Parallelizing matrix multiplication

by
Sandeep Raghuraman
CSE 633 – Fall 2011

Introduction

- Matrix multiplication of 2 matrices:

$$\mathbf{A} = \begin{vmatrix} a_{11} & a_{12} & . & . & a_{1n} \\ a_{21} & a_{22} & . & . & a_{2n} \\ . & . & . & . & . \\ . & . & . & . & . \\ a_{m1} & a_{m2} & . & . & a_{mn} \end{vmatrix} \quad \mathbf{B} = \begin{vmatrix} b_{11} & b_{12} & . & . & b_{1p} \\ b_{21} & b_{22} & . & . & b_{2p} \\ . & . & . & . & . \\ . & . & . & . & . \\ b_{n1} & b_{n2} & . & . & b_{np} \end{vmatrix}$$

m * n n * p

- C = A*B. C will have m rows and p columns.
- $(C)_{i,j} = \sum_{k=1}^{n} A_{ik} B_{kj},$ (Picture source: http://en.wikipedia.org/wiki/Matrix_multiplication)
- Sequential brute force algorithm takes O(m*n*p) time

Test parameters

- Brute force sequential matrix multiplication run on a single processor/core
- Number of rows and columns were equal. Both matrices had the same dimensions
- Matrix dimensions ranged from 1000 to 10000 increasing in steps of 1000
- I used the 32-core nodes with 256 GB of RAM. Only 1 MPI process was used.
- The running time is for the computation part only(including communication between nodes for computation). It does not include the time for distributing data to and gathering data from the nodes that do computation.

Results

Each test was run 3 times and the average of the running times is shown below:

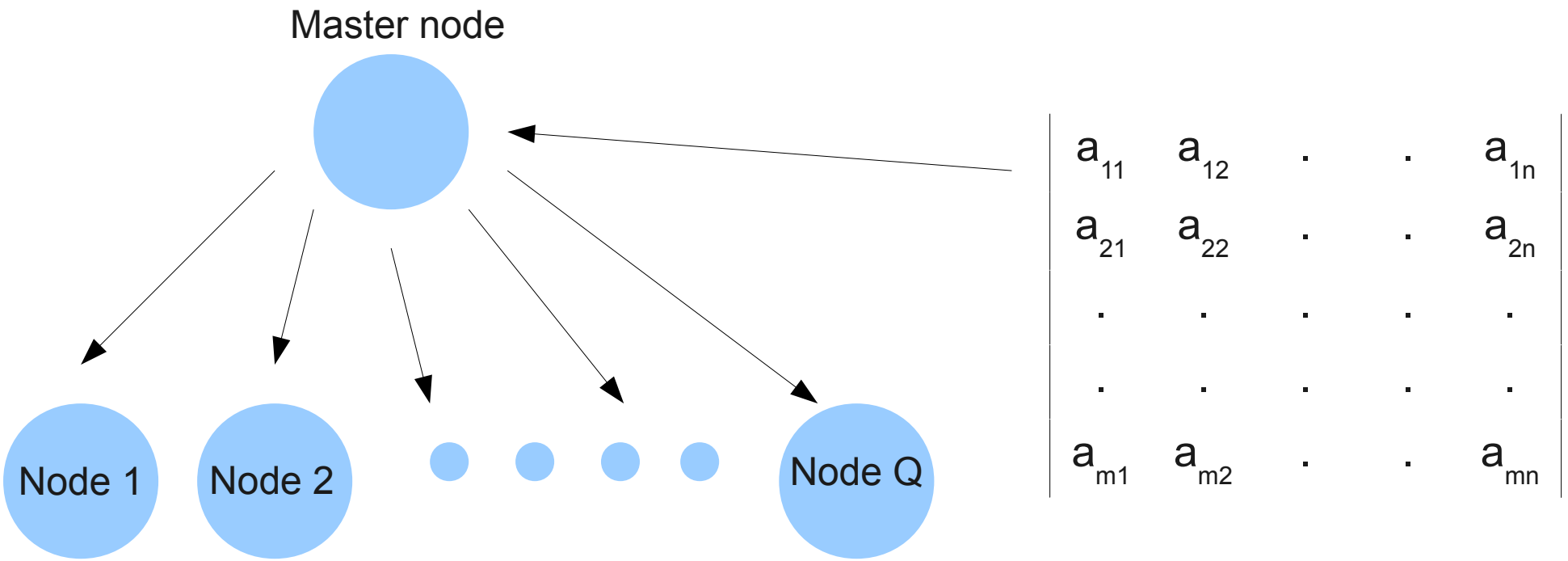
Matrix dimensions	Running time (seconds)
1000	11
2000	84
3000	373
4000	854
5000	2005
6000	2916
7000	5391
8000	6576
9000	11442
10000	12996

Parallel version

- Use multiple nodes to perform computations in parallel
- Each node works independently on different sections of the matrix

How the work is split up

- Assume Q nodes are used
- A master node(MPI process with rank 0) generates random matrix data allocates m/Q rows from matrix A to each node. Each node also gets the entire matrix B. The master node also participates in the computation.
- So, node i gets rows from (i-1)*(m/Q)+1 to i*(m/Q) of matrix A


Master node
Node 1
Node 2
Node Q
$\begin{vmatrix} a_{11} & a_{12} & . & . & a_{1n} \\ a_{21} & a_{22} & . & . & a_{2n} \\ . & . & . & . & . \\ . & . & . & . & . \\ a_{m1} & a_{m2} & . & . & a_{mn} \end{vmatrix}$

- Each node multiplies the portion of A that it has and the matrix B.

$$\begin{vmatrix} a_{11} & a_{12} & . & . & a_{1n} \\ a_{21} & a_{22} & . & . & a_{2n} \\ . & . & . & . & . \\ . & . & . & . & . \\ a_{m/Q,1} & a_{m/Q,2} & . & . & a_{m/Q,n} \end{vmatrix} \quad \begin{vmatrix} b_{11} & b_{12} & . & . & b_{1p} \\ b_{21} & b_{22} & . & . & b_{2p} \\ . & . & . & . & . \\ . & . & . & . & . \\ b_{n1} & b_{n2} & . & . & b_{np} \end{vmatrix}$$

- Each node i calculates the rows from

  (i-1)*(m/Q)+1 to i*(m/Q) of the matrix C

- Basically, different portions of the matrix are calculated in parallel

- The calculated results are sent back to the master node which can store the final result in a file

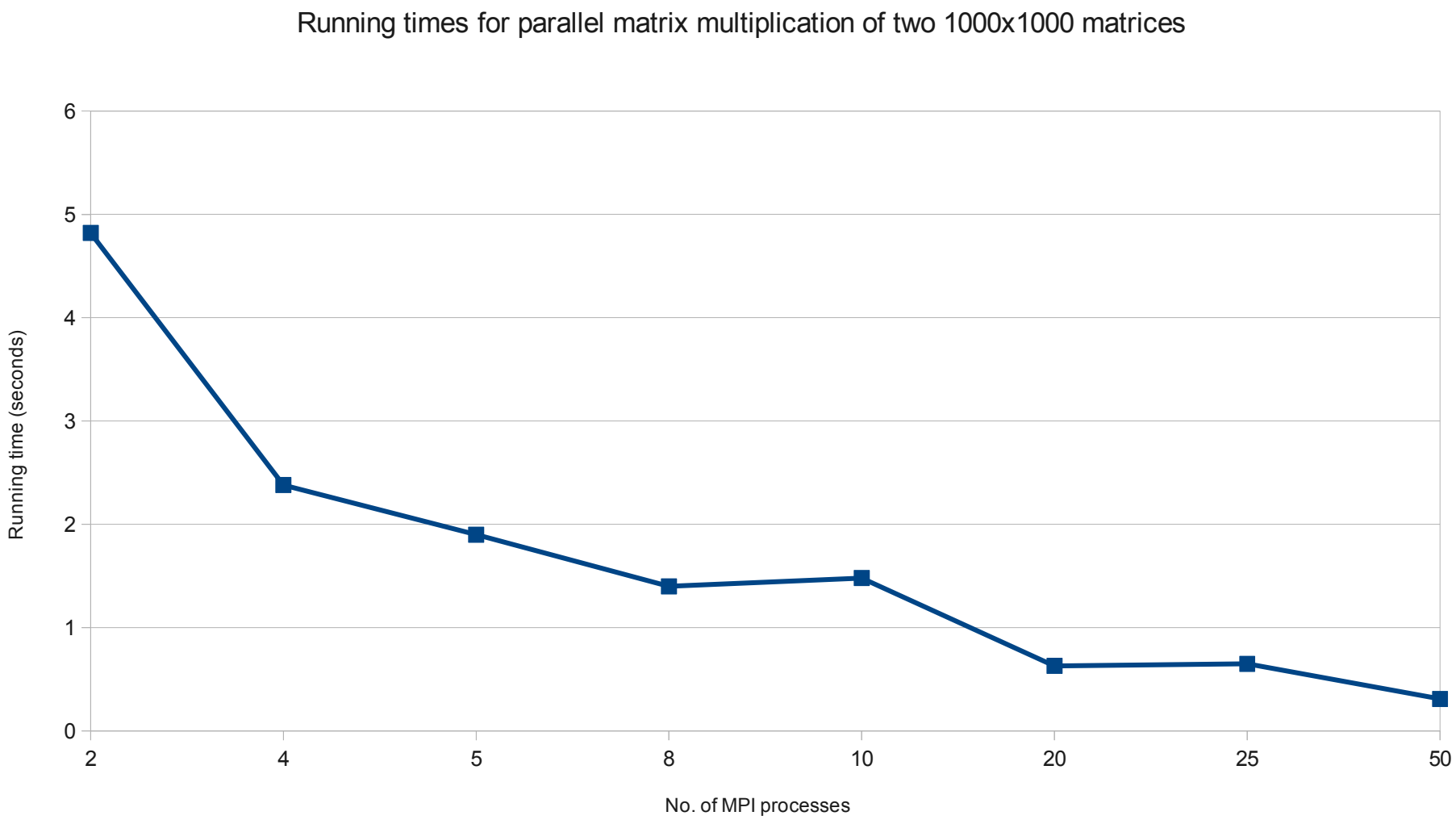
Assumptions

- Matrix dimensions are divisible by number of MPI processes

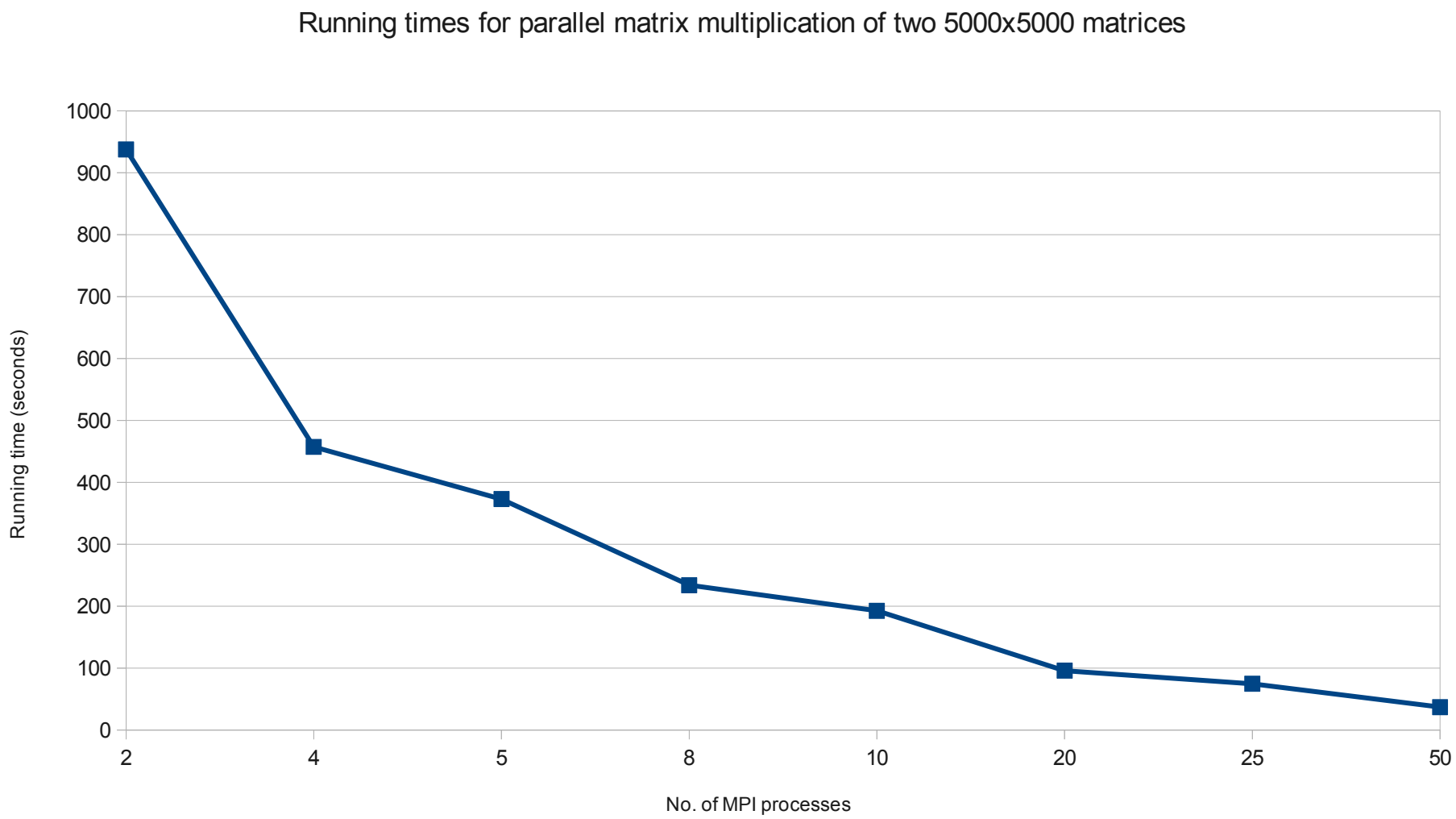
Test parameters

- Matrix dimensions ranged from 1000 to 10000 increasing in steps of 1000
- Number of rows and columns were equal. Both matrices had the same dimensions
- No. of processes used were 2,4,5,8,10,20,25,50
- I used the 32-core nodes with 256 GB of RAM
- For the 50 process test, used two 32-core nodes, for the others used only one 32-core node
- Each test was run 3 times
- The running time is for the computation part only(including communication between nodes for computation). It does not include the time for distributing data to and gathering data from the nodes that do computation.

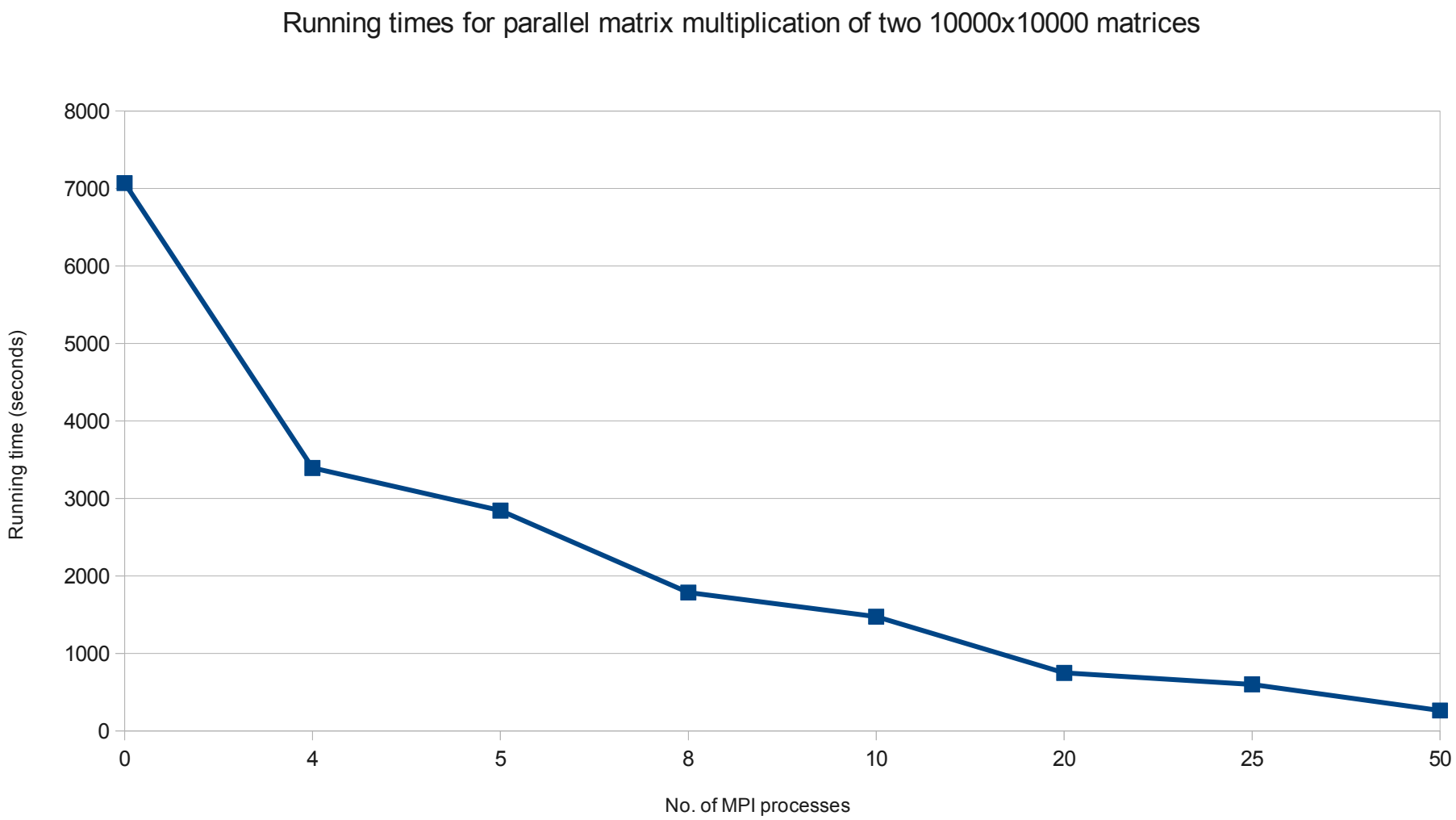
Test results



Test results (contd.)



Test results (contd.)



Second parallel version

- Instead of each node storing the entire matrix B, each node can store p/Q columns of matrix B. Thus, node i will have columns (i-1)*(p/Q)+1 to i*(p/Q)
- Each node then calculates a part of matrix C using the data it has.
- Then, each node i passes the columns of B that it has to node i+1. In the case of the Qth node, it passes the columns it has to node 1. The above step is repeated P times.

- Using this method, lesser memory per process is required

- The running time might increase since some communication is required between different processors.

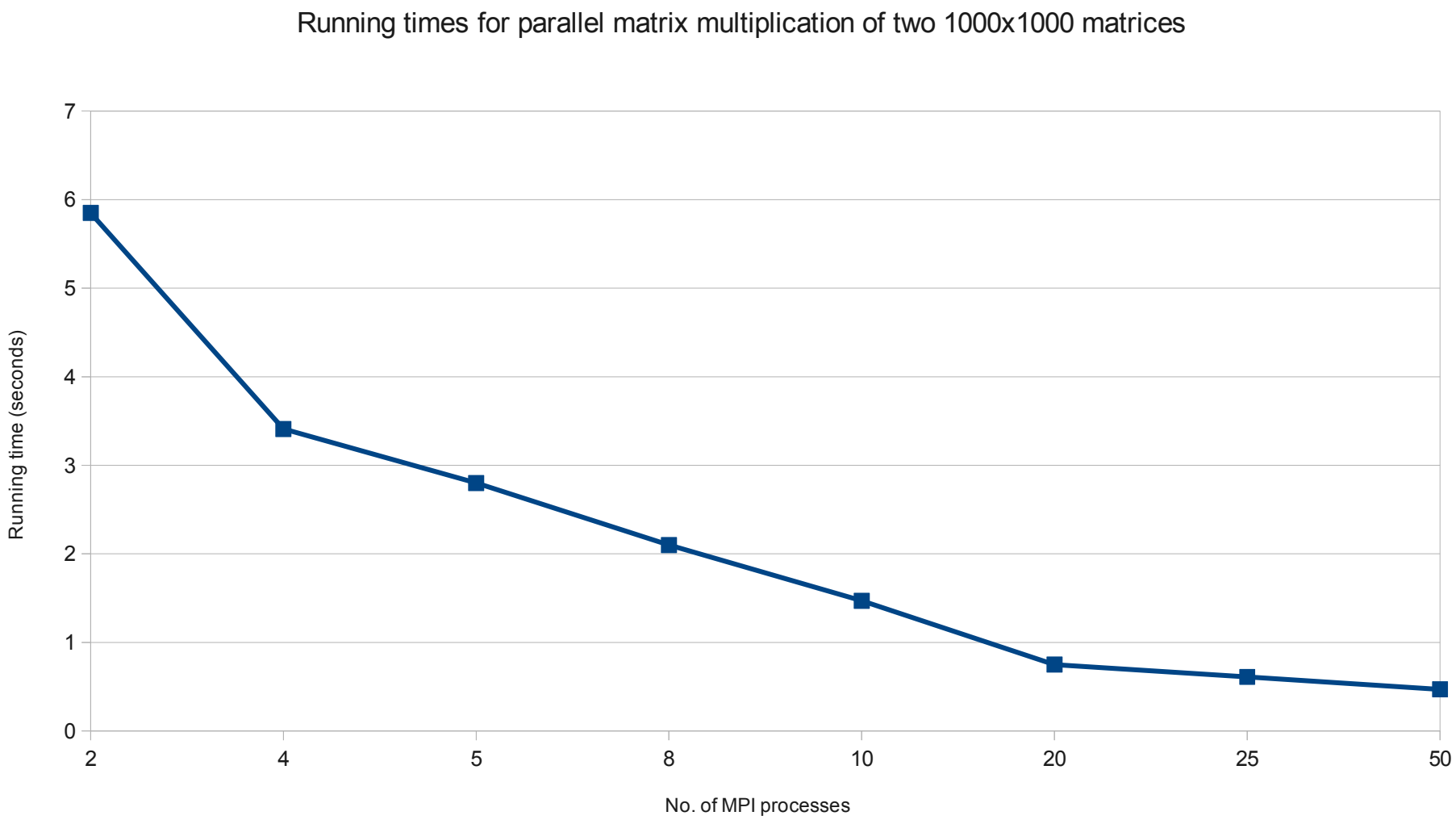
Test parameters

- Matrix dimensions ranged from 1000 to 10000 increasing in steps of 1000
- Number of rows and columns were equal. Both matrices had the same dimensions
- No. of processes used were 2,4,5,8,10,20,25,50
- I used the 32-core nodes with 256 GB of RAM
- For the 50 process test, used two 32-core nodes, for the others used only one 32-core node
- Each test was run 2 times
- The running time is for the computation part only(including communication between nodes for computation). It does not include the time for distributing data to and gathering data from the nodes that do computation.

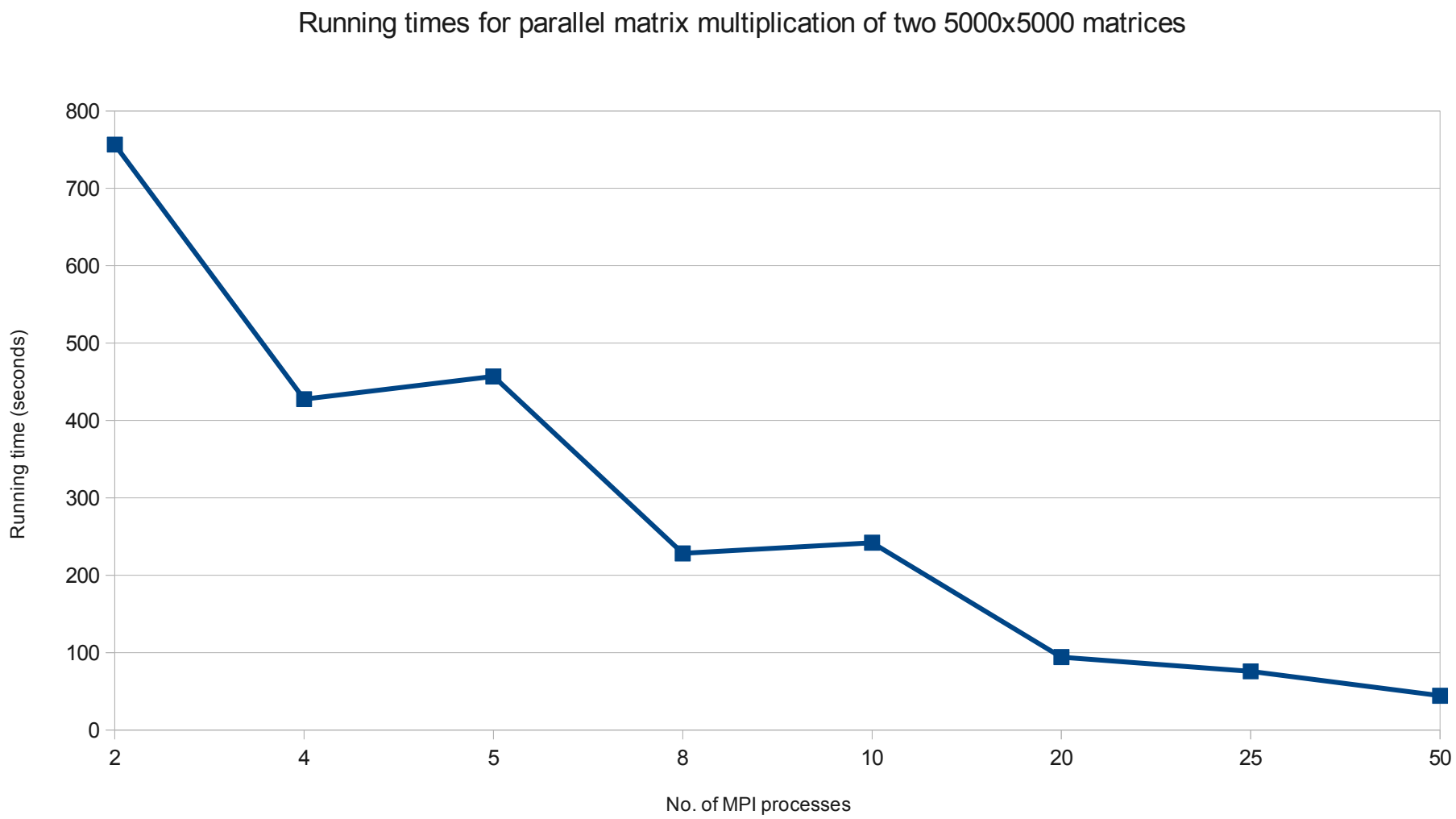
Test parameters

- Matrix dimensions ranged from 1000 to 10000 increasing in steps of 1000
- Number of rows and columns were equal. Both matrices had the same dimensions
- No. of processes used were 2,4,5,8,10,20,25,50
- I used the 32-core nodes with 256 GB of RAM
- For the 50 process test, used two 32-core nodes, for the others used only one 32-core node
- Each test was run 2 times

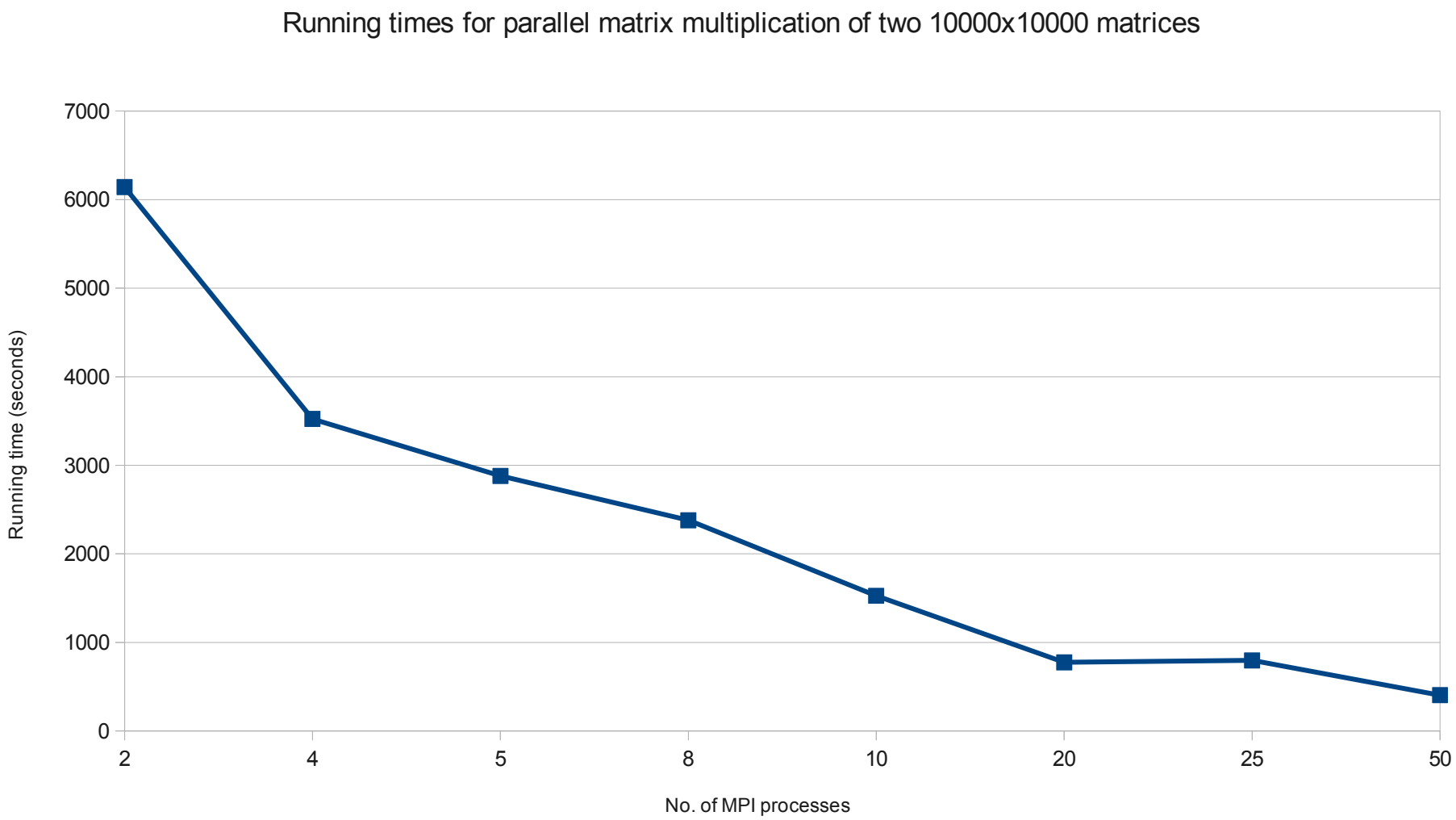
Test results



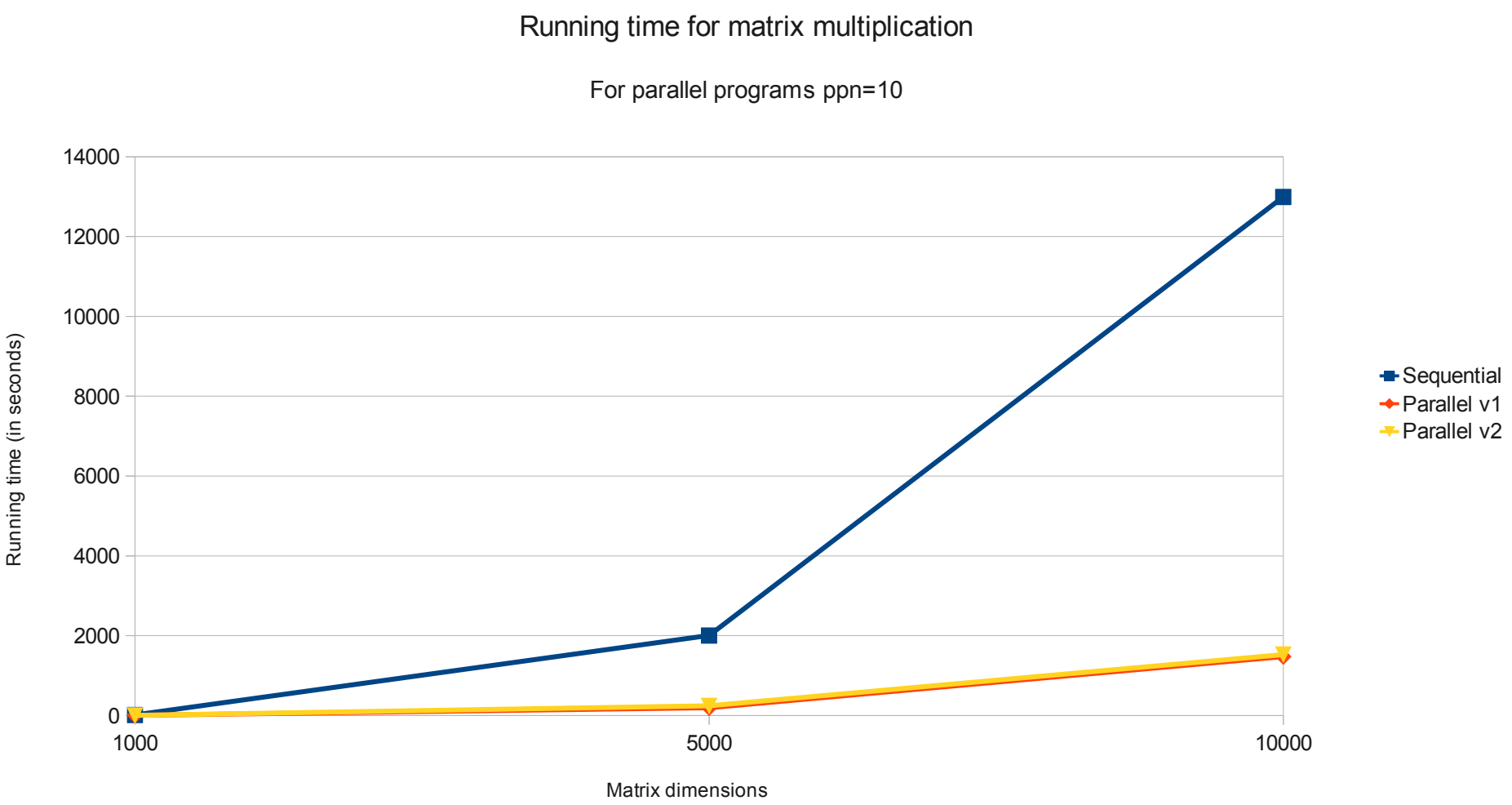
Test results (contd.)



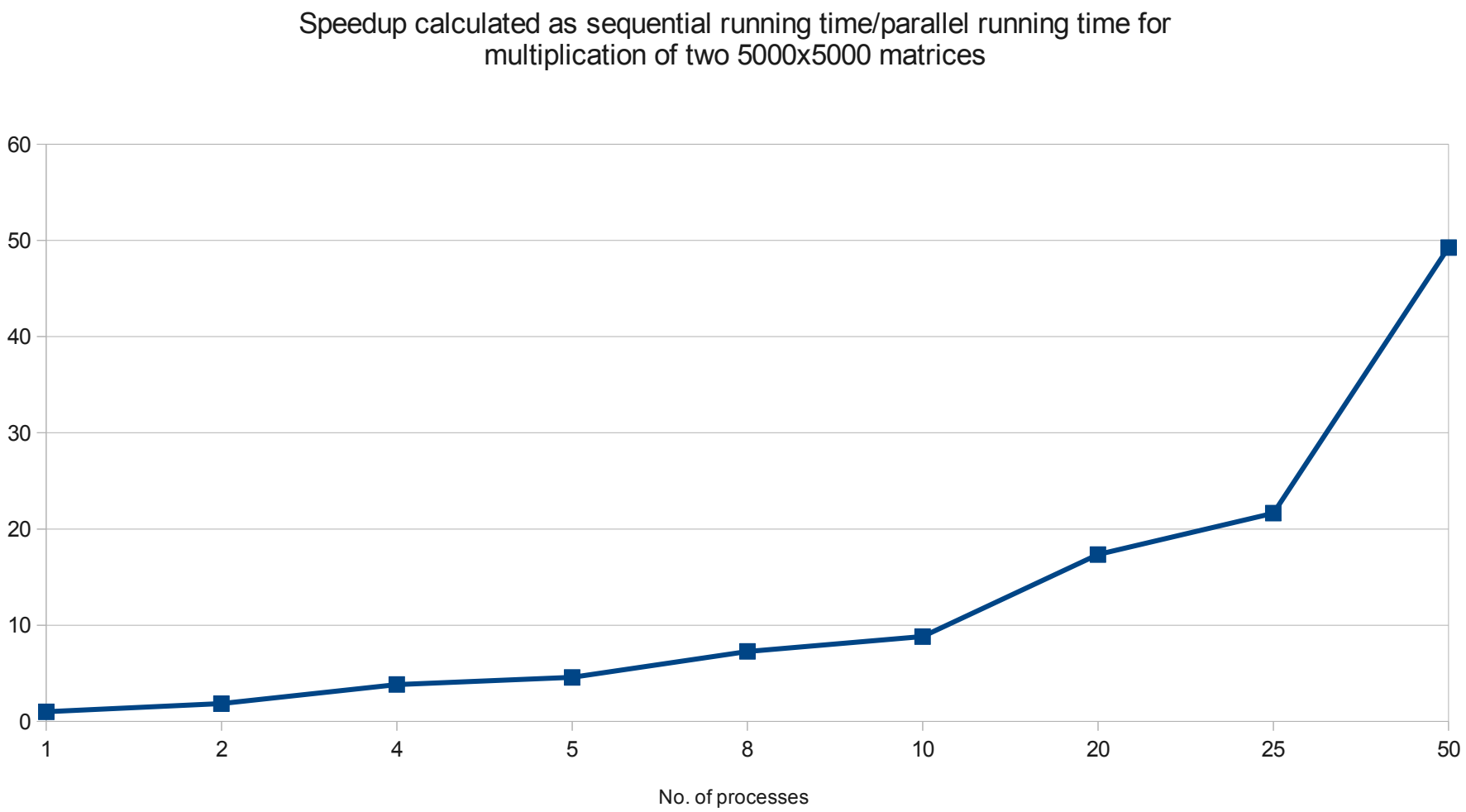
Test results (contd.)



Comparison


Running time for matrix multiplication
For parallel programs ppn=10
Running time (in seconds)
14000
12000
10000
8000
6000
4000
2000
0
1000
5000
10000
Matrix dimensions
Sequential
Parallel v1
Parallel v2

Speedup


Speedup calculated as sequential running time/parallel running time for multiplication of two 5000x5000 matrices
60
50
40
30
20
10
0
1
2
4
5
8
10
20
25
50
No. of processes

Future work

- Allow arbitrary matrix dimensions and any number of MPI processes
- Add the capability to read input from a file
- Use a more efficient sequential algorithm (like Strassen's matrix multiplication)
- Use 1 process per node to minimize communication. Use OpenMP to distribute work among the processors/cores in each node.

References

- http://en.wikipedia.org/wiki/Matrix_multiplication

The End