Parallel Algorithm for Dense Matrix Multiplication

CSE633 Parallel Algorithms

Fall 2012

Ortega, Patricia

Outline

- Problem definition
- Assumptions
- Implementation
- Test Results
- Future work
- Conclusions

Problem definition

- Given a matrix A(m × r) m rows and r columns, where each of its elements is denoted a_{ij} with $1 \leq i \leq m$ and $1 \leq j \leq r$, and a matrix B(r × n) of r rows and n columns, where each of its elements is denoted b_{ij} with $1 \leq i \leq r$, and $1 \leq j \leq n$, the matrix C resulting from the operation of multiplication of matrices A and B, C = A × B, is such that each of its elements is denoted ij with $1 \leq i \leq m$ and $1 \leq j \leq n$, and is calculated follows

$$c_{ij} = \sum_{k=1}^{r} a_{ik} \times b_{kj}$$

Problem definition (Cont.)

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \qquad \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix}$$

m x r   r x n

- The simplest way of multiplying two matrices takes about n^3 steps.

Problem definition (Cont.)

- The number of operation required to multiply A x B is:

$$m \times n \times (2r-1)$$

- For simplicity, usually it is analyzed in terms of square matrices of order n. So that the quantity of basic operations between scalars is :

$$2n^3 - n^2 = O(n^3)$$

Sequential algorithm

```
for (i = 0; i < n; i++)
    for (j = 0; i < n; j++)
        c[i][j] = 0;
        for (k = 0; k < n; k++)
            c[i][j]  += a[i][k] * b[k][j]
        end for
    end for
end for
```

Assumptions

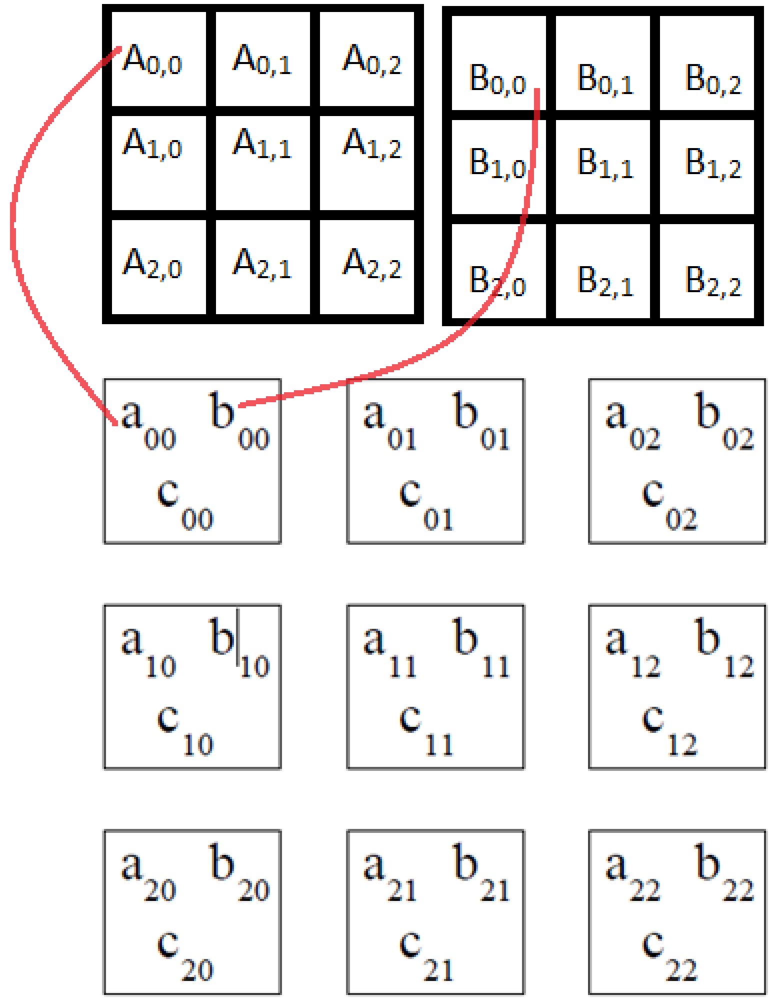
- For simplicity, we will work with square matrices of size n x n.
- Considered the number of processors available in parallel machines as p.
- The matrixes to multiply will be A and B. Both will be treated as dense matrices (with few 0's), the result will be stored it in the matrix C.
- It is assumed that the processing nodes are homogeneous, due this homogeneity it is possible achieve load balancing.

Implementation

- Consider two square matrices A and B of size n that have to be multiplied:

1. Partition these matrices in square blocks p, where p is the number of processes available.
2. Create a matrix of processes of size $p^{1/2}$ x $p^{1/2}$ so that each process can maintain a block of A matrix and a block of B matrix.
3. Each block is sent to each process, and the copied sub blocks are multiplied together and the results added to the partial results in the C sub-blocks.
4. The A sub-blocks are rolled one step to the left and the B sub-blocks are rolled one step upward.
5. Repeat steps 3 y 4 sqrt(p) times.

Implementation (Cont.)



Example

Matrices to be multiplied

$$A = \begin{bmatrix} 2 & 1 & 5 & 3 \\ 0 & 7 & 1 & 6 \\ 9 & 2 & 4 & 4 \\ 3 & 6 & 7 & 2 \end{bmatrix}$$

$$B = \begin{bmatrix} 6 & 1 & 2 & 3 \\ 4 & 5 & 6 & 5 \\ 1 & 9 & 8 & -8 \\ 4 & 0 & -8 & 5 \end{bmatrix}$$

Example:

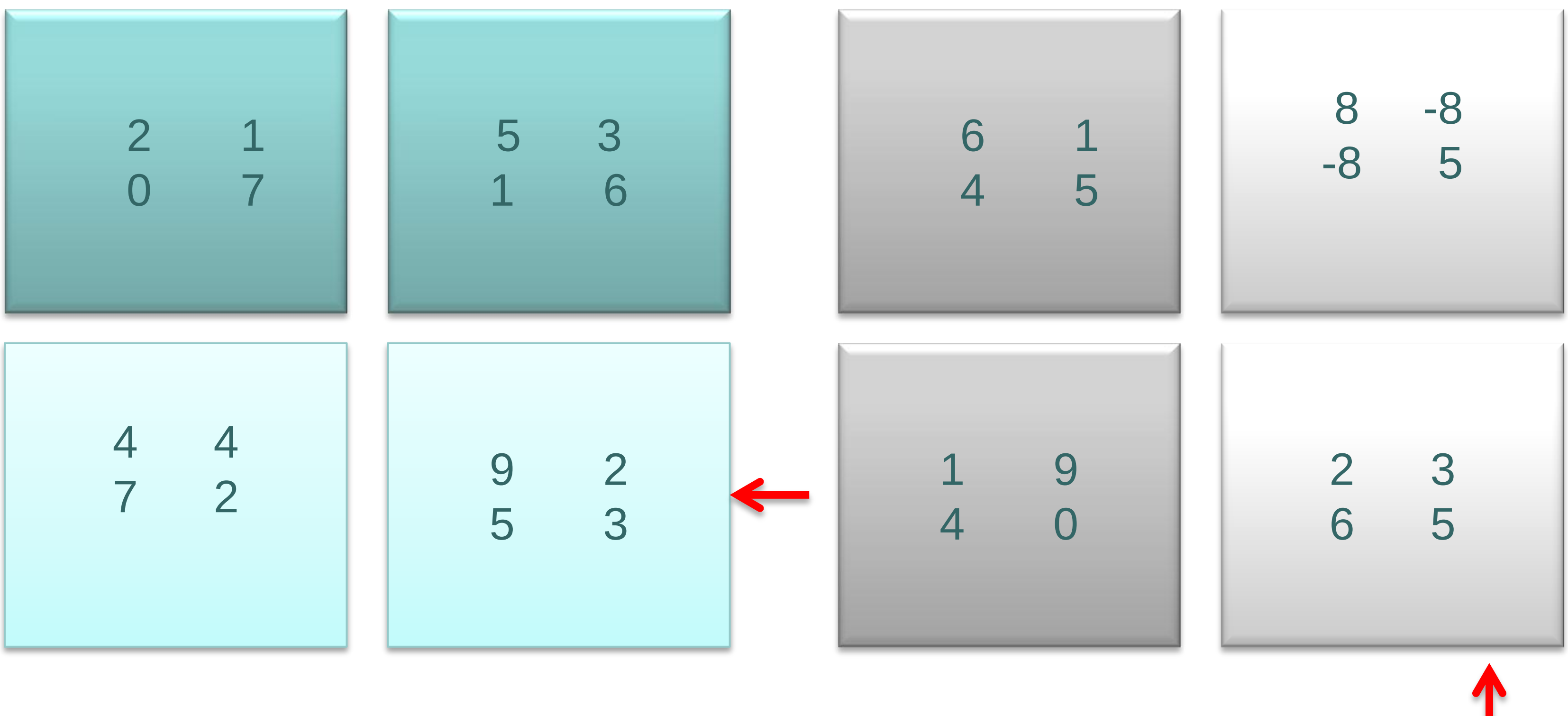
- These matrices are divided into 4 square blocks as follows:

$P_{0,0}$ 2 1 0 7	$P_{0,1}$ 5 3 1 6
$P_{1,0}$ 9 2 5 3	$P_{1,1}$ 4 4 7 2

$P_{0,0}$ 6 1 4 5	$P_{0,1}$ 2 3 6 5
$P_{1,0}$ 1 9 4 0	$P_{1,1}$ 8 -8 -8 5

Example:

- Matrices A and B after the initial alignment.



Example:

- Local matrix multiplication.

$$C_{0,0} = \begin{bmatrix} 2 & 1 \\ 0 & 7 \end{bmatrix} \times \begin{bmatrix} 6 & 1 \\ 4 & 5 \end{bmatrix} = \begin{bmatrix} 16 & 7 \\ 28 & 35 \end{bmatrix}$$

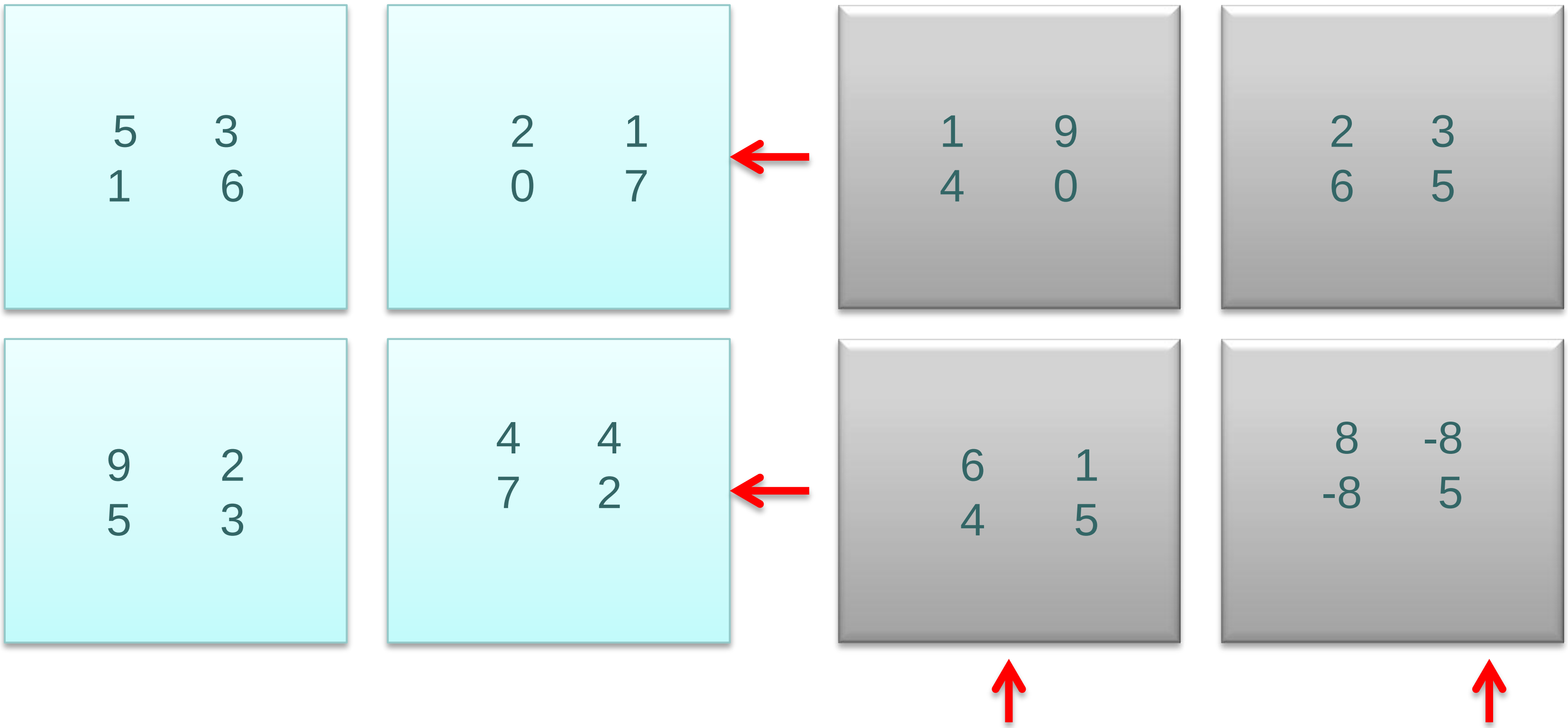
$$C_{0,1} = \begin{bmatrix} 5 & 3 \\ 1 & 6 \end{bmatrix} \times \begin{bmatrix} 8 & -8 \\ -8 & 5 \end{bmatrix} = \begin{bmatrix} 16 & -25 \\ -40 & 22 \end{bmatrix}$$

$$C_{1,0} = \begin{bmatrix} 4 & 4 \\ 7 & 2 \end{bmatrix} \times \begin{bmatrix} 1 & 9 \\ 4 & 0 \end{bmatrix} = \begin{bmatrix} 20 & 36 \\ 15 & 63 \end{bmatrix}$$

$$C_{1,1} = \begin{bmatrix} 9 & 2 \\ 5 & 3 \end{bmatrix} \times \begin{bmatrix} 2 & 3 \\ 6 & 5 \end{bmatrix} = \begin{bmatrix} 30 & 37 \\ 42 & 39 \end{bmatrix}$$

Example:

- Shift A one step to left, shift B one step up



Example:

- Local matrix multiplication.

$$C_{0,0} = C_{0,0} + \begin{bmatrix} 2 & 1 \\ 0 & 7 \end{bmatrix} \times \begin{bmatrix} 6 & 1 \\ 4 & 5 \end{bmatrix} = \begin{bmatrix} 16 & 7 \\ 28 & 35 \end{bmatrix} + \begin{bmatrix} 17 & 45 \\ 25 & 9 \end{bmatrix} = \begin{bmatrix} 33 & 52 \\ 53 & 44 \end{bmatrix}$$

$$C_{0,1} = C_{0,1} + \begin{bmatrix} 5 & 3 \\ 1 & 6 \end{bmatrix} \times \begin{bmatrix} 8 & -8 \\ -8 & 5 \end{bmatrix} = \begin{bmatrix} 16 & -25 \\ -40 & 22 \end{bmatrix} + \begin{bmatrix} 10 & 11 \\ 42 & 35 \end{bmatrix} = \begin{bmatrix} 26 & -14 \\ 2 & 57 \end{bmatrix}$$

$$C_{1,0} = C_{1,0} + \begin{bmatrix} 4 & 4 \\ 7 & 2 \end{bmatrix} \times \begin{bmatrix} 1 & 9 \\ 4 & 0 \end{bmatrix} = \begin{bmatrix} 20 & 36 \\ 15 & 63 \end{bmatrix} + \begin{bmatrix} 62 & 19 \\ 42 & 33 \end{bmatrix} = \begin{bmatrix} 82 & 55 \\ 57 & 96 \end{bmatrix}$$

$$C_{1,1} = C_{1,1} + \begin{bmatrix} 9 & 2 \\ 5 & 3 \end{bmatrix} \times \begin{bmatrix} 2 & 3 \\ 6 & 5 \end{bmatrix} = \begin{bmatrix} 30 & 37 \\ 42 & 39 \end{bmatrix} + \begin{bmatrix} 0 & -12 \\ 40 & -46 \end{bmatrix} = \begin{bmatrix} 30 & 25 \\ 82 & -7 \end{bmatrix}$$

Test

Objective:

- Analyze speedup achieved by the parallel algorithm when increases the size of the input data and the number of cores of the architecture.

Constraints:

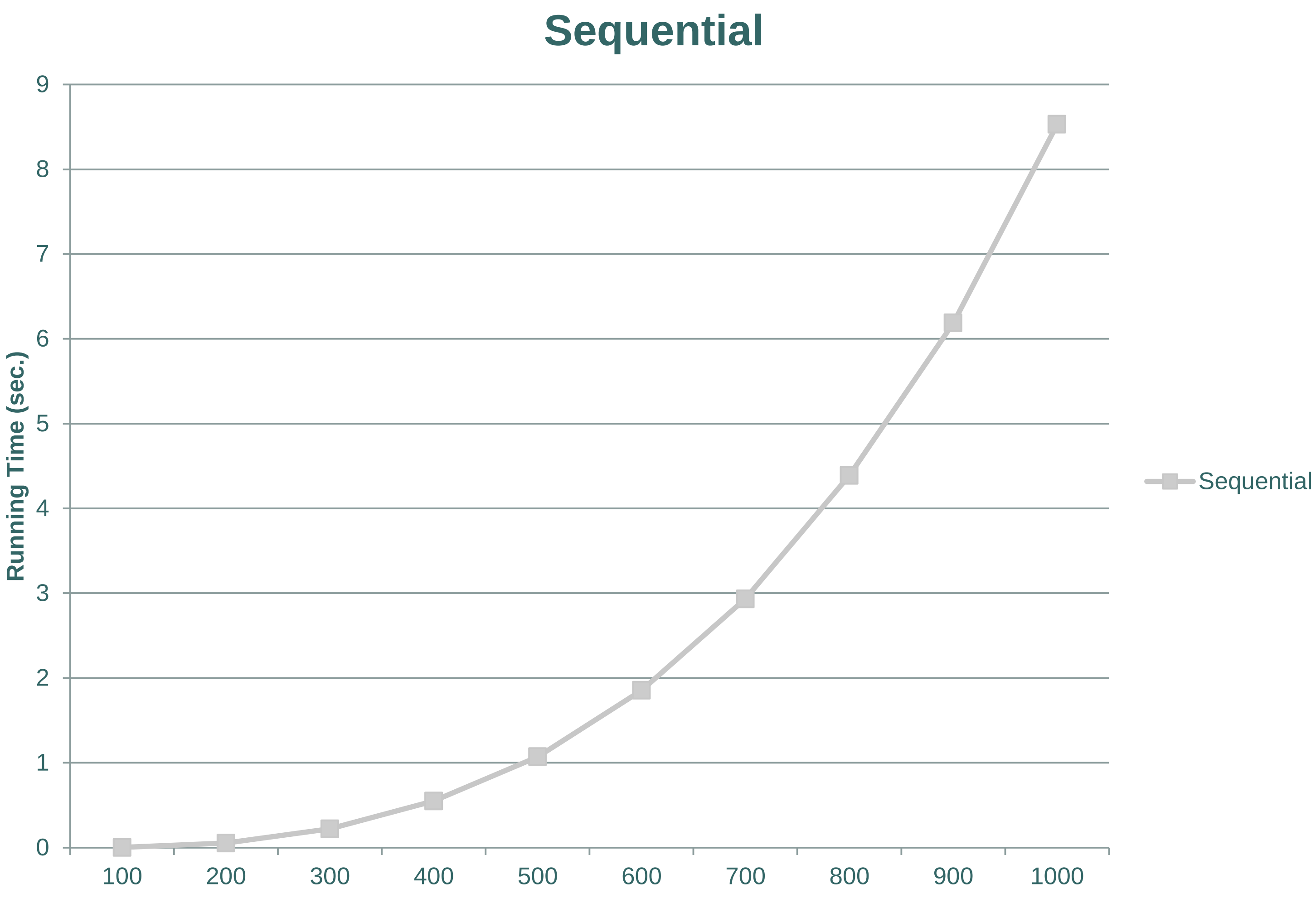
- The number of cores used must be a exact square root.
- Must be possible the exact distribution to the total amount of data into the available cores.

Test

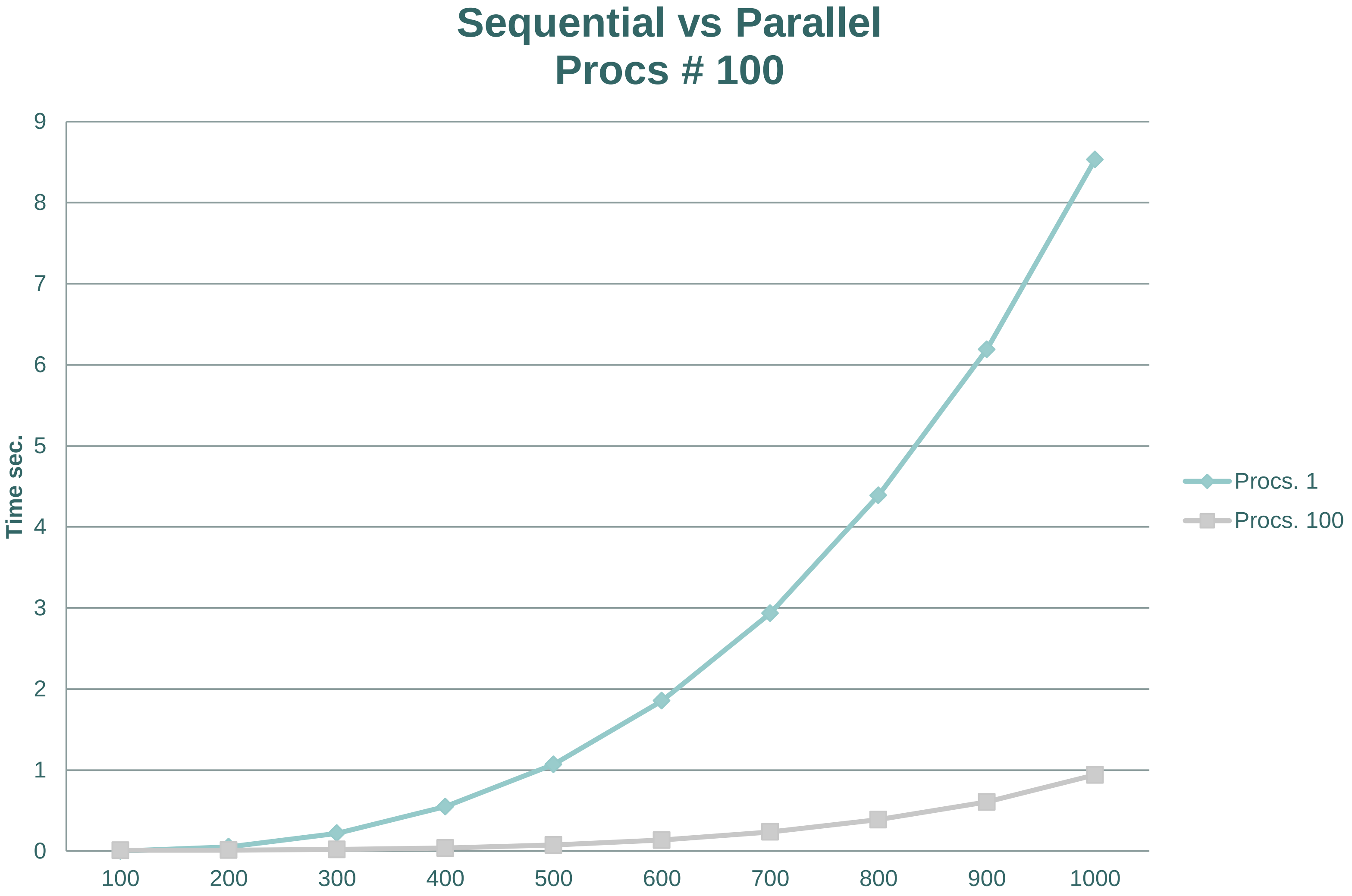
Execution:

- Run sequential algorithm on a single processor/ core.
- For test the parallel algorithm were used the following number of cores: 4,9,16,25,36,49,64,100
- The results were obtained from the average over three tests of the algorithms.
- Test performed in matrices with dimensions up 1000x1000, increasing with steps of 100.

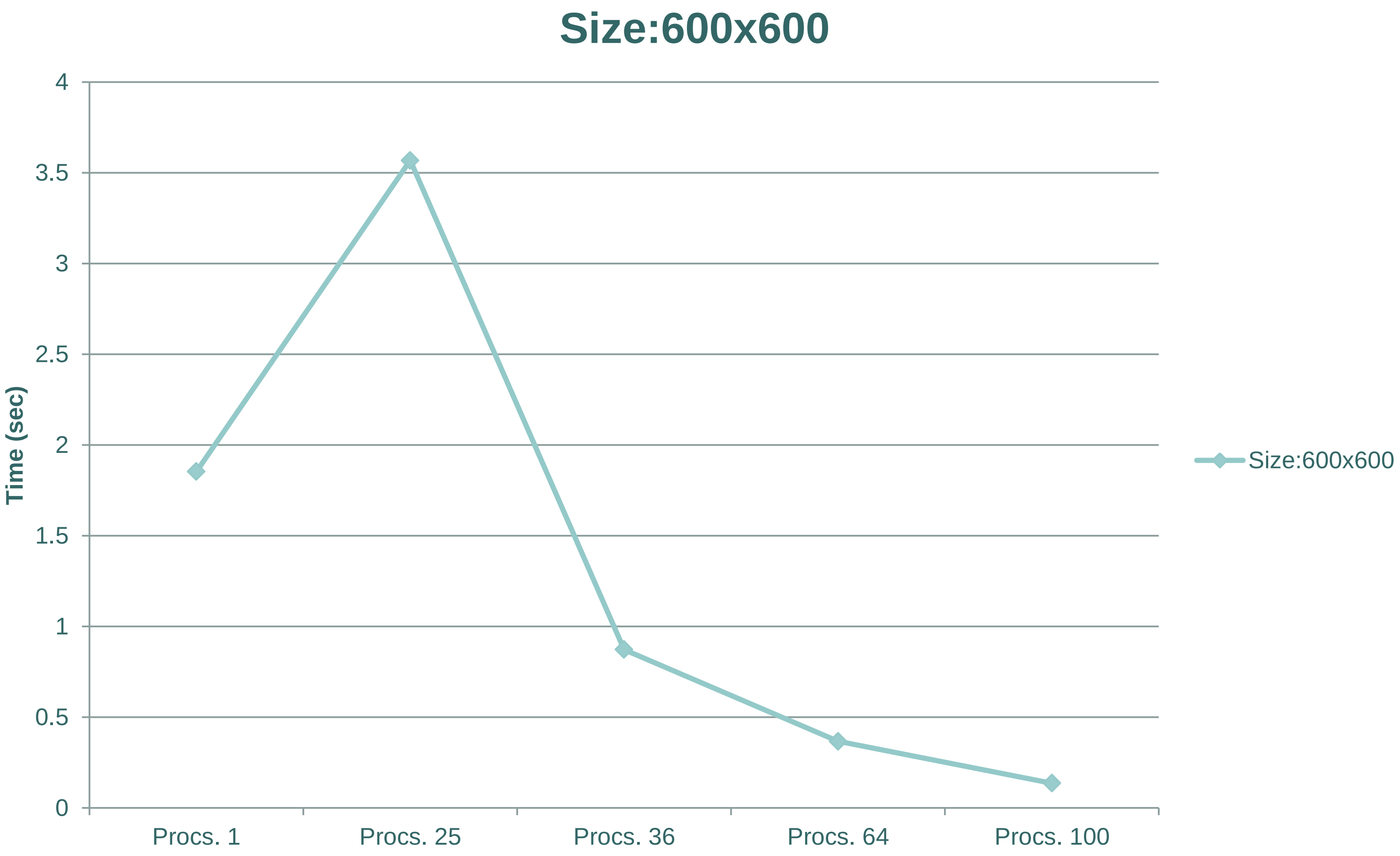
Test Results - Sequential



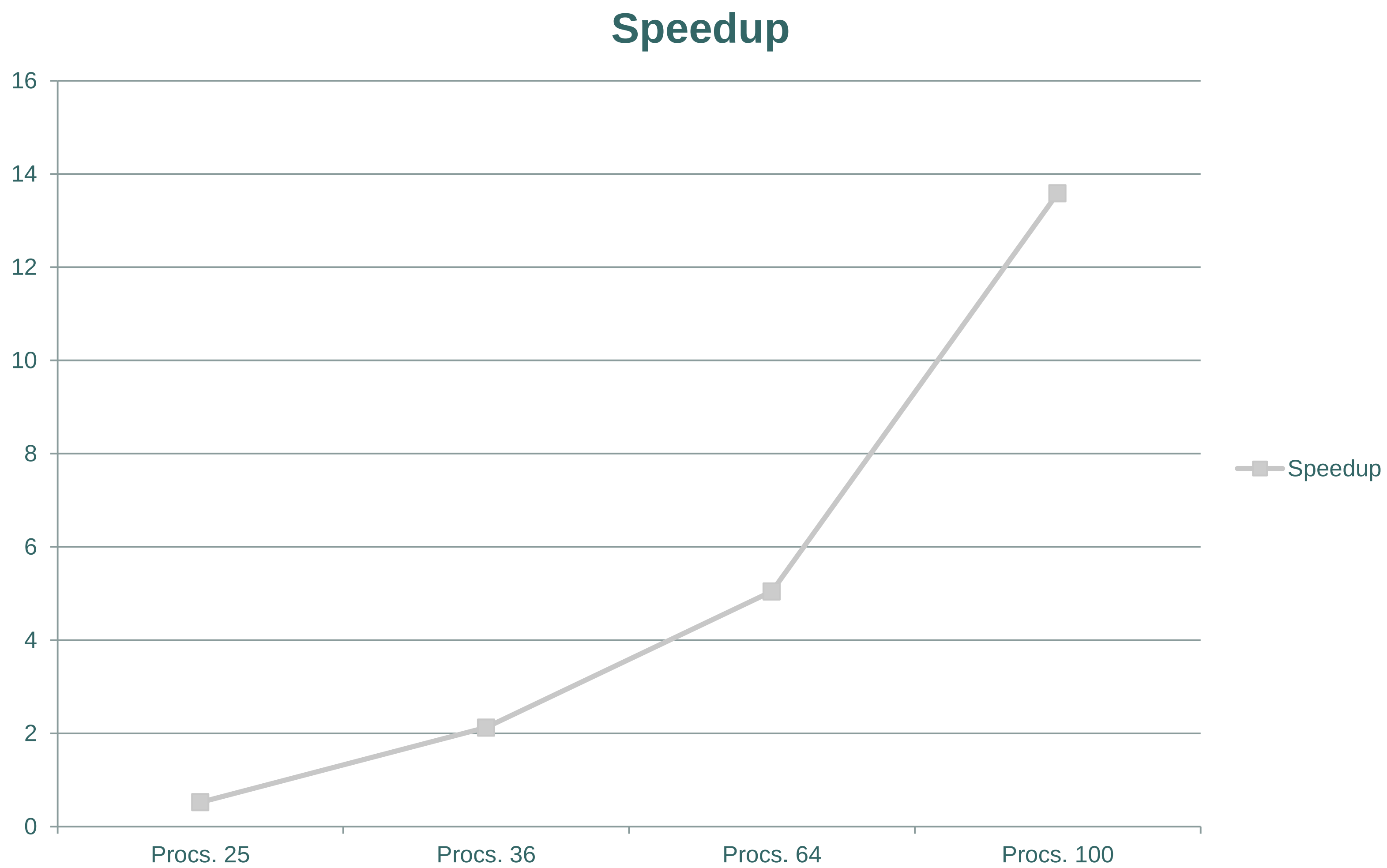
Test Results Sequential vs. Parallel



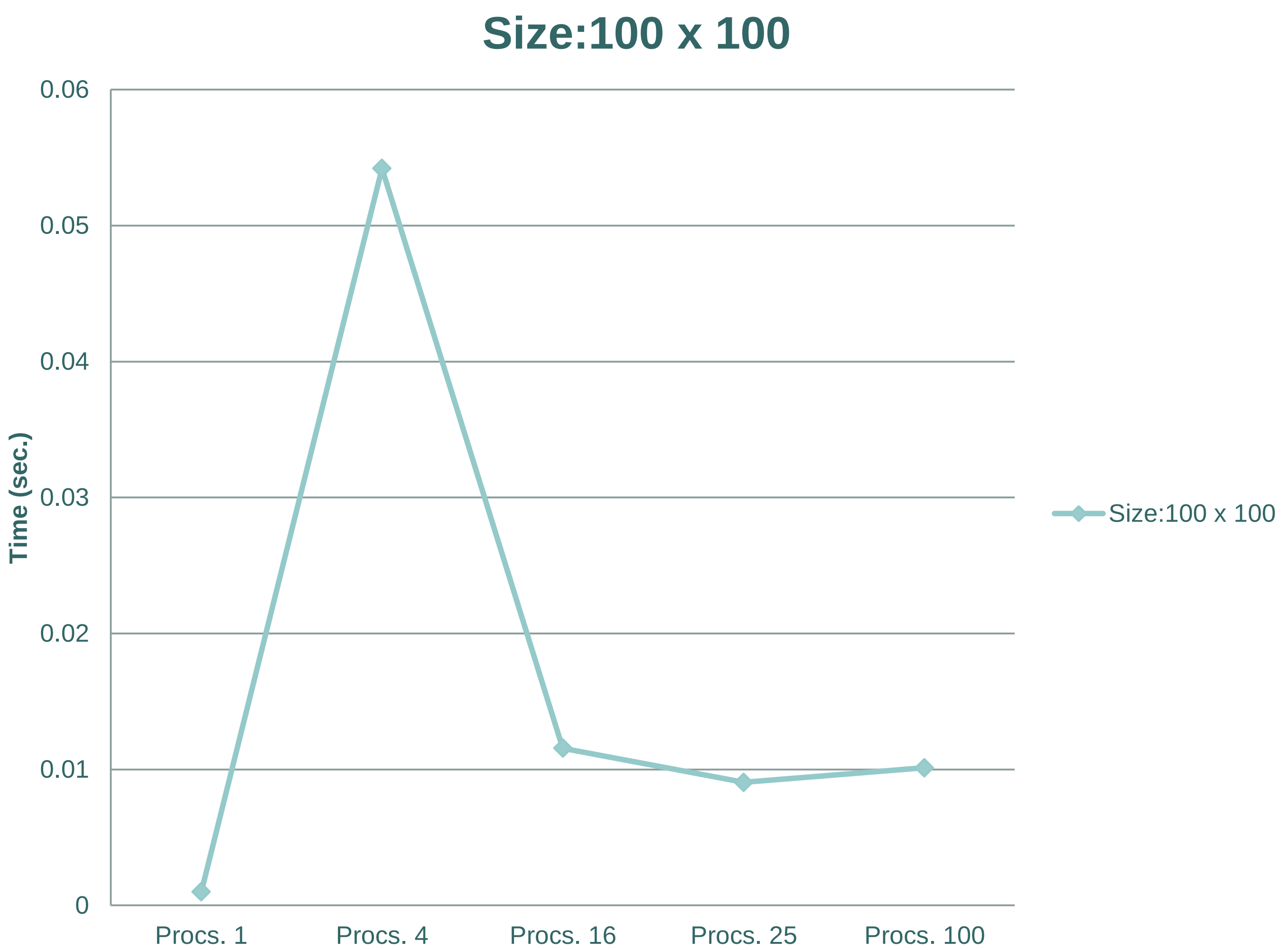
Test Results - Parallel



Test Results - Parallel



Test Results – Parallel (Counter Intuitive in small test data)



Further work

- Implement the algorithm in OpenMP to compare the performance of the two solutions.
- Implementation in MPI platform using threads when a processor receives more than one piece of data.

Conclusions

- The distribution of data and computing division across multiple processors offers many advantages:
  - With MPI it is required less effort in terms of the timing required for data handling, since each process has its own portion.
  - MPI offers flexibility for data exchange.

References

- [1] V. Vassilevska Williams, "Breaking the Coppersmith-Winograd barrier,“ [Online]. Available: http://www.cs.berkeley.edu/~virgi/matrixmult.pdf. [Accessed 18 09 2012].
- [2] Gupta, Anshul; Kumar, Vipin; , "Scalability of Parallel Algorithms for Matrix Multiplication," *Parallel Processing, 1993. ICPP 1993. International Conference on* , vol.3, no., pp.115-123, 16-20 Aug. 1993
doi: 10.1109/ICPP.1993.160
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4134256&isnumber=4134231

Questions…