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z
Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z
Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z
Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z
Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z
Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z
Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z
Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z
Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z
Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z
Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z
Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z
Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z
Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z
Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z
Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z
Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z
Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z
Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z
Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z
All humans have been infected!
Elapsed time: 2032 milliseconds
```

Fig. 5: Screenshot of students’ output

3) Students Feedback: The use of animation to demonstrate PDC in a fun and relatable context, such as a zombie outbreak, proved to be an effective pedagogical tool. Several observations were made during the simulation:

- Parallelism: Students learn performance improvements through visual feedback as they increased thread counts. *”The simulation made parallel computing clearer—I could see the impact of multiple threads in real-time.”*
- Engagement: The game-like animation captured students’ interest, making HPC concepts more accessible. *”I enjoyed experimenting with different thread counts to see how the infection spread. It made learning more fun.”*
- Interactive Learning: Allowing students to adjust parameters and observe changes in real-time fostered a more active learning environment, where students could experiment and learn from their observations.

Despite the success of the simulation, some challenges were encountered. For example, some students struggled with the implementation details of OpenMP and parallelism. *"More detailed programming-based instructions would be helpful." "Spent over 10 hours on the implementation."*

B. Future Work

Future work may involve expanding the simulation to incorporate additional features, such as more complex infection dynamics or additional parallelism techniques. Additionally, providing students with more detailed resources on parallel programming could help them better understand the underlying mechanisms of the simulation.

References

[1] C. Bourke and J. Firestone, "Codeless PDC modules for early computing curriculum," in *Proc. IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2024.

[2] H. Bücker, J. Corrado, D. Fedorin, D. Garcìa-Àlvarez, A. Gonzalez-Escribano, J. Li, M. Pantoja, E. Pautsch, M. Plesske, M. Ponce, S. Rizzi, E. Saule, J. Schoder, G. Thiruvathukal, R. van Zon, W. Weber, and D. Bunde, "Peachy parallel assignment (EduHPC 2023)," in *Proc. Workshop on Education for High-Performance Computing (EduHPC)*, 2023.

[3] R. Chandra, *Parallel Programming in OpenMP*. Academic Press, 2001.

[4] B. Chaudhury, A. Varma, Y. Keswani, Y. Bhatnagar, and S. Parikh, "Let's HPC: A web-based platform to aid parallel, distributed and high performance computing education," *J. Parallel and Distributed Computing*, vol. 118, pp. 213–232, 2018.

[5] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. [Online]. Available: https://doi.org/10.1145/1327452.1327492

[6] A. Lazar, V. Niculescu, and D. Bunde, "Peachy parallel assignments (EduPar 2023)," in *Proc. 13th NSF/TCPP workshop on parallel and distributed computing education (EduPar)*, 2023.

[7] A. Lazar, E. Scheelk, E. Shoop, and D. Bunde, "Peachy parallel assignments (EduPar 2024)," in *Proc. 14th NSF/TCPP workshop on parallel and distributed computing education (EduPar)*, 2024.

[8] P. Munz and et al., "When zombies attack!: mathematical modelling of an outbreak of zombie infection," *Infectious disease modelling research progress*, vol. 4, pp. 133–150, 2009.

[9] D. E. Post, R. P. Kendall, and R. F. Lucas, "The opportunities, challenges, and risks of high performance computing in computational science and engineering," *Advances in Computers*, vol. 66, pp. 239–301, 2006.

[10] J. Schoder and H. M. Bücker, "Exploring data science workflows: A practice-oriented approach to teaching processing of massive datasets," *Journal of Parallel and Distributed Computing*, p. 105043, 2025. [Online]. Available: https://doi.org/10.1016/j.jpdc.2025.105043

[11] ——, "2D Fourier Transform with MapReduce," Git Repository, 2024. [Online]. Available: https://git.uni-jena.de/big_data_assignments/2D-DFT

[12] X. Suo, "Zombie attack simulation," GitHack, 2025, available: https://raw.githack.com/xiaoyuansuo51-webster/teaching-PDC-HPC/refs/heads/main/ZombieAttach_XSUO.html, Accessed: Feb. 13, 2025.

[13] X. Suo, O. Glebova, D. Liu, A. Lazar, and D. Bein, "A survey of teaching PDC content in undergraduate curriculum," in *Proc. 2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)*, 2021.

[14] C. Van Loan, *Computational Frameworks for the Fast Fourier Transform*, ser. Frontiers in Applied Mathematics. Philadelphia: SIAM, 1992, vol. 10.