CSE 587 FA2024: Parallel Computing
Assignment 2: Report on MPI program

Yichen Lu nechy@umich.edu

October 1, 2024

1 Verification

Matrix Size ($m \times n$)	Processes (p)	Execution Time (s)	Speedup	Sum	Sum Square
2000 × 500	1	32.735807	1 (Unit)	3.01232e+07	6.85952e+09
2000 × 500	4	8.308352	3.94	3.01232e+07	6.85952e+09
2000 × 500	16	2.062765	15.87	3.01232e+07	6.85952e+09
2000 × 500	36	0.921060	35.54	3.01232e+07	6.85952e+09
1000 × 4000	1	129.032864	1 (Unit)	1.1929e+08	1.41824e+10
1000 × 4000	4	32.313758	3.99	1.1929e+08	1.41824e+10
1000 × 4000	16	8.117760	15.90	1.1929e+08	1.41824e+10
1000 × 4000	36	4.003438	32.23	1.1929e+08	1.41824e+10

Table 1: Matrix Size, Processes, Execution Time, Speedup and Sums

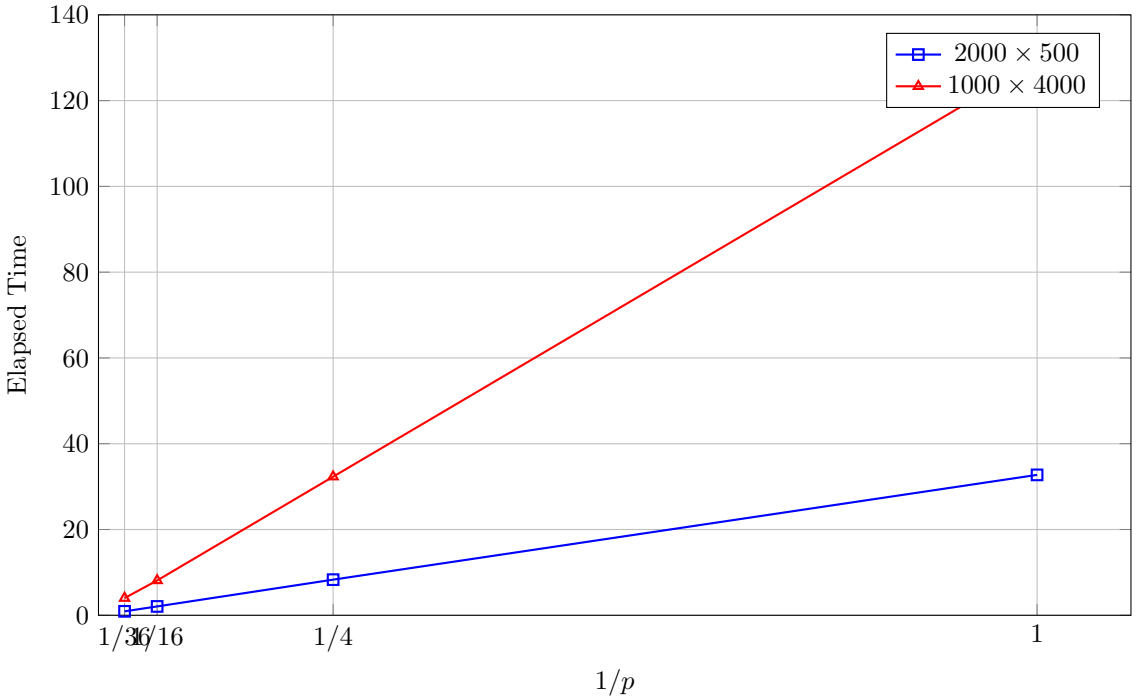

Figure 1: Execution Time and Number of Processors

2 Description

2.1 Matrix Decomposition

We know that the input matrix is 2D and the number of cores used for processing the computation is a perfect square. Therefore, we assume the size of the matrix is $m \times n$, and the core processes are arranged in a $q \times q$ grid, where $q = \sqrt{p}$ and

p is the total number of processes. Each process is identified by its grid coordinates (row_rank, col_rank), calculated as:

$$\text{row\_rank} = \left\lfloor \frac{\text{rank}}{q} \right\rfloor, \quad \text{col\_rank} = \text{rank}\%q$$

Focued on each processor, I decomposed the matrix into blocks of size local_m × local_n. To handle cases where m and n are not perfectly divisible by q, extra rows and columns are separately filled into the front processes.

Rows:

$$\text{local\_m} = \left\lfloor \frac{m}{q} \right\rfloor + \begin{cases} 1, & \text{if row\_rank} < m\%q \\ 0, & \text{otherwise} \end{cases}$$

Columns:

$$\text{local\_n} = \left\lfloor \frac{n}{q} \right\rfloor + \begin{cases} 1, & \text{if col\_rank} < n\%q \\ 0, & \text{otherwise} \end{cases}$$

For example, in the case of $m = 2000$, $n = 500$, and $p = 4$:

- Processes is a 2×2 grid
- Each process handles 1000×250 entries

2.2 Communication

The matrix update is governed by the following equation:

$$z = \frac{f\left(A_o(i-1, j-1)\right) + f\left(A_o(i-1, j+1)\right) + f\left(A_o(i+1, j-1)\right) + f\left(A_o(i+1, j+1)\right) + f\left(A_o(i, j)\right)}{5}$$

Each process's local matrix is extended by adding an extra row and column on each side (top, bottom, left, right) as ghost layers, store boundary data received from adjacent processes.

- **Horizontal Communication**:
  - **Sending**: Each process sends its rightmost data column to the left ghost layer of its right neighbor.
  - **Receiving**: Each process receives the leftmost data column from its left neighbor and stores it in its left ghost layer.
- **Vertical Communication**:
  - **Sending**: Each process sends its bottommost data row to the top ghost layer of its bottom neighbor.
  - **Receiving**: Each process receives the topmost data row from its top neighbor and stores it in its top ghost layer.

An `MPI_Barrier` is employed after initializing the matrix in every processor and at the end of each iteration. This ensures that all processes have completed their communication before proceeding to the next iteration, and the computation in the subsequent iteration operates on the most recent data.

2.3 Time Result

Figure 1 shows that the execution time (s) has a linear relationship with the inverse of the number of processors, which is validated by the speedup equation:

$$\textit{Speedup} = \frac{\text{Time with one processor}}{\text{Time with } p \text{ processors}} \approx p$$

Besides, the speedup ratio for the 2000×500 matrix is higher than that for the 1000×4000 matrix. This indicates that efficiency decreases as the number of processes increases, particularly for smaller matrix sizes.