Matrix Inversion using Parallel Gaussian Elimination

CSE 633 Parallel Algorithms (Spring 2014)
Aravindhan Thanigachalam
Email: athaniga@buffalo.edu

Instructor: Dr. Russ Miller

Outline

- Problem
- Example
- Sequential Algorithm
- Leveraging Parallelism
- Row Oriented Distribution
- Column Oriented Distribution
- Performance Results
- Blunders
- Future Scope
- References

Problem

Given a $n \times n$ matrix A, determine the inverse of the matrix denoted by A^{-1}

$$A \times B = B \times A = I_n \Rightarrow B = A^{-1}$$

Elementary Row Operations:

- Interchange distinct rows of A
- Multiply a row of A by a non zero constant $c \neq 0$
- Add a constant multiple of row i to row j , where $i \neq j$

We know that if a sequence σ of such operations applied to A transforms it into I_n then the same sequence σ applied to I_n transforms it into A^{-1}.

Thus, we can find A^{-1} by finding such a sequence σ that transforms the augmented matrix $[\, A \mid I_n \,]$ *to* $[\, I_n \mid A^{-1} \,]$

Example

$$A = \begin{bmatrix} 5 & -3 & 2 \\ -3 & 2 & -1 \\ -3 & 2 & -2 \end{bmatrix} \qquad [A \,|\, I_3] = \left[\begin{array}{ccc|ccc} 5 & -3 & 2 & 1 & 0 & 0 \\ -3 & 2 & -1 & 0 & 1 & 0 \\ -3 & 2 & -2 & 0 & 0 & 1 \end{array}\right]$$

Gaussian Elimination Phase: $a_{i,i} = 1,\ 1 \le i \le n$, and $a_{i,j} = 0,\ 1 \le j < i \le n$

1. Divide row 1 by 5

$$\left[\begin{array}{ccc|ccc} 1 & -0.6 & 0.4 & 0.2 & 0 & 0 \\ -3 & 2 & -1 & 0 & 1 & 0 \\ -3 & 2 & -2 & 0 & 0 & 1 \end{array}\right]$$

Example (continued)

2. Add 3 times row 1 to row 2, and 3 times row 1 to row 3

$$\left[\begin{array}{ccc|ccc} 1 & -0.6 & 0.4 & 0.2 & 0 & 0 \\ 0 & 0.2 & 0.2 & 0.6 & 1 & 0 \\ 0 & 0.2 & -0.8 & 0.6 & 0 & 1 \end{array}\right]$$

3. Divide row 2 by 0.2

$$\left[\begin{array}{ccc|ccc} 1 & -0.6 & 0.4 & 0.2 & 0 & 0 \\ 0 & 1 & 1 & 3 & 5 & 0 \\ 0 & 0.2 & -0.8 & 0.6 & 0 & 1 \end{array}\right]$$

Example (continued)

4. Subtract 0.2 times row 2 from row 3

$$\left[\begin{array}{ccc|ccc} 1 & -0.6 & 0.4 & 0.2 & 0 & 0 \\ 0 & 1 & 1 & 3 & 5 & 0 \\ 0 & 0 & -1 & 0 & -1 & 1 \end{array}\right]$$

5. Divide row 3 by -1

$$\left[\begin{array}{ccc|ccc} 1 & -0.6 & 0.4 & 0.2 & 0 & 0 \\ 0 & 1 & 1 & 3 & 5 & 0 \\ 0 & 0 & 1 & 0 & 1 & -1 \end{array}\right]$$

Example (continued)

Back Substitution Phase:

1. Subtract 0.4 times row 3 from row 1, and 1 times row 3 from row 2

$$\left[\begin{array}{ccc|ccc} 1 & -0.6 & 0 & 0.2 & -0.4 & 0.4 \\ 0 & 1 & 0 & 3 & 4 & 1 \\ 0 & 0 & 1 & 0 & 1 & -1 \end{array}\right]$$

2. Add 0.6 times row 2 to row 1

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 2 & 2 & 1 \\ 0 & 1 & 0 & 3 & 4 & 1 \\ 0 & 0 & 1 & 0 & 1 & -1 \end{array}\right]$$

$$A^{-1} = \begin{bmatrix} 2 & 2 & 1 \\ 3 & 4 & 1 \\ 0 & 1 & -1 \end{bmatrix}$$

Sequential Algorithm

Gaussian Elimination Phase:

1. For $i = 1$ to n, do

a) If $A[i,i] = 0$ and $A[m,i] = 0$ for all $m > i$, conclude that A^{-1} does not exist and halt the algorithm.

b) If $A[i,i] = 0$ and $A[m, i] \neq 0$ for some smallest $m > i$, interchange rows i and m in the array A and in the array I.

c) Divide row i of A and row i of I by $A[i, i]$. That is, let $scale = A[i, i]$ and then for $j = 1$ to n, replace $A[i, j]$ by $A[i, j]/scale$.

d) Now we have $A[i, i] = 1$. If $i < n$, then for $r > i$, subtract $A[r, i]$ times row i from row r in both the arrays A and I.

Sequential Algorithm (continued)

Gaussian elimination phase:

```
For i = 1 to n
    If A[i,i] = 0, then
        Swap row with the nearest subsequent row such that after swapping A[i,i] ≠ 0
        If no such row exists then EXIT 'INVERSE DOES NOT EXIST'
    scale ← A[i,i]
    For col = 1 to n
        A[i,j] ← A[i, j]/scale
        I[i,j] ← I[i, j]/scale
    End For col
    If i < n, then
        For row = i + 1 to n
            factor ← A[row,i]
            For col = 1 to n
                A[row,col] ← A[row,col] − factor × A[i,col]
                I[row,col] ← I[row,col] − factor × I[i,col]
            End For col
        End For row
    End If
End For i
```

Sequential Algorithm (continued)

Back Substitution Phase:

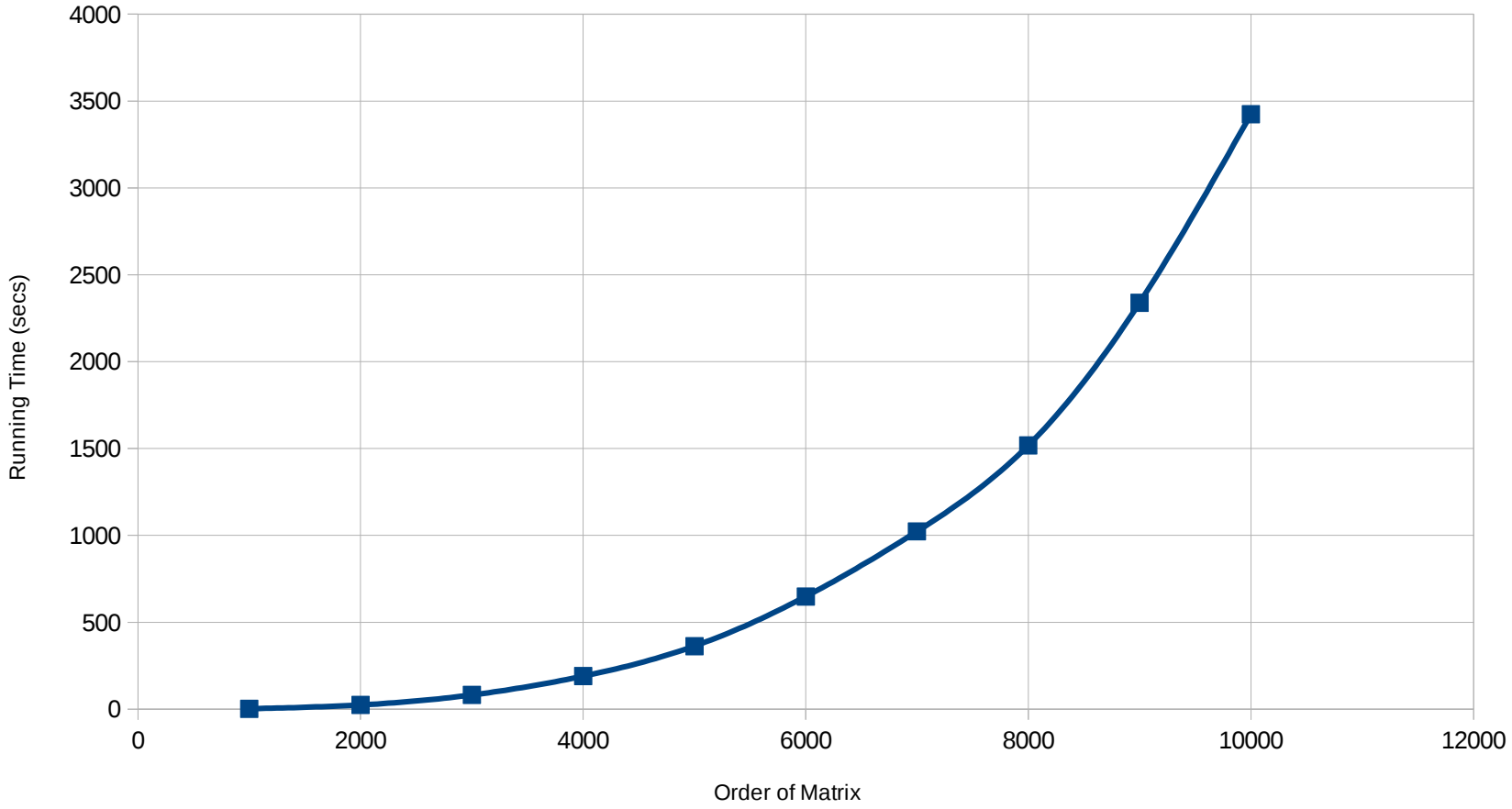
```
For zeroingCol = n downto 2
    For row = zeroingCol − 1 downto 1
        factor ← A[row,zeroingCol]
        For col = 1 to n
            A[row,col] ← A[row,col] − factor × A[zeroingCol,col]
            I[row,col] ← I[row,col] − factor × I[zeroingCol,col]
        End For col
    End For row
End For zeroingCol
```

Total Sequential Running Time => $O(n^3)$

Sequential Running Time

Order of Matrix	Running Time (seconds)
1000	1.89
2000	24.56
3000	81.93
4000	190.09
5000	363.02
6000	648.4
7000	1023.14
8000	1517.68
9000	2338.89
10000	3267.43

Sequential Running Time (cont..)



Leveraging Parallelism

Gaussian elimination phase:

```
For i = 1 to n      /** Inherently Sequential **/
    If A[i,i] = 0, then
        Swap row I with the nearest subsequent row j such that after swapping A[i,i] ≠ 0
        If no such row exists then EXIT 'INVERSE DOES NOT EXIST'
    scale ← A[i,i]
    For col = 1 to n
        A[i,j] ← A[i, j]/scale
        I[i,j] ← I[i, j]/scale
    End For col
    If i < n, then
        For row = i + 1 to n
            factor ← A[row,i]
            For col = 1 to n
                A[row,col] ← A[row,col] − factor × A[i,col]
                I[row,col] ← I[row,col] − factor × I[i,col]
            End For col
        End For row
    End If
End For i
```

Only with Column wise distribution

Outer for loop – row wise distribution

Inner for loop – column wise distribution

Data to be communicated

- Column wise distribution
- Row wise distribution

Leveraging Parallelism (continued)

Data to be communicated

- Column wise distribution
- Row wise distribution

Back Substitution Phase:

```
For zeroingCol = n downto 2      /** Inherently Sequential **/
    For row = zeroingCol − 1 downto 1
        factor ← A[row,zeroingCol]
        For col = 1 to n
            A[row,col] ← A[row,col] − factor × A[zeroingCol,col]
            I[row,col] ← I[row,col] − factor × I[zeroingCol,col]
        End For col
    End For row
End For zeroingCol
```

Outer for loop – row wise distribution

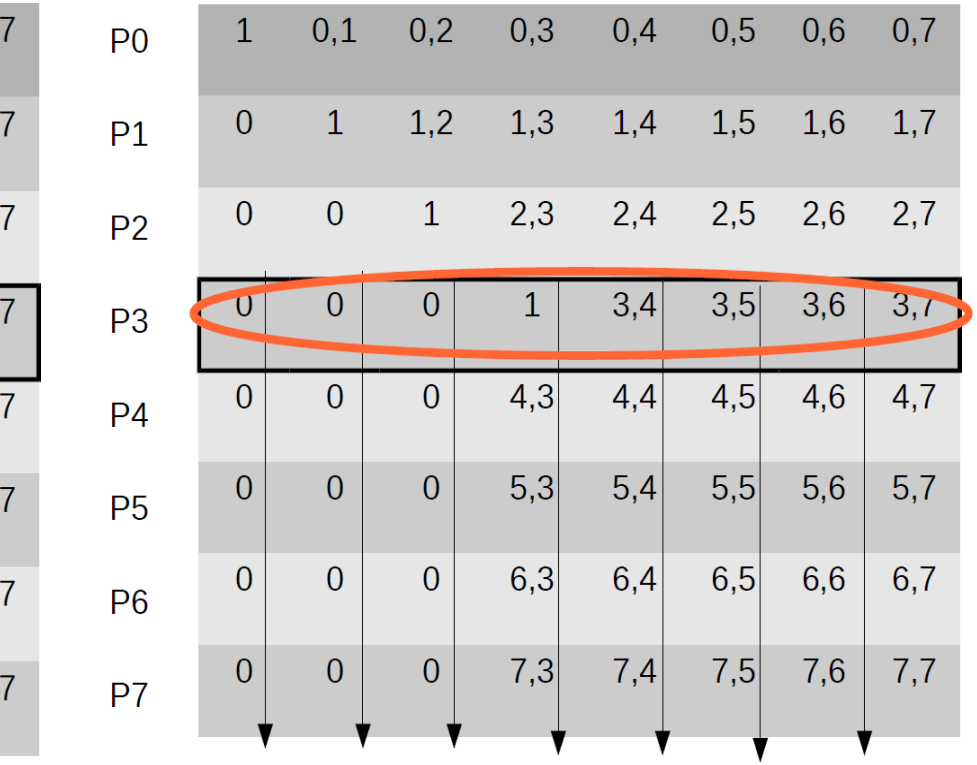
Inner for loop – column wise distribution

Parallel Running Time => $O(n^2)$

Row Oriented Distribution

Gaussian Elimination Phase: Iteration No. 4 Snapshot

P0	1	0,1	0,2	0,3	0,4	0,5	0,6	0,7
P1	0	1	1,2	1,3	1,4	1,5	1,6	1,7
P2	0	0	1	2,3	2,4	2,5	2,6	2,7
P3	0	0	0	3,3	3,4	3,5	3,6	3,7
P4	0	0	0	4,3	4,4	4,5	4,6	4,7
P5	0	0	0	5,3	5,4	5,5	5,6	5,7
P6	0	0	0	6,3	6,4	6,5	6,6	6,7
P7	0	0	0	7,3	7,4	7,5	7,6	7,7



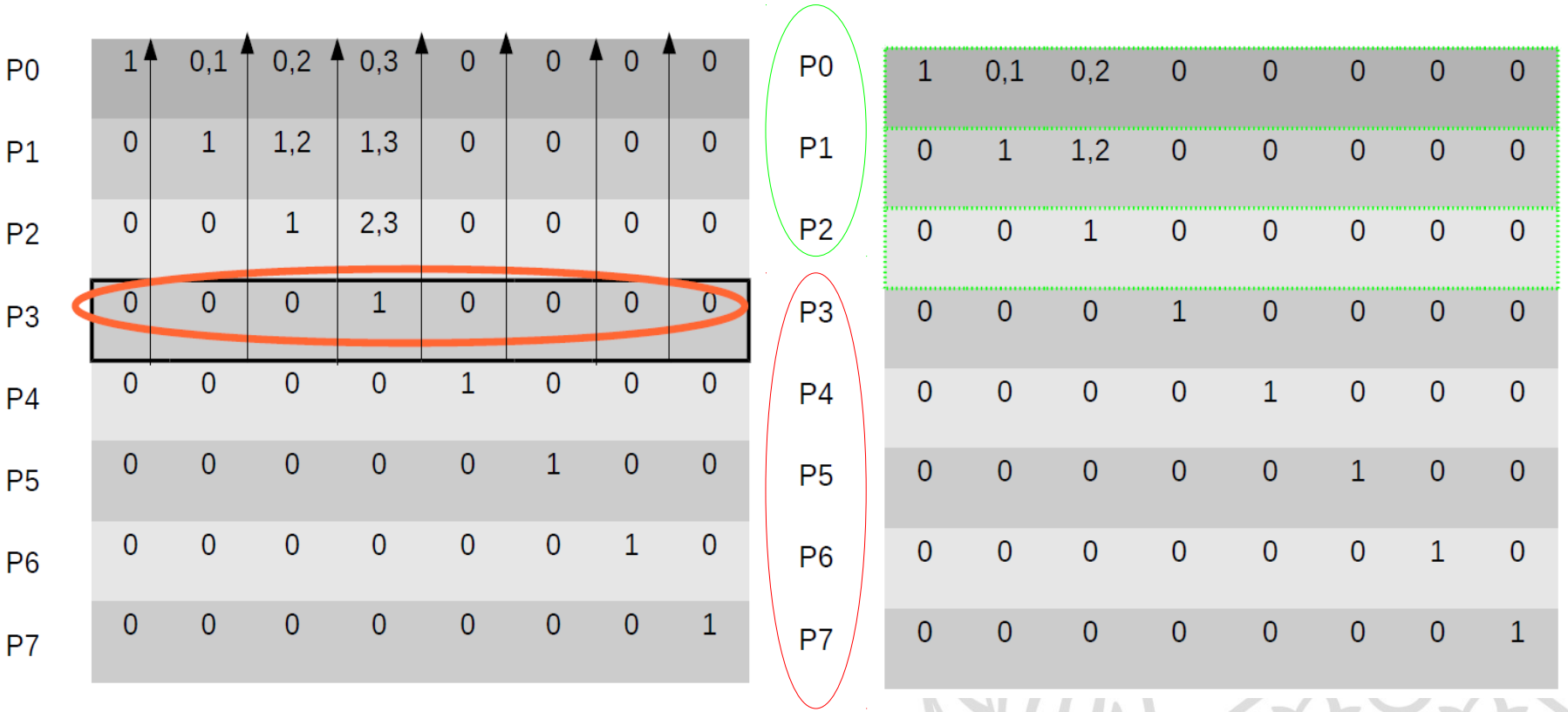
Row Oriented Distribution (cont)

Gaussian Elimination Phase: Iteration No. 4 Snapshot

P0	1	0,1	0,2	0,3	0,4	0,5	0,6	0,7
P1	0	1	1,2	1,3	1,4	1,5	1,6	1,7
P2	0	0	1	2,3	2,4	2,5	2,6	2,7
P3	0	0	0	1	3,4	3,5	3,6	3,7
P4	0	0	0	0	4,4	4,5	4,6	4,7
P5	0	0	0	0	5,4	5,5	5,6	5,7
P6	0	0	0	0	6,4	6,5	6,6	6,7
P7	0	0	0	0	7,4	7,5	7,6	7,7

Row Oriented Distribution (cont)

Back Substitution Phase: Iteration No. 5 Snapshot



Column Oriented Distribution

Gaussian Elimination Phase: Iteration No. 4 Snapshot

P0	P1	P2	P3	P4	P5	P6	P7
1	0,1	0,2	0,3	0,4	0,5	0,6	0,7
0	1	1,2	1,3	1,4	1,5	1,6	1,7
0	0	1	2,3	2,4	2,5	2,6	2,7
0	0	0	3,3	3,4	3,5	3,6	3,7
0	0	0	4,3	4,4	4,5	4,6	4,7
0	0	0	5,3	5,4	5,5	5,6	5,7
0	0	0	6,3	6,4	6,5	6,6	6,7
0	0	0	7,3	7,4	7,5	7,6	7,7

P0	P1	P2	P3	P4	P5	P6	P7
1	0,1	0,2	0,3	0,4	0,5	0,6	0,7
0	1	1,2	1,3	1,4	1,5	1,6	1,7
0	0	1	2,3	2,4	2,5	2,6	2,7
0	0	0	1	3,4	3,5	3,6	3,7
0	0	0	4,3	4,4	4,5	4,6	4,7
0	0	0	5,3	5,4	5,5	5,6	5,7
0	0	0	6,3	6,4	6,5	6,6	6,7
0	0	0	7,3	7,4	7,5	7,6	7,7

Column Oriented Distribution(cont)

Gaussian Elimination Phase: Iteration No. 4 Snapshot

P0	P1	P2	P3	P4	P5	P6	P7
1	0,1	0,2	0,3	0,4	0,5	0,6	0,7
0	1	1,2	1,3	1,4	1,5	1,6	1,7
0	0	1	2,3	2,4	2,5	2,6	2,7
0	0	0	1	3,4	3,5	3,6	3,7
0	0	0	4,3	4,4	4,5	4,6	4,7
0	0	0	5,3	5,4	5,5	5,6	5,7
0	0	0	6,3	6,4	6,5	6,6	6,7
0	0	0	7,3	7,4	7,5	7,6	7,7

P0	P1	P2	P3	P4	P5	P6	P7
1	0,1	0,2	0,3	0,4	0,5	0,6	0,7
0	1	1,2	1,3	1,4	1,5	1,6	1,7
0	0	1	2,3	2,4	2,5	2,6	2,7
0	0	0	1	3,4	3,5	3,6	3,7
0	0	0	0	4,4	4,5	4,6	4,7
0	0	0	0	5,4	5,5	5,6	5,7
0	0	0	0	6,4	6,5	6,6	6,7
0	0	0	0	7,4	7,5	7,6	7,7

Column Oriented Distribution(cont)

Back Substitution Phase: Iteration No. 5 Snapshot

P0	P1	P2	P3	P4	P5	P6	P7
1	0,1	0,2	0,3	0	0	0	0
0	1	1,2	1,3	0	0	0	0
0	0	1	2,3	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1

P0	P1	P2	P3	P4	P5	P6	P7
1	0,1	0,2	0	0	0	0	0
0	1	1,2	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1

Comparison

Row wise distribution	Column wise distribution
Improper Load Balancing (Some processes become idle at some point) **More computation time**	Perfect load balancing (All processes participate till end) **Less Computation time**
Communicate only with a subset of processes at a time **Less Communication**	Communicate with all processes at a time **More communication**

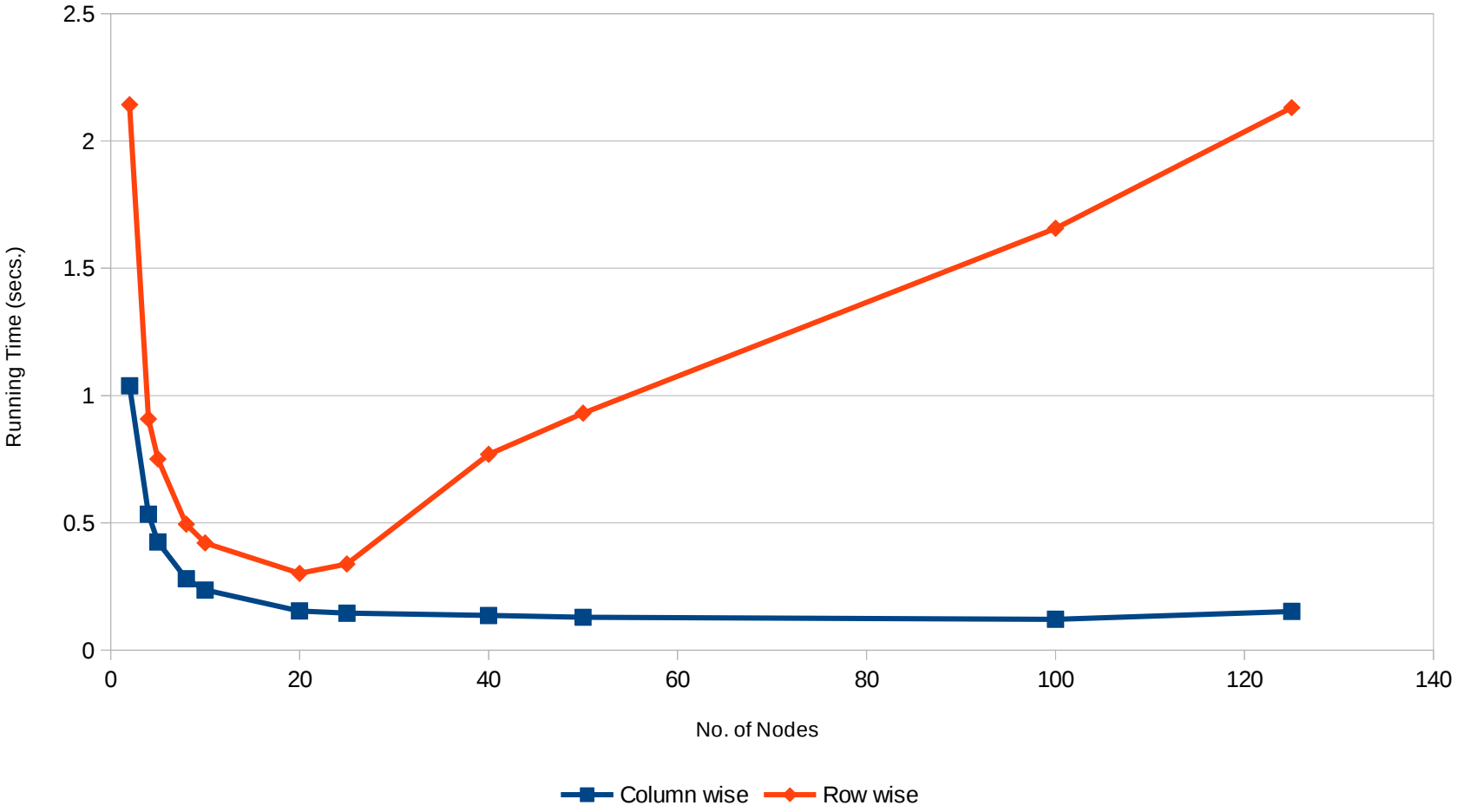
Computation_Time ∝ Order_of_matrix

Communication_Time ∝ No_of_processes

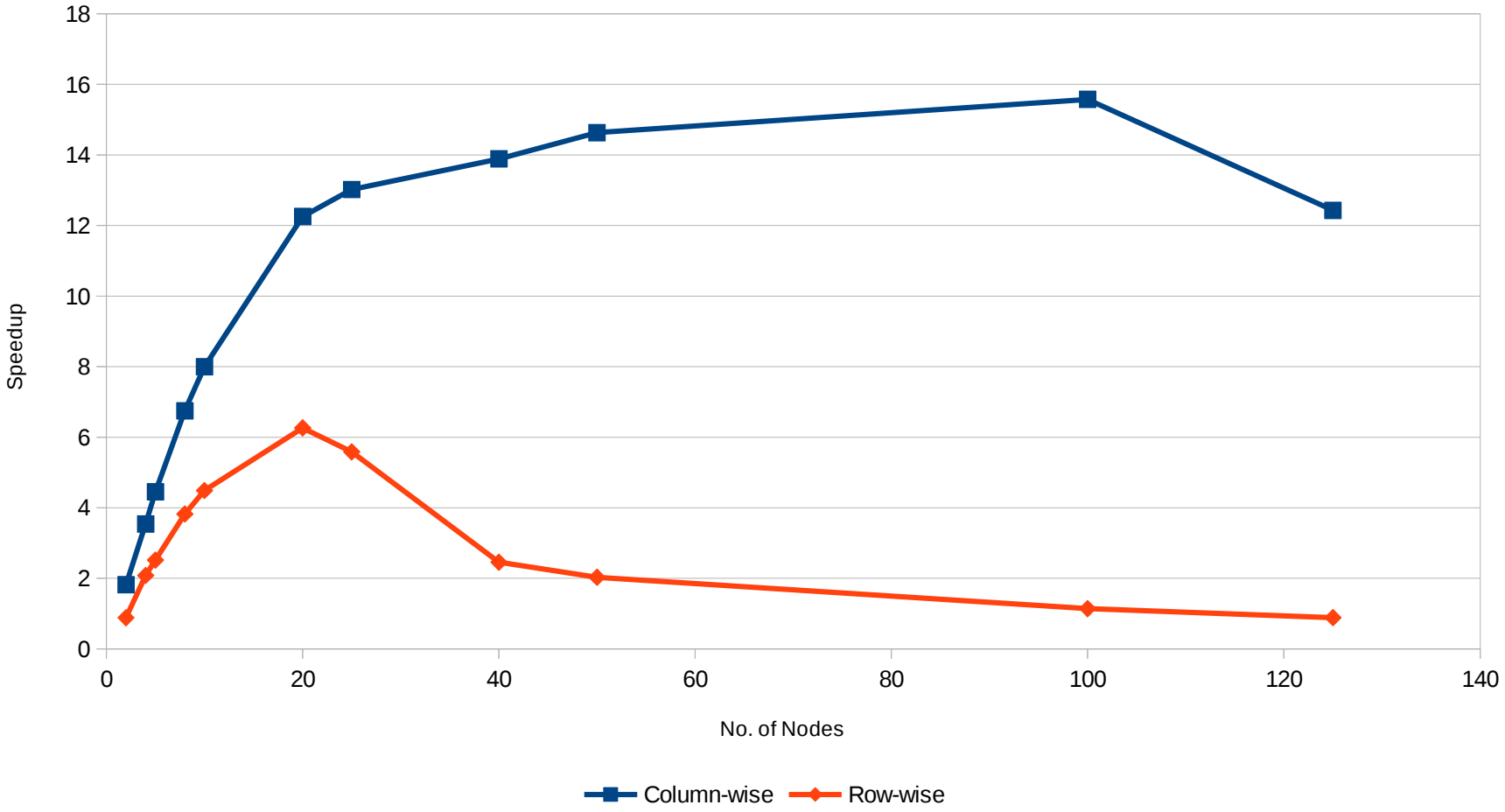
Running Time Results – 1000x1000

No. of Nodes	Row-wise Distribution (seconds)	Column-wise Distribution (seconds)
2	2.142413	1.038376
4	0.908051	0.533819
5	0.751067	0.424619
8	0.494652	0.280029
10	0.421074	0.236357
20	0.301833	0.154217
25	0.338351	0.145151
40	0.769326	0.136094
50	0.930928	0.129171
100	1.656955	0.121363
125	2.130919	0.152046

Running Time Results – 1000x1000



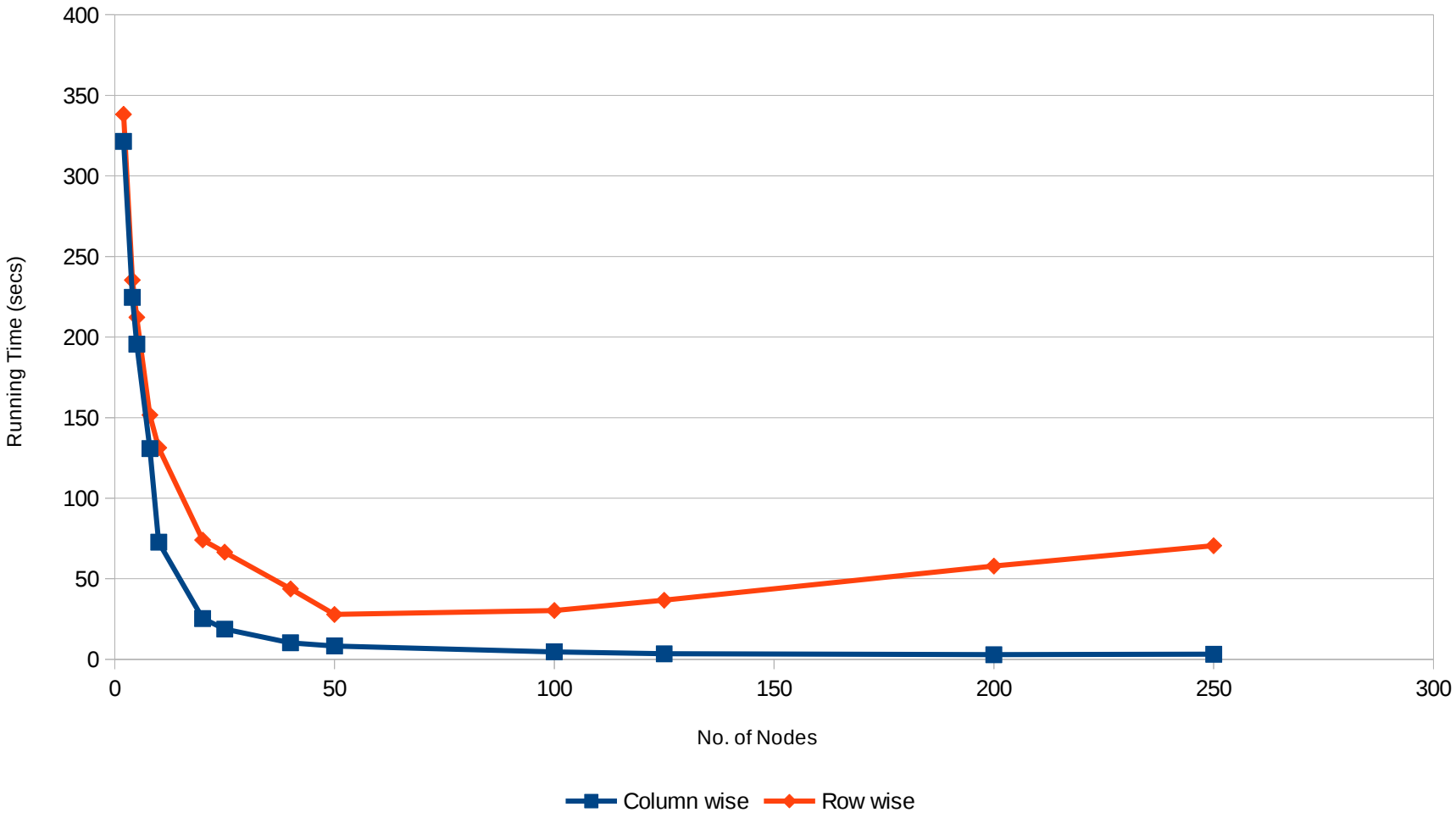
Speedup Results – 1000x1000



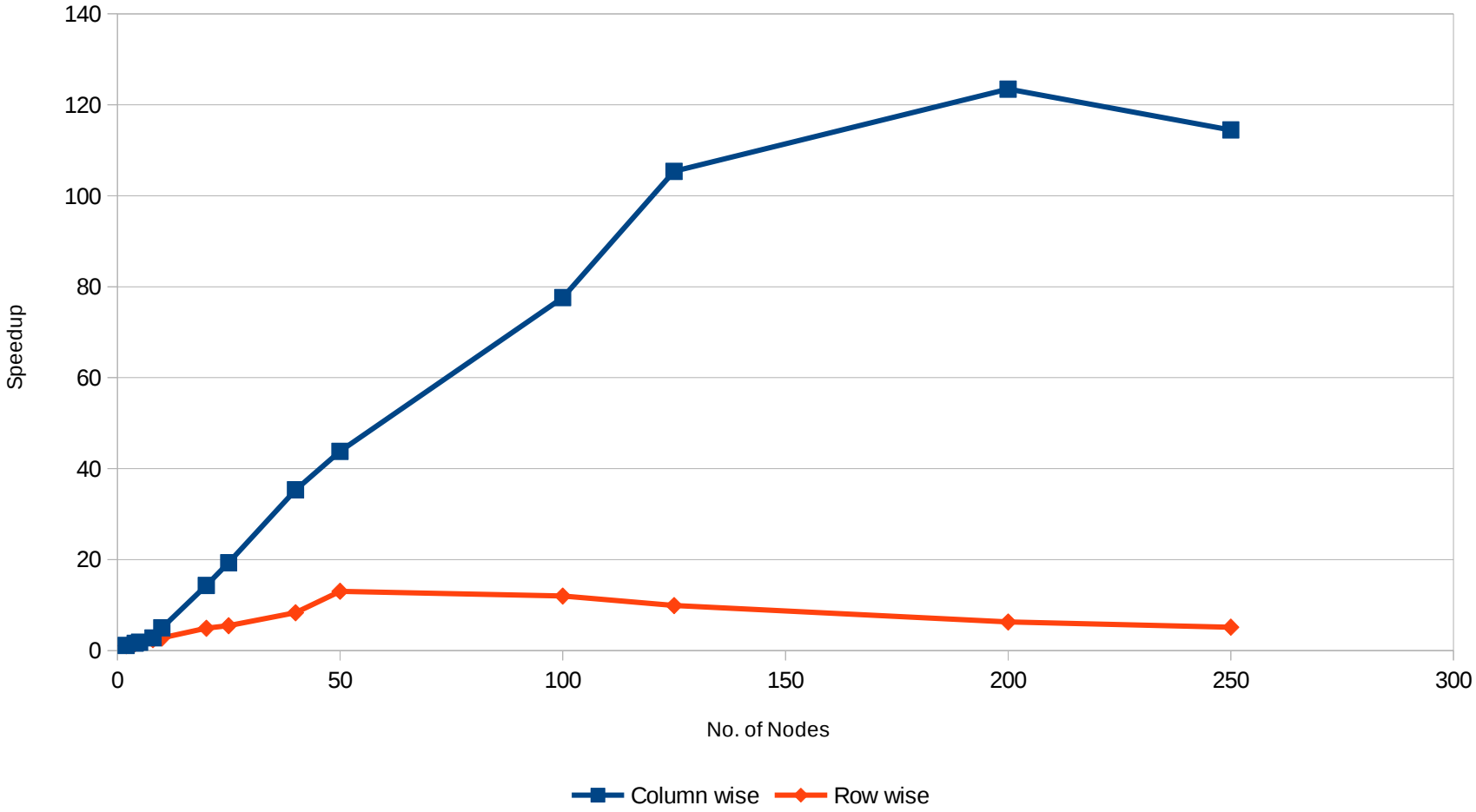
Running Time Results – 5000x5000

No. of Nodes	Row-wise Distribution (seconds)	Column-wise Distribution (seconds)
2	338.223091	321.461419
4	235.329189	224.581961
5	212.275457	195.597760
8	151.554124	130.827689
10	131.318633	72.705831
20	74.043321	25.330109
25	66.527007	18.803906
40	43.725915	10.276670
50	27.877240	8.289006
100	30.310265	4.677397
125	36.669621	3.445438
200	57.874497	2.940599
250	70.517770	3.171724

Running Time Results – 5000x5000



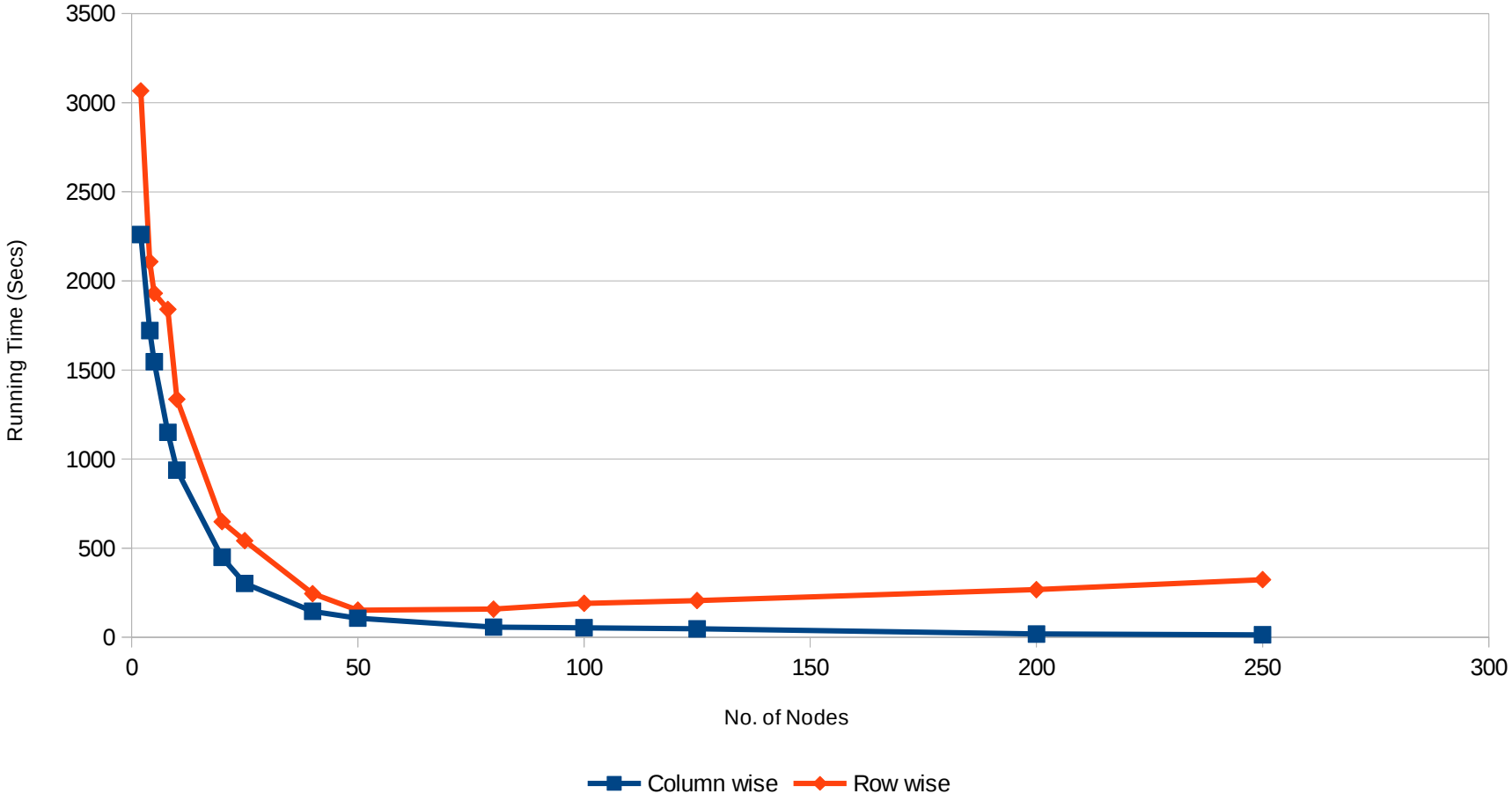
Speedup Results – 5000x5000



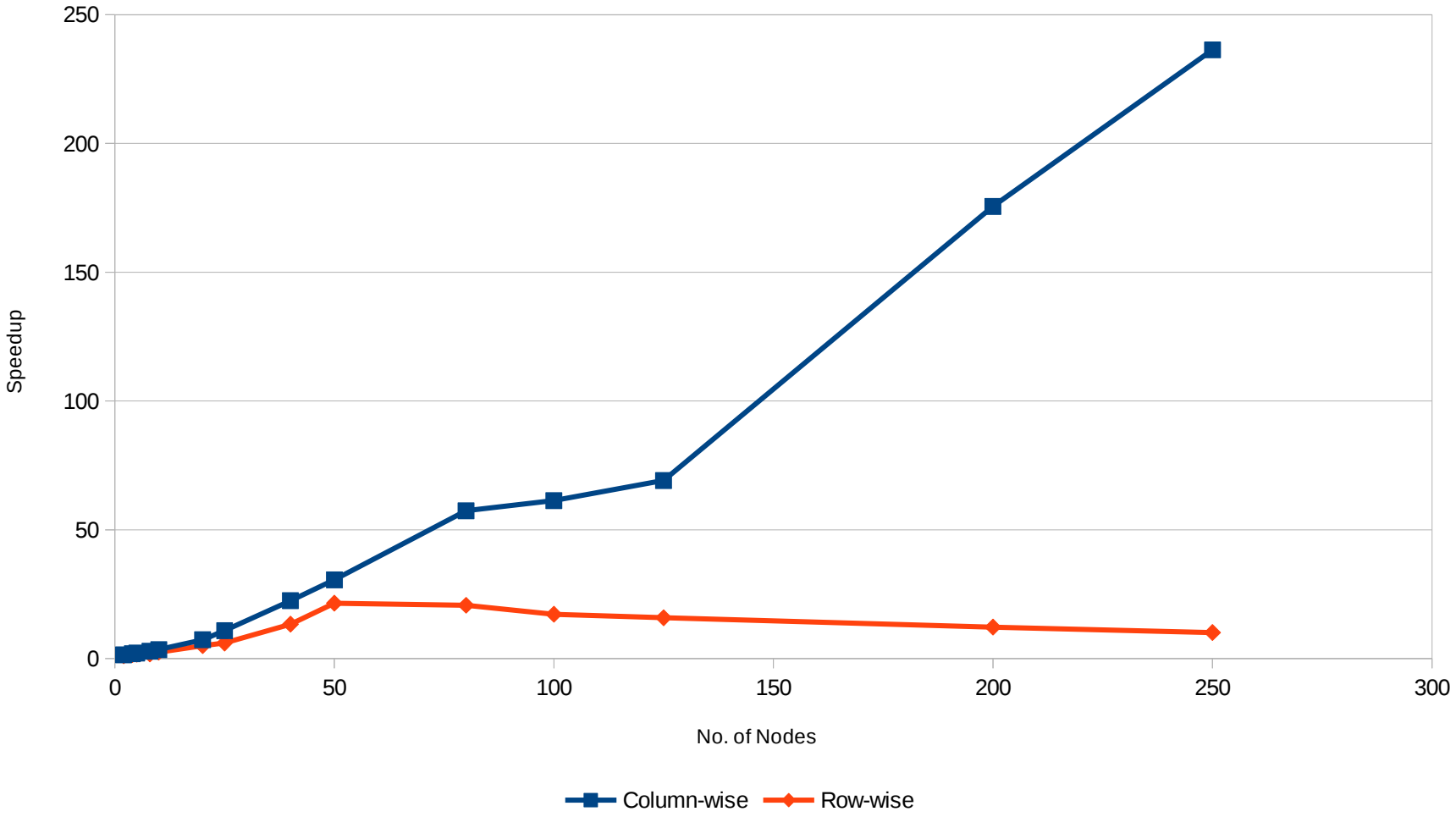
Running Time – 10000x10000

No. of Nodes	Row-wise Distribution	Column-wise Distribution
2	3066.518215	2260.243074
4	2107.468064	1721.380724
5	1930.502604	1546.118532
8	1840.306851	1150.174054
10	1335.704308	938.137521
20	648.383973	448.349273
25	541.870794	301.868054
40	245.219146	145.424132
50	152.166856	106.815007
80	157.845604	56.925365
100	190.207293	53.274259
125	206.301370	47.253685
200	267.325630	18.609259
250	323.239500	13.825426

Running Time – 10000 x 10000



Speedup Results – 10000 x 10000



Blunders

- Used MPI_Bcast (has an implicit barrier) within a fully parallel nested loop. Solution – collected all necessary data and one-time broadcast.
- Poor Cache performance - Arrays in C are stored in row-major order. Solution - Modified implementation for efficient memory access.
- Used icc for row wise distribution and gcc for column wise distribution. Solution – Used icc for both.

Future Scope

- MPI – OpenMP Hybrid Implementation
- Relax the constraint 'Matrix Dimension is divisible by the number of processes'
- Cyclic Mapping of rows – Load balancing effects in Row wise distribution
- Block_Cyclic Mapping – Study of efficiency improvement & Implementation.

References

1. Miller,Russ and Boxer,Laurence 2005. Algorithms Sequential & Parallel: A Unified Approach. 3rd edition. Cengage Learning. Page 161-168.
2. Richard P. Brent. 1991. Parallel Algorithms in Linear Algebra. Report TR-CS-91-06.
3. Chu,Eleanor and George, Alan. 1985. Gaussian Elimination using Partial Pivoting and Load balancing on a Multiprocessor.
4. Ben Lee. 1994. An Analysis of Data Distribution Methods for Gaussian Elimination in Distributed-Memory Multicomputers.
5. Roundoff Errors: http://www2.lawrence.edu/fast/GREGGJ/Math420/Section_6_2.pdf
6. http://cdac.in/index.aspx?id=ev_hpc_matrix_comp_solver_mpi

Thank You

Questions?