Serial Approach

Using OpenMP
1D Decomposition
2D Decomposition

Using MPI
1D Decomposition

Parallelizing the Jacobi Iteration Algorithm

Alberto Rodriguez

Higher Institute of Technologies and Applied Sciences (InSTEC)

October 14, 2016

Outline

1. Serial Approach
2. Using OpenMP
   - 1D Decomposition
   - 2D Decomposition
3. Using MPI
   - 1D Decomposition

Serial Approach

Analyzing the Code

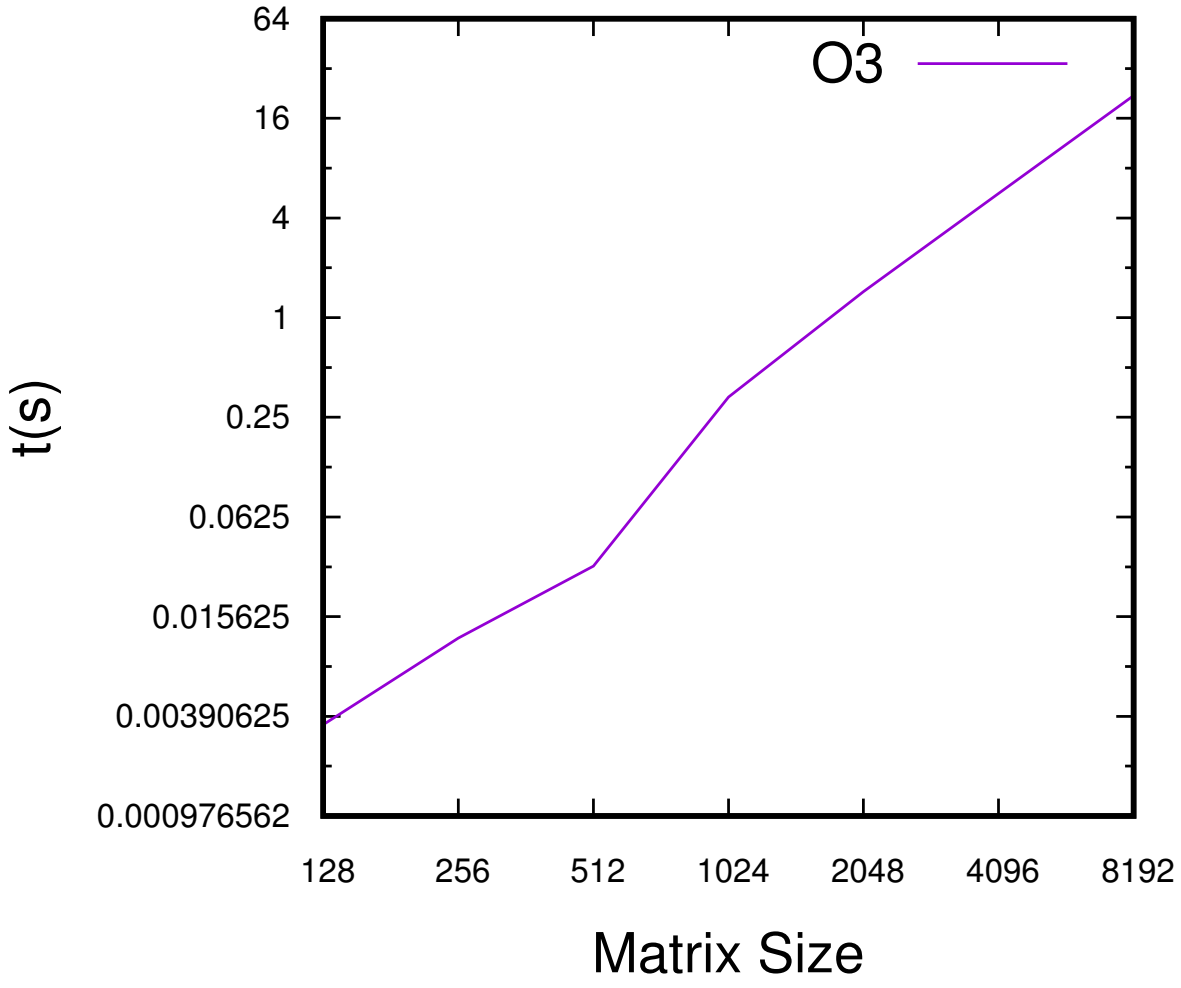
With a serial code we have:

- Time Complexity $O(I * N^2)$ (Nested loops to compute the matrix for each iteration).
- Memory Complexity $O(N^2)$ (Size of the matrix).

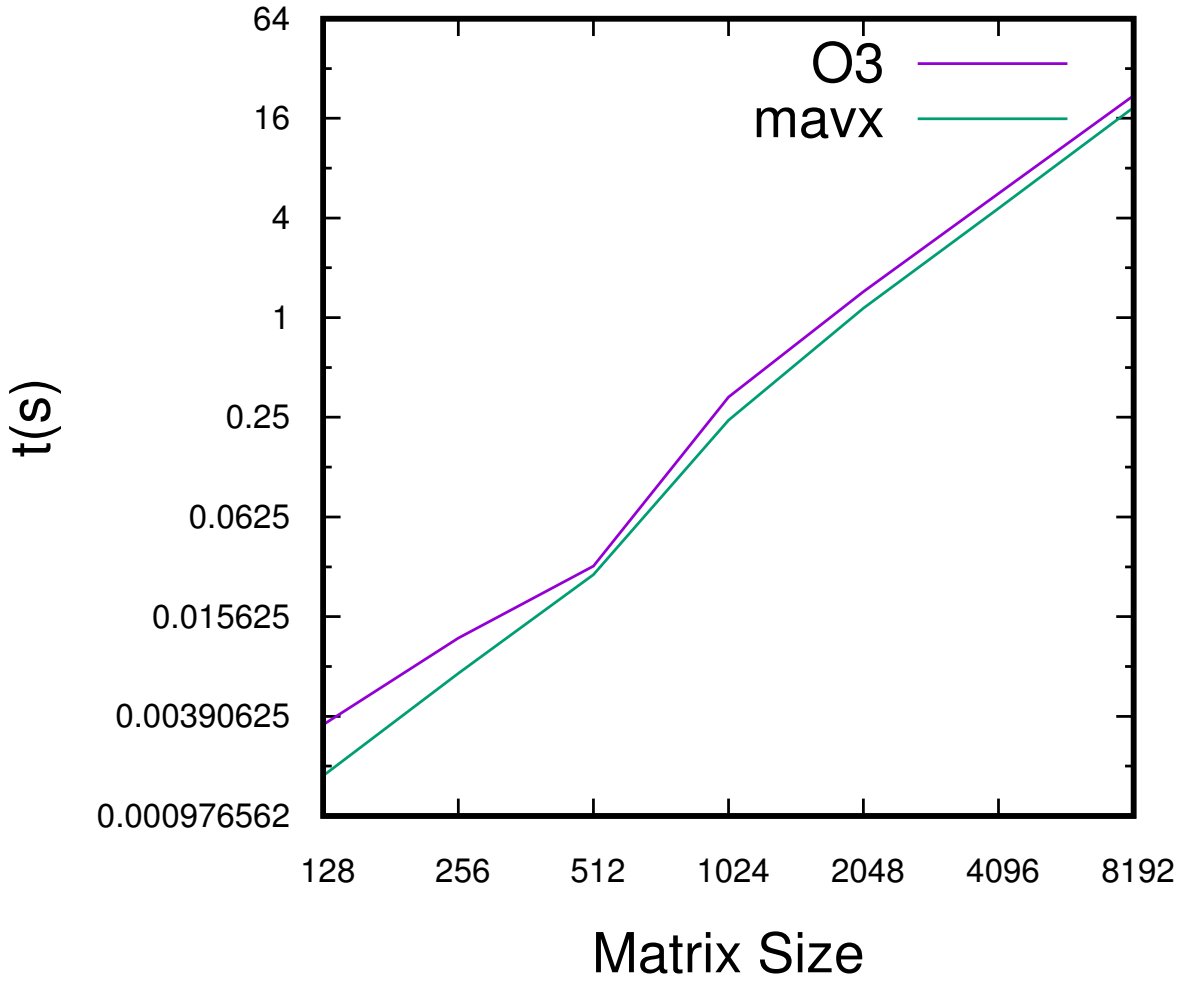
We expect:

- Long times to process matrix of bigger size and/or great amount of iterations.
- Size of the matrix is limit by system (Size of the memory of the node) .

Results

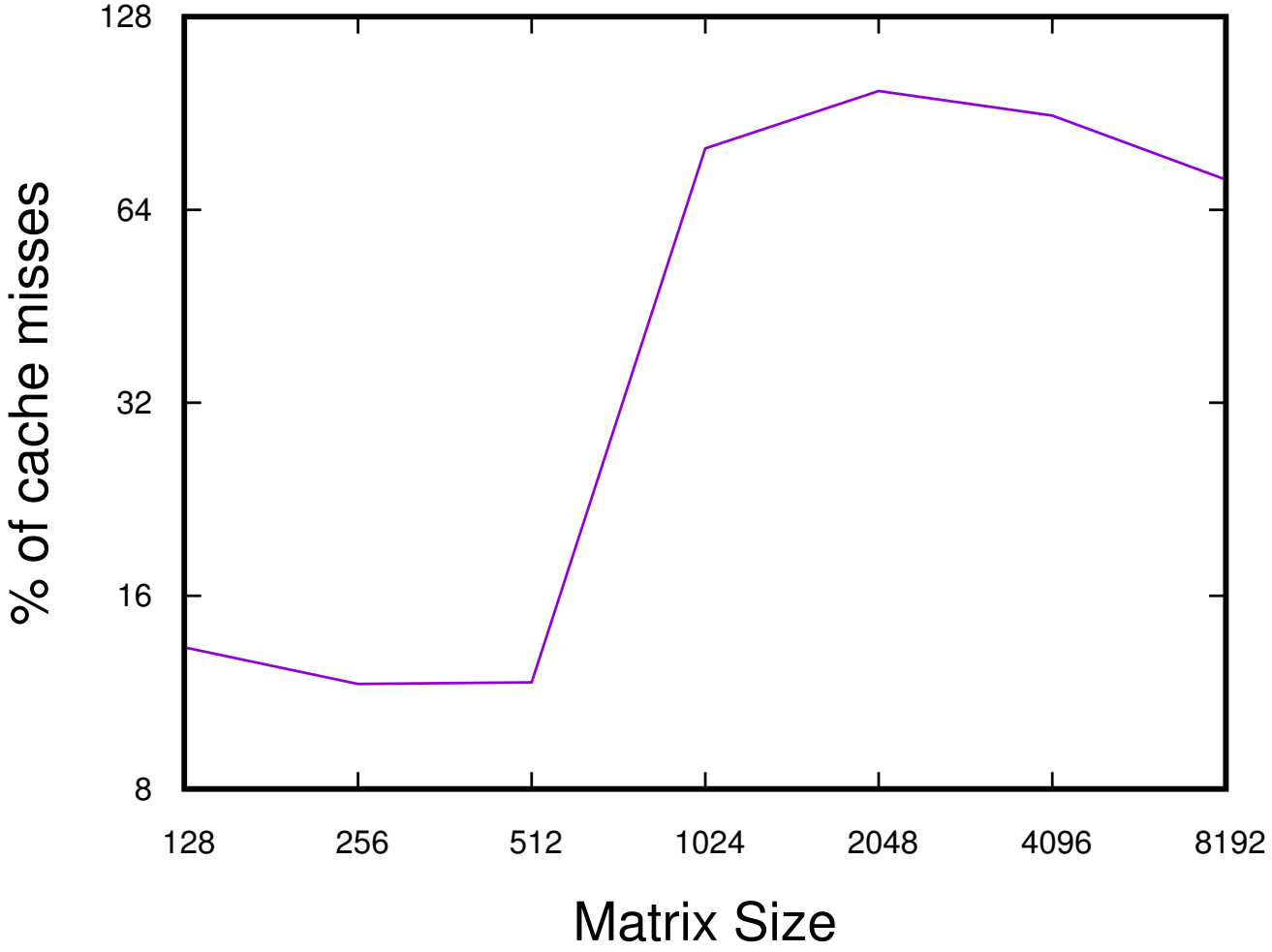

O3
t(s)
64
16
4
1
0.25
0.0625
0.015625
0.00390625
0.000976562
128
256
512
1024
2048
4096
8192
Matrix Size

Results


64
16
4
1
0.25
0.0625
0.015625
0.00390625
0.000976562
t(s)
O3
mavx
128
256
512
1024
2048
4096
8192
Matrix Size

Results

Data obtained from perf tool.



Using OpenMP

What part of the code need parallelization?

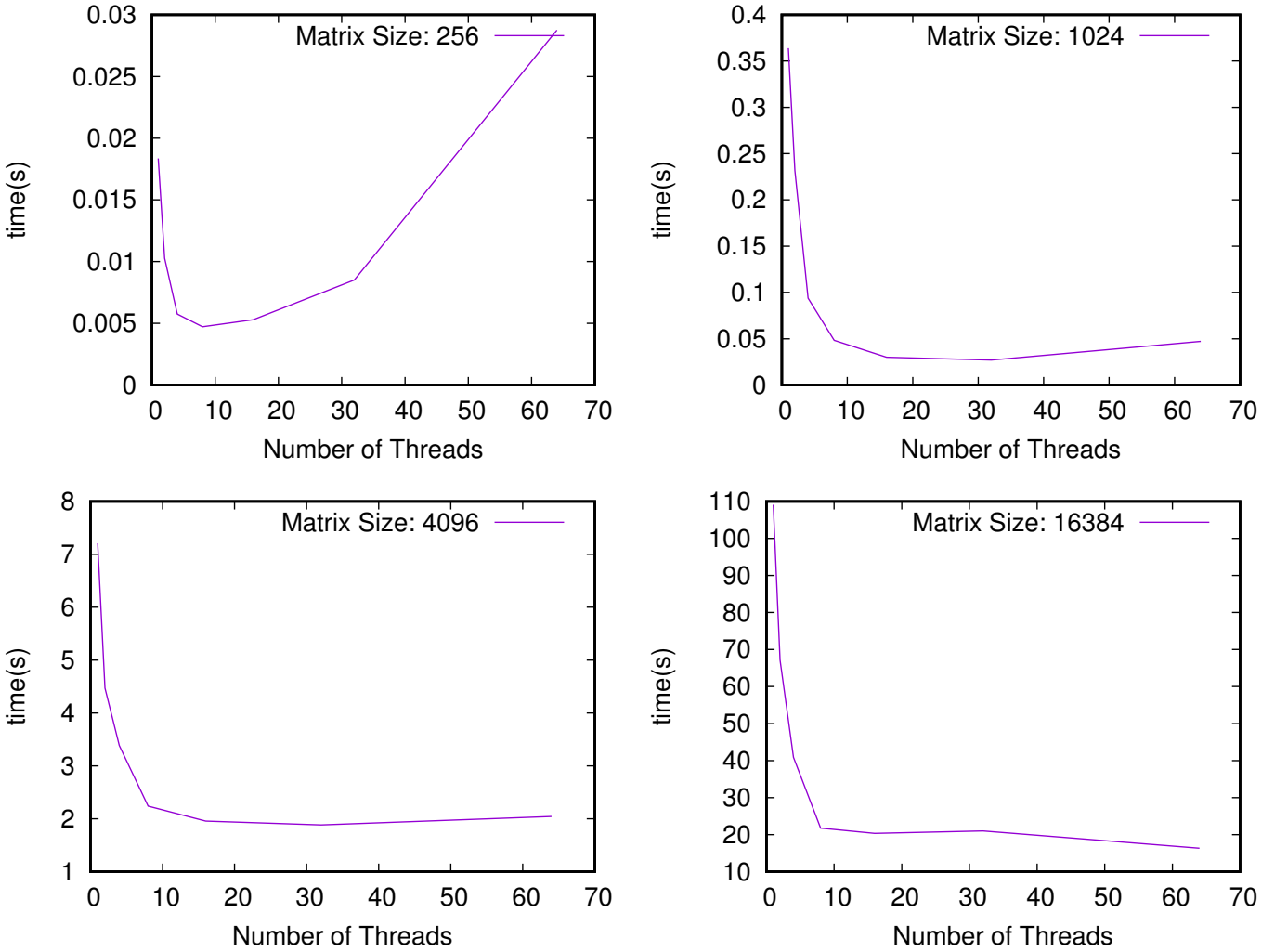
$$
\begin{aligned}
&for(iCount = 1; iCount <= Iterations; iCount++)\\
&\quad for(i = 0; i < Dimension; i++)\\
&\qquad for(j = 0; j < Dimension; j++)\\
&\qquad\qquad \{...\}
\end{aligned}
$$

- The outer loop can't be parallelize because the new matrix depends to the old matrix.
- We can parallelize the inner loops.

Instructions using OpenMP

$$
\begin{aligned}
&for(iCount = 1; iCount <= Iterations; iCount++)\\
&\quad \#pragma\ \ omp\ \ parallel\ \ for\ \ private(i, j)\\
&\quad for(i = 0; i < Dimension; i++)\\
&\qquad for(j = 0; j < Dimension; j++)\\
&\qquad\qquad \{...\}
\end{aligned}
$$

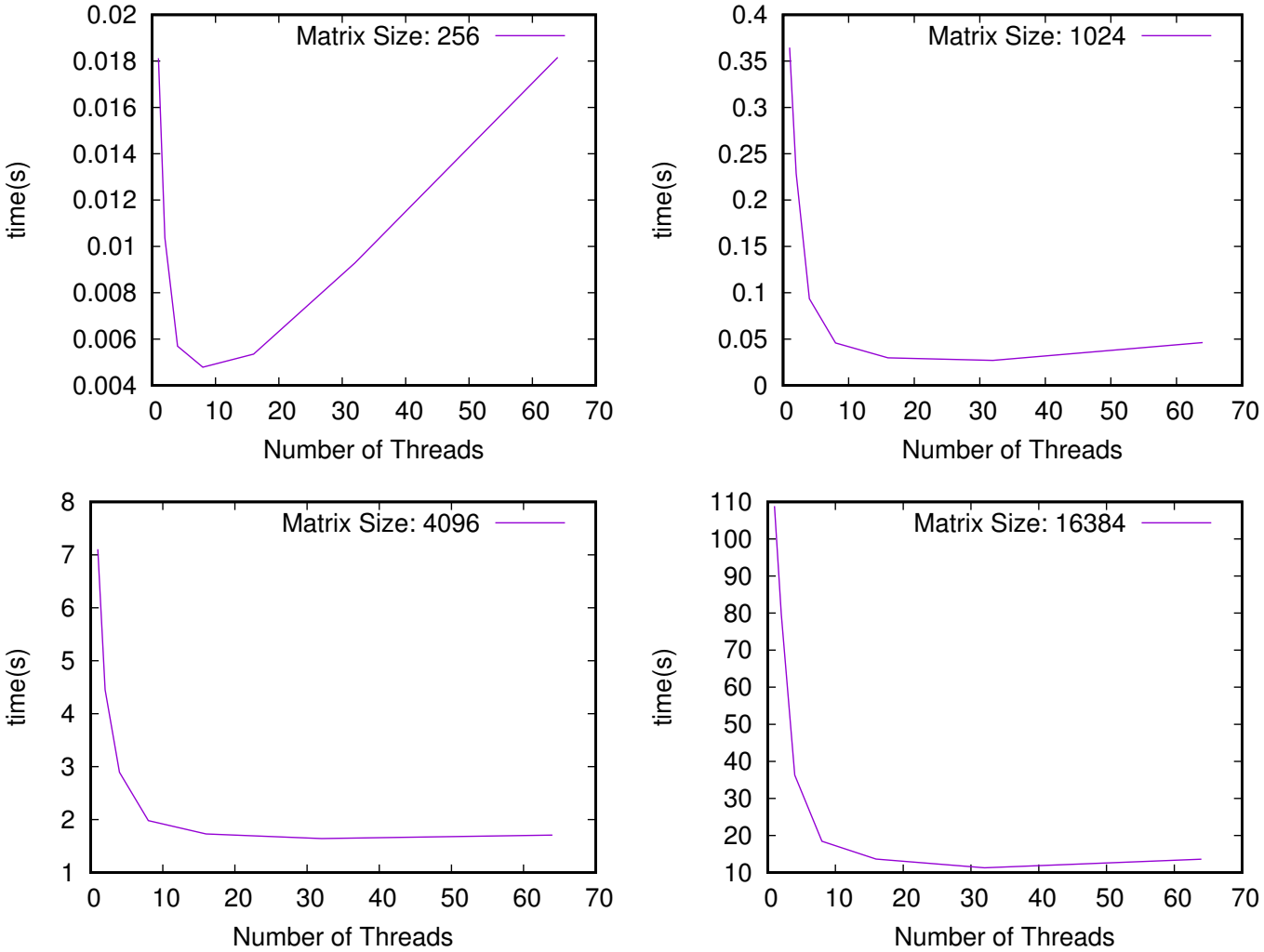
Results


Matrix Size: 256
time(s)
Number of Threads
Matrix Size: 1024
time(s)
Number of Threads
Matrix Size: 4096
time(s)
Number of Threads
Matrix Size: 16384
time(s)
Number of Threads

Instructions using OpenMP

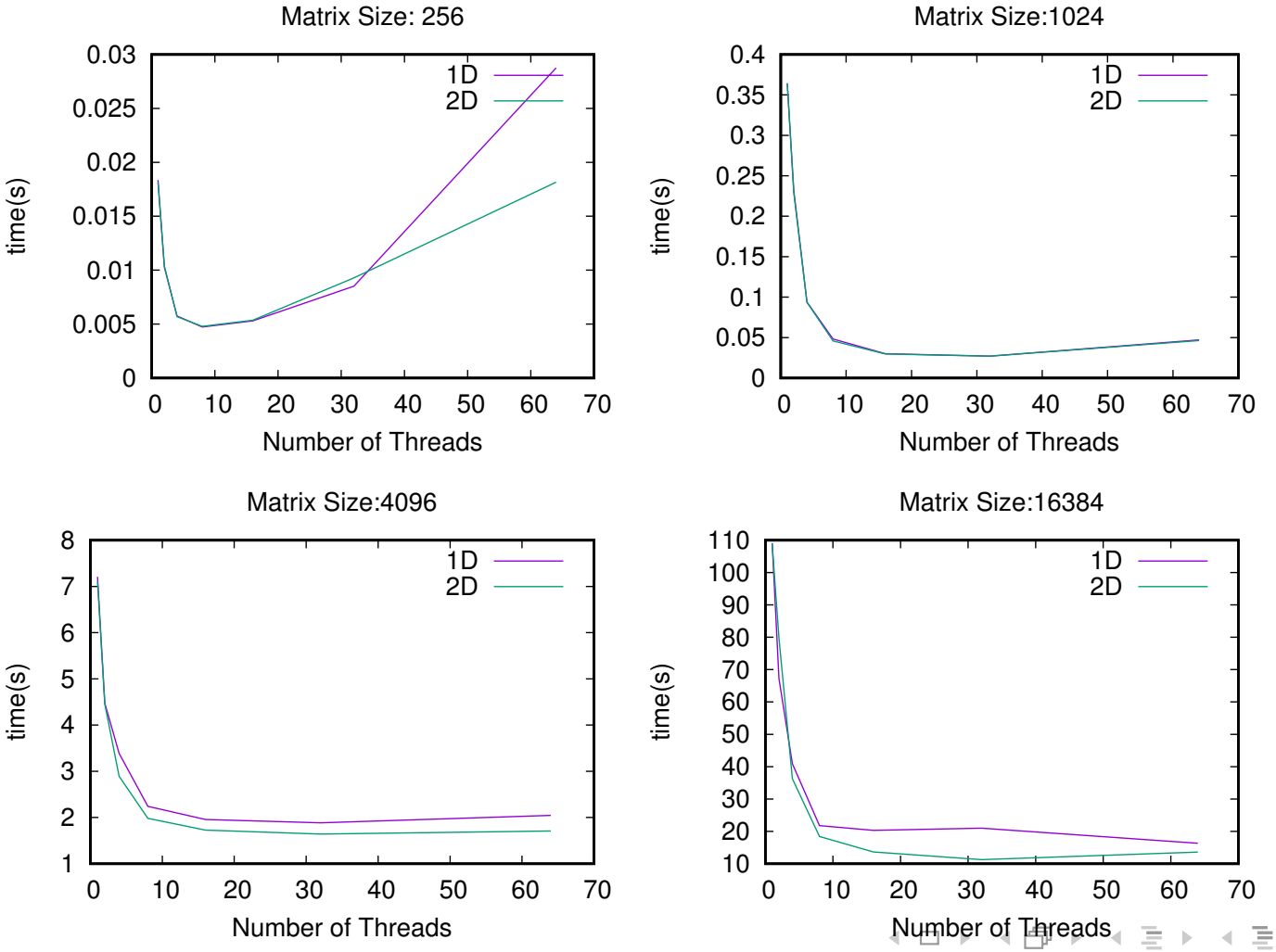
$$
\begin{aligned}
&for(iCount = 1; iCount <= Iterations; iCount++)\\
&\quad \#pragma\ omp\ parallel\ for\ collapse(2)\ private(i, j)\\
&\quad for(i = 0; i < Dimension; i++)\\
&\qquad for(j = 0; j < Dimension; j++)\\
&\qquad\qquad \{...\}
\end{aligned}
$$

Results


Matrix Size: 256
time(s)
Number of Threads
Matrix Size: 1024
time(s)
Number of Threads
Matrix Size: 4096
time(s)
Number of Threads
Matrix Size: 16384
time(s)
Number of Threads

Results

Comparison between 1D and 2D decomposition.



Using MPI

How to divide the matrix

We want:

- Minimize cache misses:
  - Divide the matrix by groups of rows. (C)
- Every process has roughly the same amount of job:
  - Every process will have the same amount of rows.
  - If that is no posible the last ones are going to have 1 more row.
- Every process access some data that is store in the previous process and in the next one:
  - We need to allocate 2 more rows for each process. (Ghost or boundary cells)

How to divide the matrix

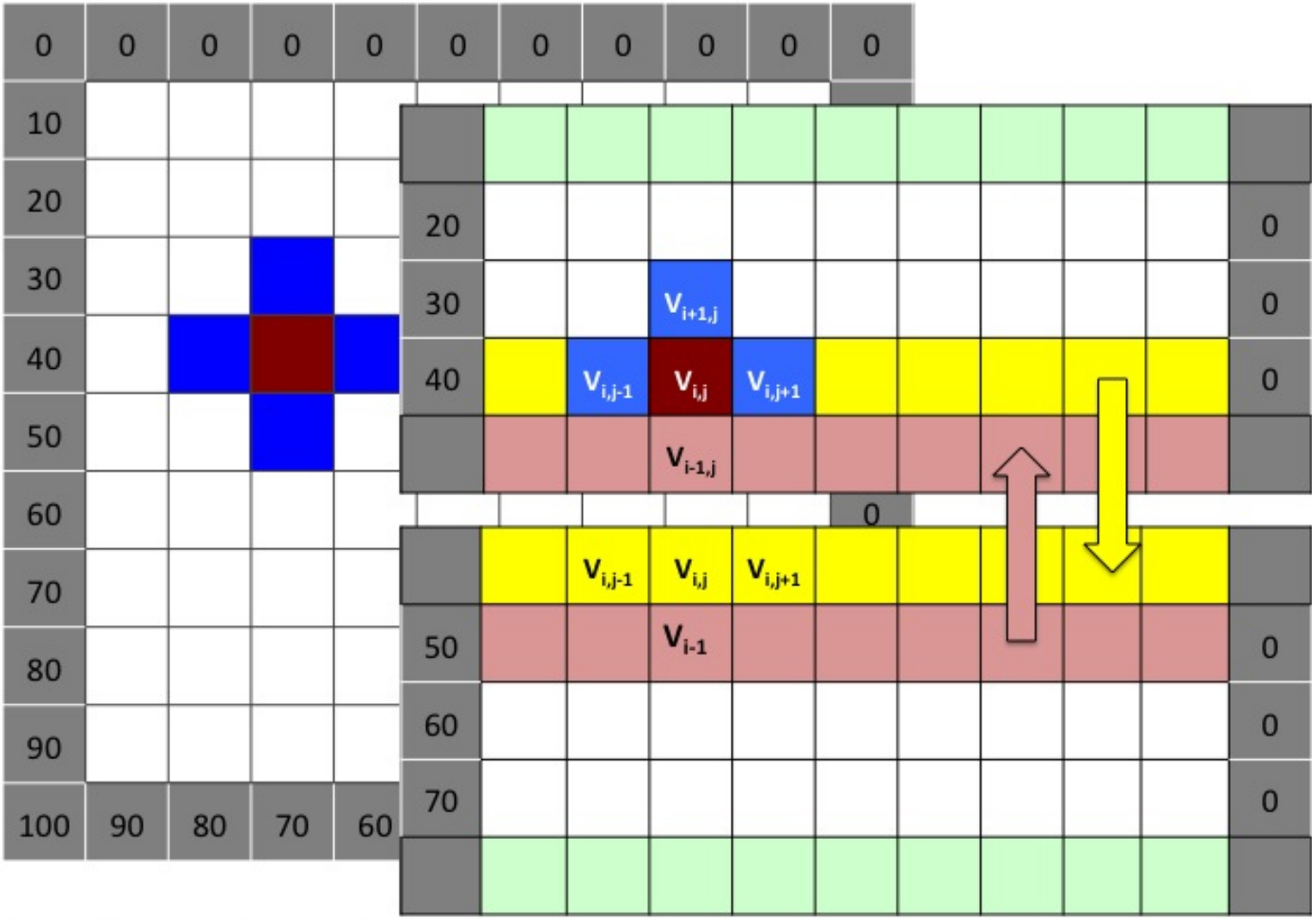


Figure 2: A diagram of the 1-D decomposition of the Jacobi Relaxation for Solving the Laplace's Equation, showing that the boundary elements that need to be communicated between processors.

Communications

How many communications we are going to have between processes?

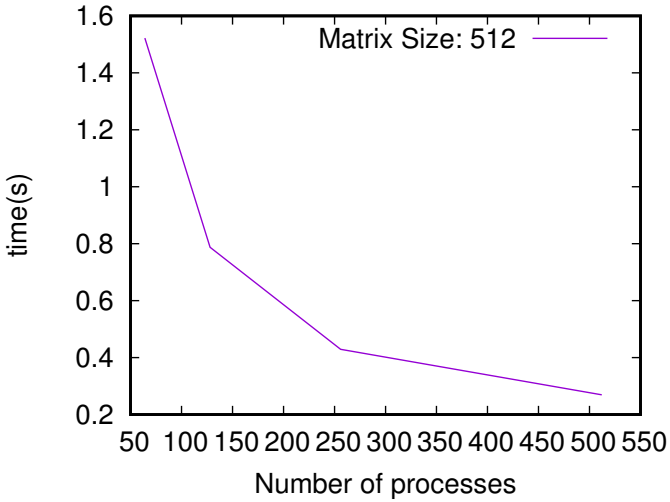
- Every process (except the first an the last ones) needs to send data to the previous process and to the next one.
- Every process (except the first an the last ones) needs to receive data from the previous process and the next one in order to update its ghost cells.

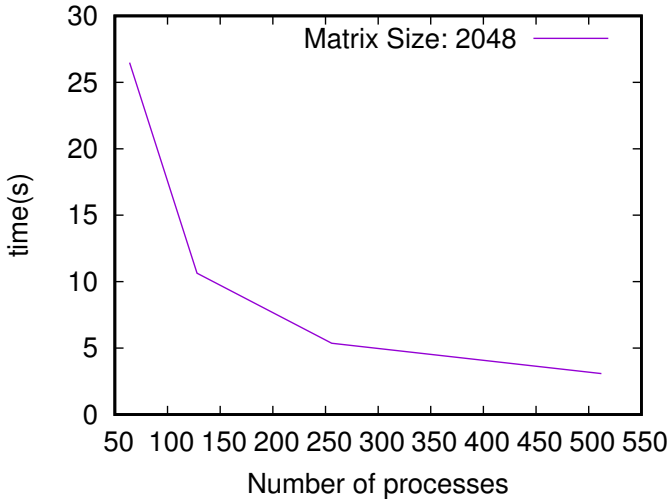
Communications

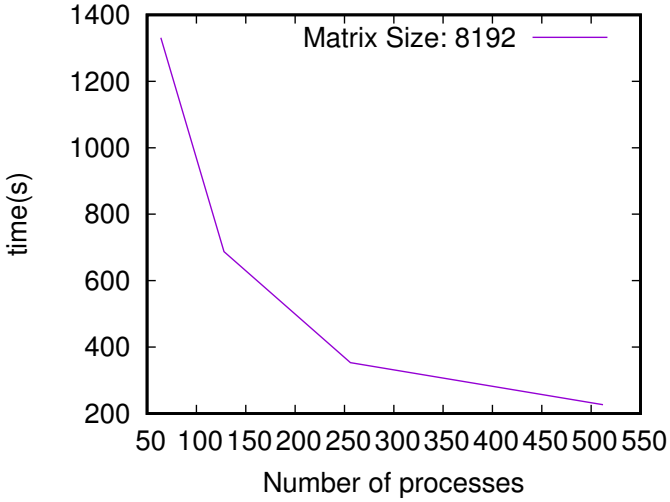
How many values are communicated per iteration?

- The program has N processes.
- Every process needs to communicate four times.
- In every communication an entire row is communicated.
- Every row has M elements.
- Math: $N * 4 * 1 * M = 4 * N * M$

Results


Matrix Size: 512
time(s)
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
50 100 150 200 250 300 350 400 450 500 550
Number of processes


Matrix Size: 2048
time(s)
30
25
20
15
10
5
0
50 100 150 200 250 300 350 400 450 500 550
Number of processes


Matrix Size: 8192
time(s)
1400
1200
1000
800
600
400
200
50 100 150 200 250 300 350 400 450 500 550
Number of processes

Results

Comparison between AMD and Intel machines.