TDDE65 Miniproject Report

Albin Arvidsson albar556
Martin Kaller marka727

May 15, 2024

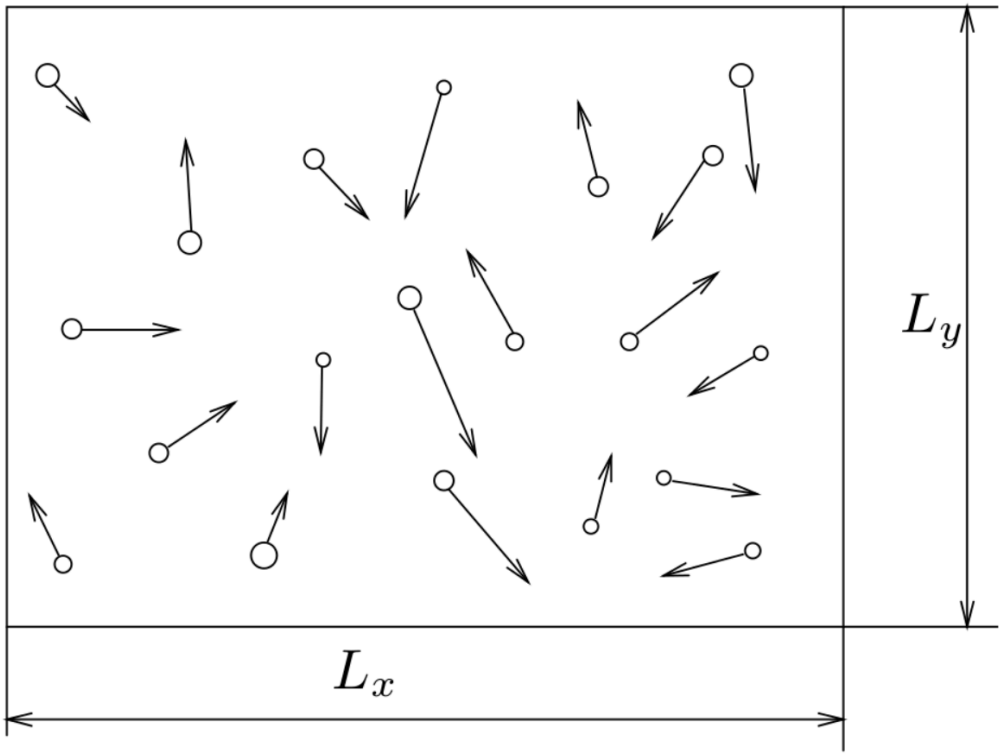

Figure 1: Particle simulation.

1 Introduction

We made a particle simulator with MPI (Message Passing Interface) [1], to verify the gas law $pV = nRT$. All particles had a radius of 1 and all collisions were perfectly elastic. No friction or other physical forces acted upon the particles. The particles moved around inside a rectangular space for simpler calculations, see figure 1. MPI was used to speed up the calculations via parallelization.

2 Method

We divided the box into a grid, where each process gets one cell. We use a 2 dimensional MPI layout for assigning the cell.

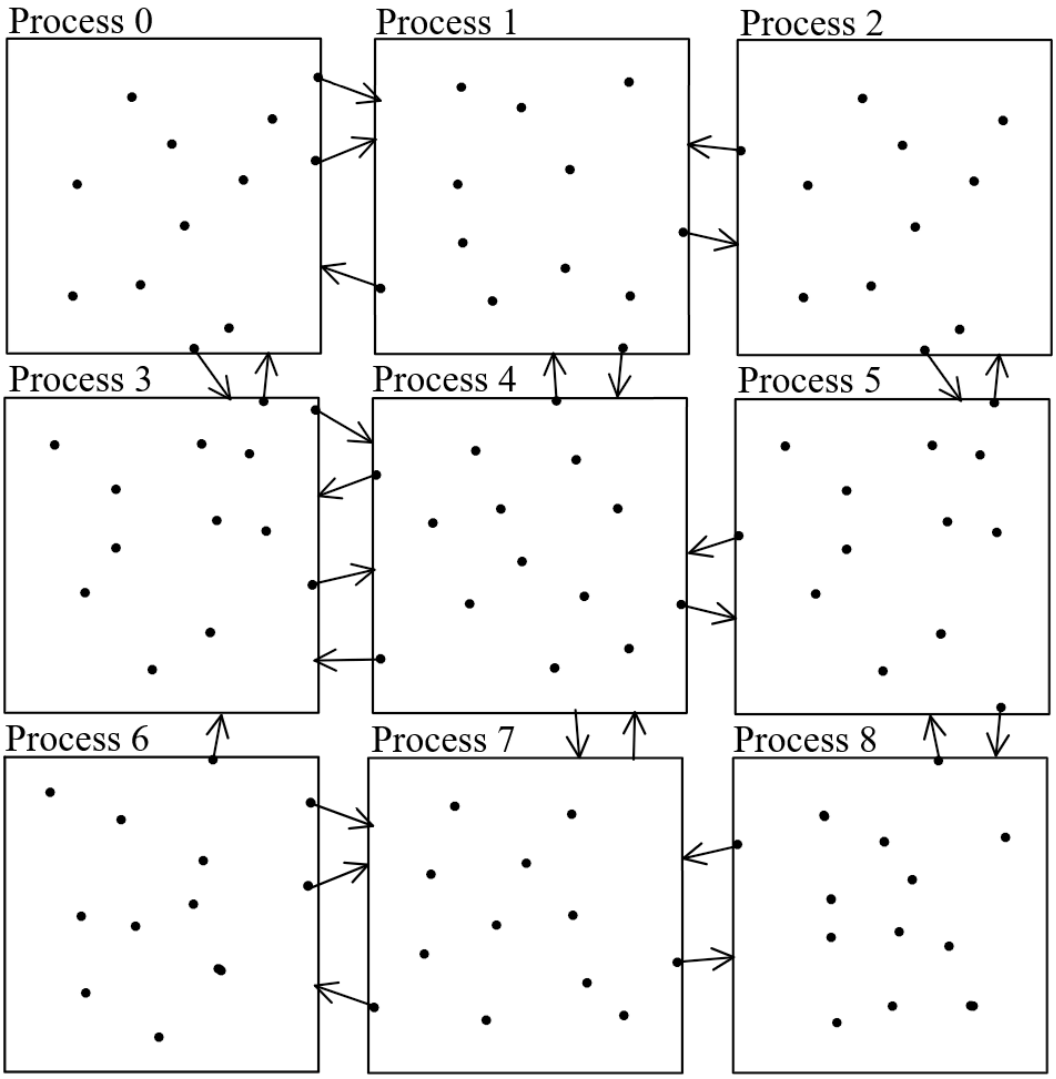

Figure 2: MPI processes in a 2D grid, with particles in each process cell. Communications of particles leaving cell visualized.

```
MPI_Comm GRID_COMM_MPI;
MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, reorder, &GRI
int my_coords[2];
int my_rank;
MPI_Cart_get(GRID_COMM_MPI, 2, dims, periods, my_coords);
MPI_Comm_rank(GRID_COMM_MPI, &my_rank);
```

Each process generates `TOTAL_PARTICLES/processes` amount of random particles. Particles are stored in `std::vector` (faster than `std::forward_list`, see discussion).

At the end of one timestamp iteration, we check which particles are outside of the process' cell. If it has a neighbor who could receive them, we remove the particles from our particle list and send them to our neighbor.

```
unsigned send_up_size = 0, send_left_size = 0, send_right_size

for (auto p = particles.begin(); p != particles.end();)
{
    if (p->coords.x < box_border.x0 && left_rank != -1)
    {
        particles_send_left[send_left_size++] = p->coords;
        p = particles.erase(p);
    }
    else if (p->coords.x > box_border.x1 && right_rank != -1)
    {
       // ...
    }
    else if (p->coords.y < box_border.y0 && down_rank != -1)
    {
        // ...
    }
    else if (p->coords.y > box_border.y1 && up_rank != -1)
    {
        // ...
    }
    else
    {
        ++p;
    }
}
```

All communications are done asynchronously in MPI with `MPI_Isend` and `MPI_Irecv`

```
if (up_rank != -1)
{
    // begin send and receive
    MPI_Isend(particles_send_up, send_up_size, PCORD_MPI, up_ra
    MPI_Irecv(particles_recv_up, INIT_NO_PARTICLES, PCORD_MPI,
}
// ... left, down, right

if (up_rank != -1) {
    // wait for response to be done
    MPI_Status status;
    MPI_Wait(&req_recv_up, &status);
    int count;
    MPI_Get_count(&status, PCORD_MPI, &count);
    for (size_t i = 0; i < count; i++)
    {
        particles.push_back({recv_buf[i], false});
    }
}
// ... left, down, right

MPI_Status tmp_status;

if (up_rank != -1)
{
    // finally wait for send request to be done
    MPI_Wait(&req_up, &tmp_status);
}
// ... left, down, right
```

2.1 Limitations

- A particle can only collide with exactly one other particle.
- A particle can not both collide and bounce on the wall
    - Can causes a particle to be outside of box wall for one timestamp
- A particle needs two timesteps to travel diagonally
- Collisions can not happen between two cells in the border between them.

3 Debugging with DDT

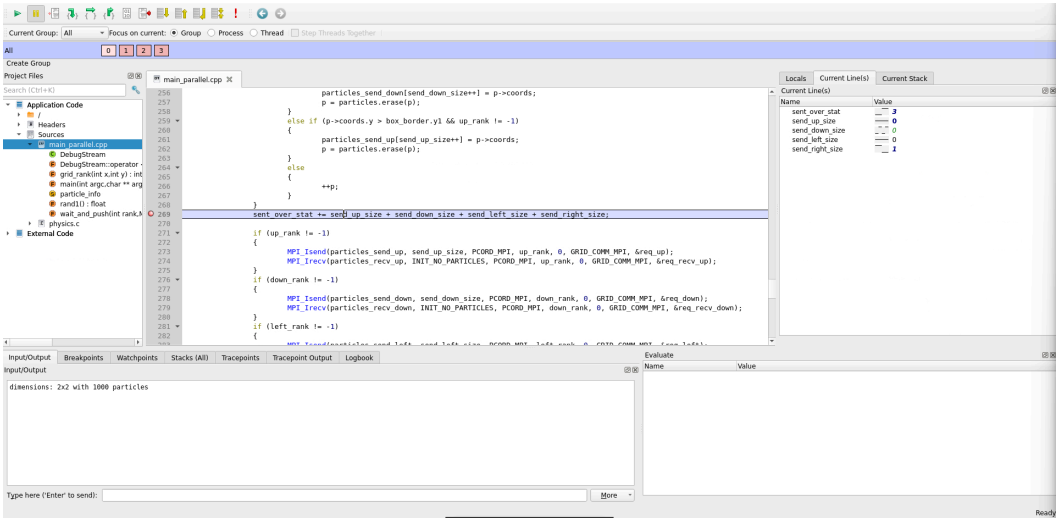

Figure 3: DDT Debugger used to inspect sent data.

We used DDT for debugging, and gathering information during our miniproject. We encountered several issues during the implementation of the miniproject and used DDT to resolve them. For example issues with particles being sent to non-existent neighbors and just disappearing. We noticed our pressure results strangely diminishing with higher timestamps. With the help of DDT it was easy to identify the issue and determine what caused it. Another use case we used DDT for was measuring the amount of sent data and comparing it between each process each step. This was made easy with DDT, see figure 3.

4 Performance analysis with ITAC

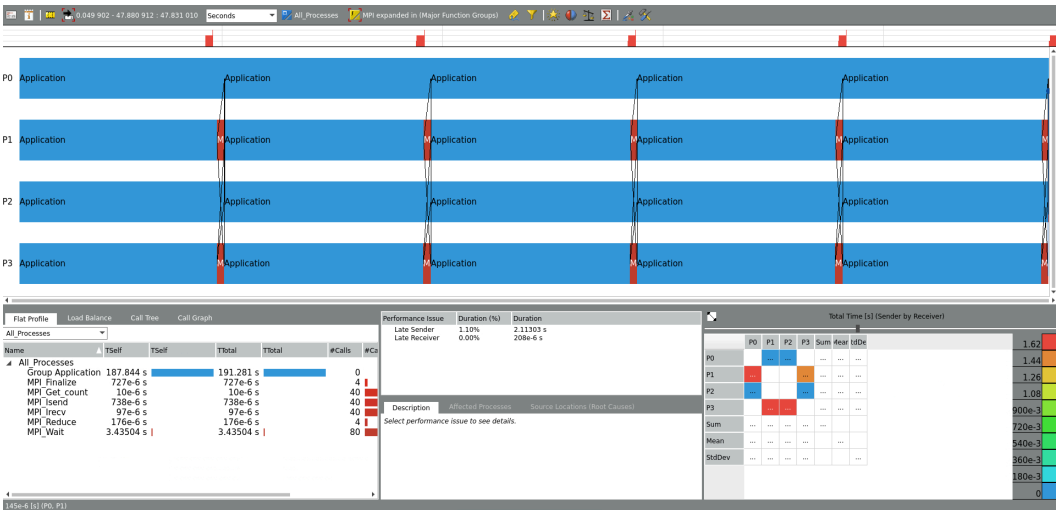

Figure 4: ITAC showing our program's executing trace.

We used ITAC during the miniproject to determine where the bottleneck in the system was. As can be seen in figure 4 we quickly determined that the computations were the heavy calculation, and that's where the optimization focus should lay for improved execution times. This differed from lab1b where the MPI communications were significantly more expensive than the actual calculations, so in lab1b we had to focus more on optimizing the MPI communications for better time performance. In the miniproject however we switched our focus to optimizing our application code, by for example improving cache locality by testing different data structures.

5 Results

The result regarding the ideal gas law, speedup and different implementations will be presented.

5.1 Ideal gas law

The gas law $pV = nRT$ was verified by calculating the temperature T based on the known values we got from running our program. If the program follows the gas law, then the temperature should remain constant for any combination of box size and particle amount. Our result can be seen in figure 5, figure 6 and figure 7

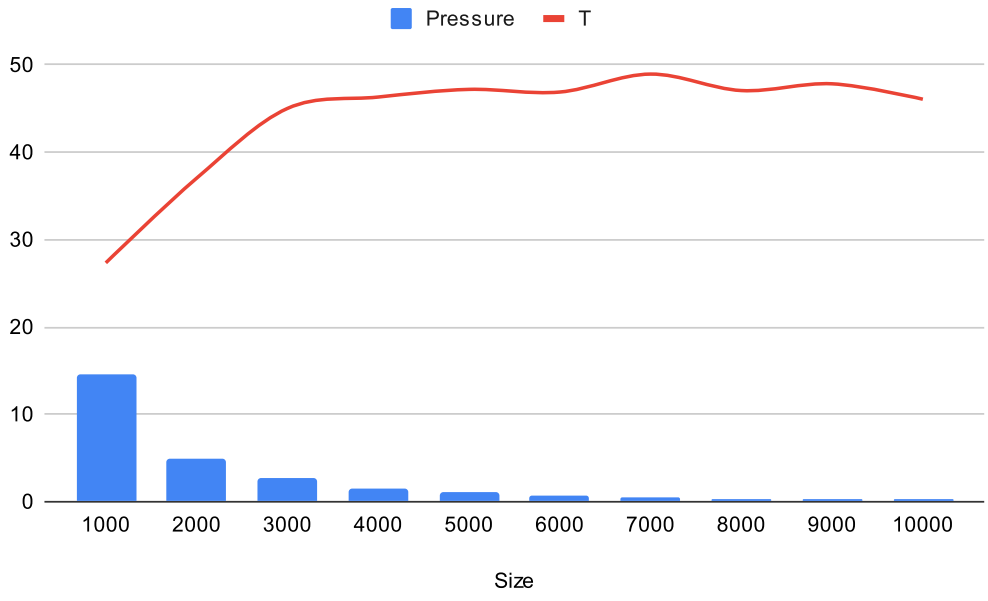

Figure 5: Relation between box size and temperature, with 32000 particles.

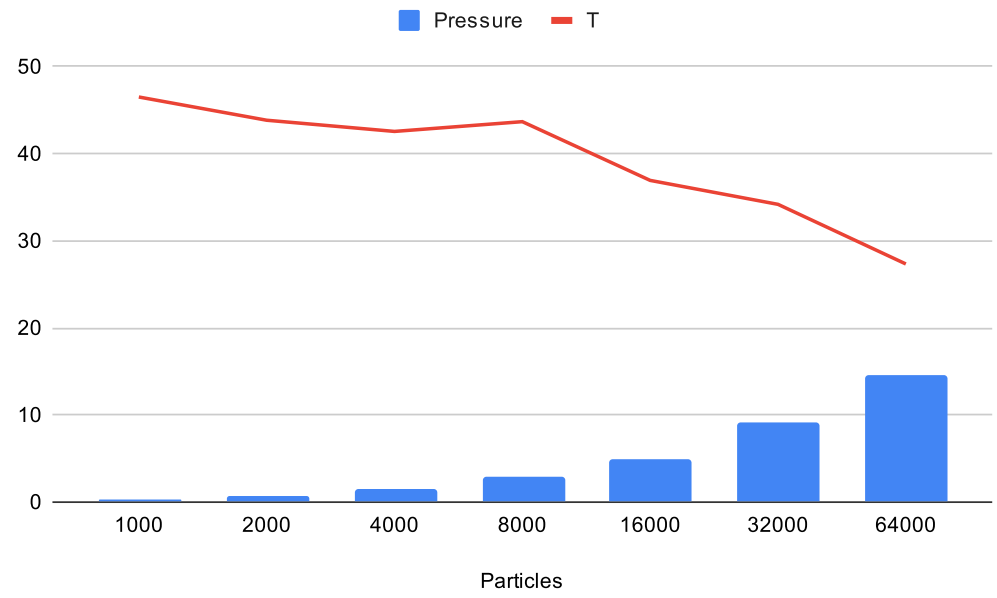

Figure 6: Relation between particle amount and temperature, with size 1000.

The simulation seems to follow the law quite well except when the ratio of particles to size is large.

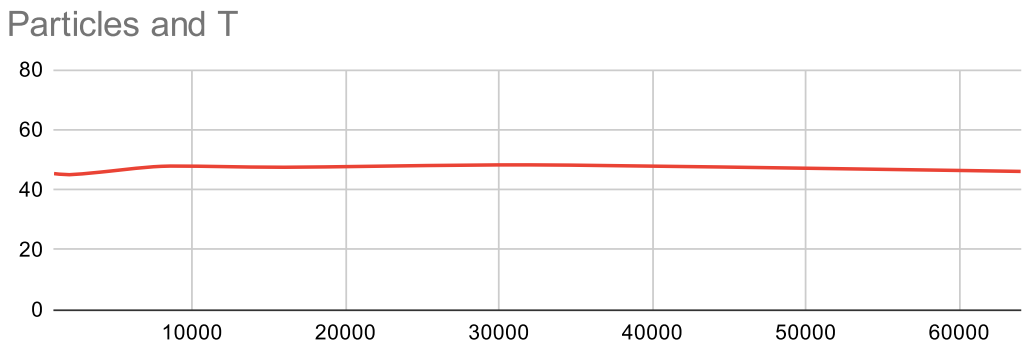

Figure 7: Relation between particle amount and temperature, with size 10000.

With a smaller particle to size ratio, the ideal gas law is almost perfectly followed with a very straight line observed in figure 7.

5.2 Speedup

Speedup is measured with 32000 particles and 100 timesteps, over an exponent of two processes.

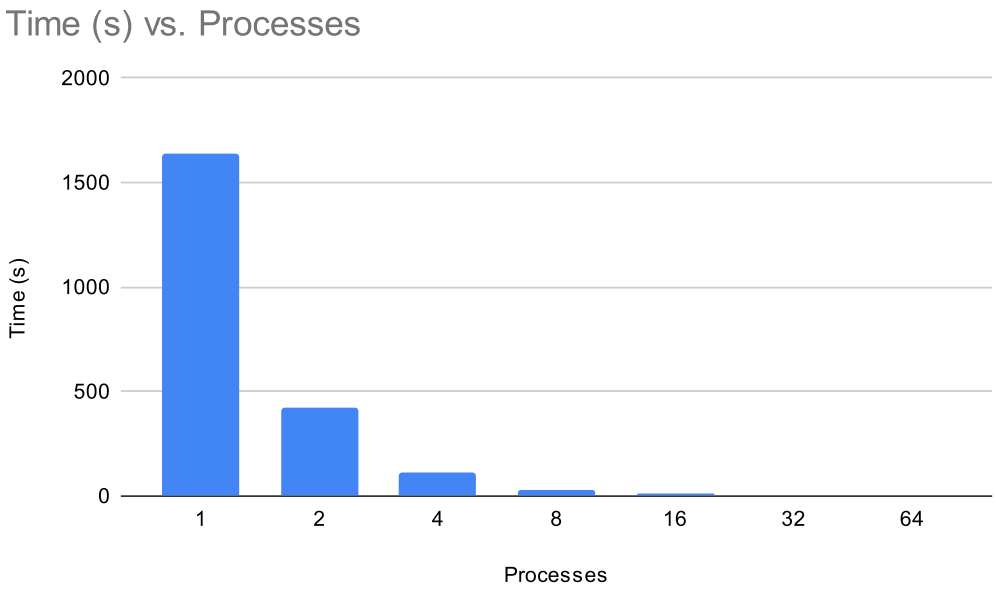

Figure 8: Relation between execution time and process amount, with 32000 particles.

Table 1: Relation between execution time and process amount, with 32000 particles.

Processes	**1**	**2**	**4**	**8**	**16**	**32**	**64**
Time (s)	1636.98	422.378	110.662	28.0934	8.29144	2.23403	0.595723

The speedup of the program from using multiple processes can be seen in figure 8 and table 1. The speedup is superlinear, with a speedup of almost fourfold when the amount of processes is doubled.

5.3 Linear vs Linked List

A comparison of two different data structures can be seen in table 2, here it can be seen that the execution time is significantly faster when using std::vector.

Table 2: Comparison of performance of std::forward_list and std::vector, with 32000 particles.

Processes	**8**	**16**	**32**	**64**
Time (s) std::forward_list	47.7115	12.7871	3.8445	0.998647
Time (s) std::vector	28.0934	8.29144	2.23403	0.595723

5.4 Grid vs rows

We measured how many particles are sent depending on if the layout is row or grid.

Table 3: Comparison of particles sent each timestep for different layouts.

Particles	Particles	Average sent particles
1x64	100000	9782
8x8	100000	2190
1x64	50000	5155
8x8	50000	1083

Table 3 confirms that choosing a grid over rows results in less communication.

6 Discussion

Ideal gas law, superlinear speedup and memory layout will be discussed.

6.1 Ideal gas law

The simulation follows the ideal gas law. As seen in figure 5 the temperature remain almost constant as size increases and pressure drops. With one exception, being that for small sizes, the temperature is lower than it should. This is likely due to that in a smaller space with many particles, there will be a lot more collision, due to our limitations, if a particle collides with another particle, it can not collide again in that timestep, resulting in less pressure.

In figure 6, it can be observed that using many particles with a small size is not as accurate as using a larger size , as seen in figure figure 7.

6.2 Superlinear speedup

If running in 1 processes, $t(n, 1) = n^2$ operations are needed to check all collisions and complete one simulation timestep.

If running in p processes, the particles is most often divided into n/p particles per process. This would result in the operations per processor being $t(n, p) =$

$(n/p)^2$.

The relative speedup is thus $S_{rel} = \frac{t(n,1)}{t(n,p)} = \frac{n^2}{n^2/p^2} = p^2$.

This matches what we see in table 1, with $T(1) = 1636.98$ and $T(2) = 422.378$, resulting in a relative speedup of $T(1)/T(2) = 3.87 \approx 2^2$

6.3 Linear vs Linked List

We initially thought the linked list would have better performance, considering its cheaper insertion and deletion, which each simulation timestep has a lot of. Although the more expensive operations of vector seemed to be negligible due to the major performance increase that was gained from its cache locality.

While analyzing the performance with ITAC, the linked list had more consistent computation time between communication steps across processes, since all operations are similarly expensive O(1), while on a vector, the operations vary from O(1) to O(n).

6.4 Grid vs rows

Grid sends less particles per timestep due to its higher ratio between area and circumference. However rows communicates with fewer neighbors. This means that if the communication overhead is relatively expensive, rows can be better, while grid performs better with more expensive communication.

7 Conclusion

In conclusion the model seems to be an alright estimation of the ideal gas law. It also benefits largely from parallelization. We were initially surprised by the superlienear speedup, but after looking into the theory of the course it made sense. This project also helped up learn that there are good debuggers for parallel programs, as well as how to use them. It also helped us learn how to use a traceanalyzer tool to further improve parallel programs. This project also helped us learn about the importance of cache locality, even if some other optimizations are sacrificed. This was ## References

References

1. Message Passing Interface :: High Performance computing. (n.d.). https://hpc.nmsu.edu/discovery/mpi/introduction/