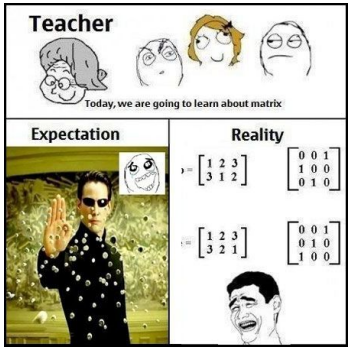
Matrix Multiplication Analysis

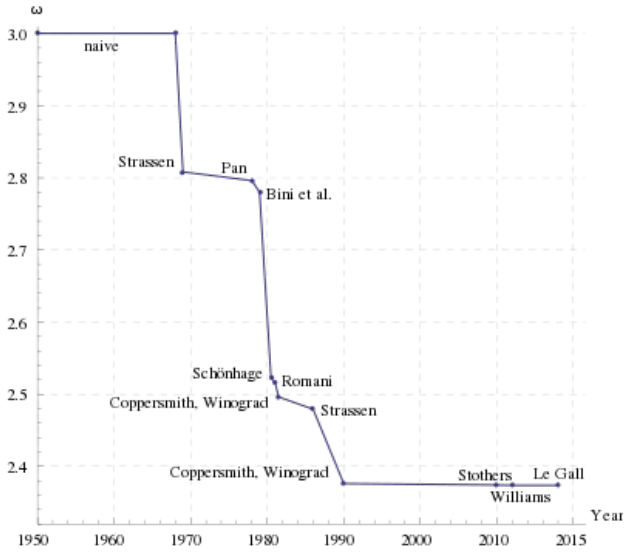


Nodebechukwu Okoye | Jakob Lopez

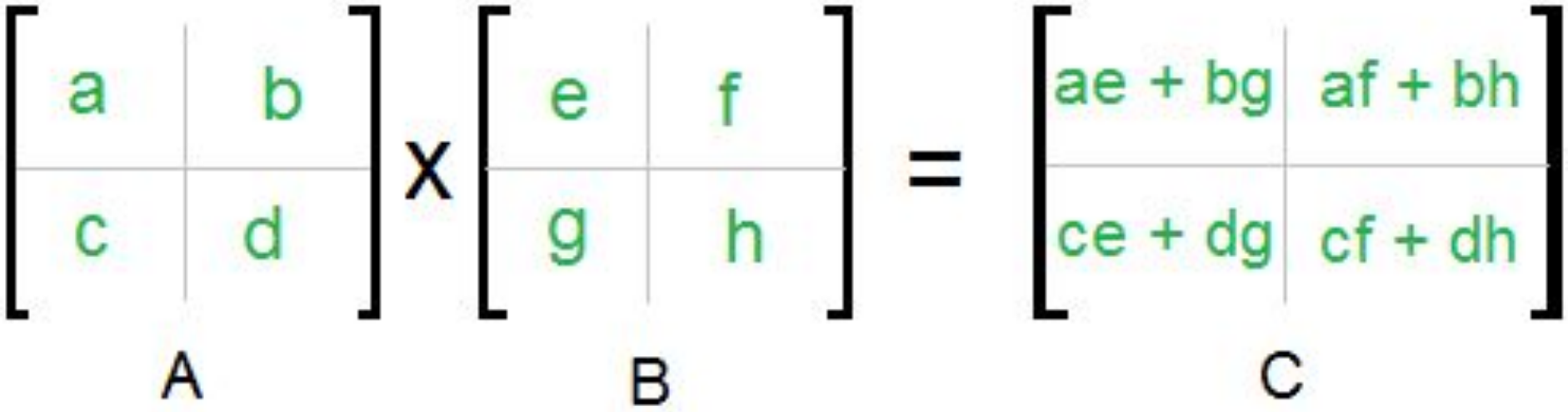
Introduction

There are a lot of ways to perform matrix multiplication

Examples are Iterative algorithm, Divide and conquer algorithm, Non-square matrices, Sub-cubic algorithms, Parallel and distributed algorithms.



Naive Algorithm



A, B and C are square metrices of size N x N
a, b, c and d are submatrices of A, of size N/2 x N/2
e, f, g and h are submatrices of B, of size N/2 x N/2

Strassen's Algorithm

$$p1 = a(f - h) \qquad p2 = (a + b)h$$
$$p3 = (c + d)e \qquad p4 = d(g - e)$$
$$p5 = (a + d)(e + h) \qquad p6 = (b - d)(g + h)$$
$$p7 = (a - c)(e + f)$$

The A x B can be calculated using above seven multiplications.
Following are values of four sub-matrices of result C

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} p5 + p4 - p2 + p6 & p1 + p2 \\ p3 + p4 & p1 + p5 - p3 - p7 \end{bmatrix}$$

A B C

A, B and C are square metrices of size N x N
a, b, c and d are submatrices of A, of size N/2 x N/2
e, f, g and h are submatrices of B, of size N/2 x N/2
p1, p2, p3, p4, p5, p6 and p7 are submatrices of size N/2 x N/2

Design Considerations

- Matrix sizes
  - 2048 x 2048(2^{11} x 2^{11})
  - 8192 x 8192(2^{13} x 2^{13})
  - 16384 x 16384(2^{14} x 2^{14})
- Processes (Strassen)
  - 1 for Serial
  - 7 for Parallel
- Threads (Naive)
  - 1 for Serial
  - 4 for Parallel (Control)
  - 8 for Parallel
  - 16 for Parallel

Stampede2 Hardware Specs

- Processor: Intel Xeon Phi 7250 ("Knights Landing")
- Cores per node: 68 cores on a single socket
- Threads per core: 4
- Threads per node: 272
- Clock rate: 1.4 GHz
- RAM: 96GB DDR4 plus 16GB high-speed MCDRAM
- Cache: 32KB L1 data cache per core; 1MB L2 per two-core tile
- 18 petaflop

Software and Tools

Python

- Mpi4py : for MPI
- Numpy : for all matrix calculation
- Multiprocessing : for threads
- Time: for timing the code

Stampede 2 : For the cluster

Collaboration

- Git and Github

Compilation and Execution Commands

- Submission
  - Sbatch <script name>

```
#
#   -- If you're running out of memory, try running
#      fewer tasks per node to give each task more memory.
#
#----------------------------------------------------

#SBATCH -J OkoyeLopezProjectNaiveParallel          # Job name
#SBATCH -o 16_16384.o%j       # Name of stdout output file
#SBATCH -e OkoyeLopezProjectNaiveParallel.e%j       # Name of stderr error file
#SBATCH -p normal          # Queue (partition) name
#SBATCH -N 1               # Total # of nodes
#SBATCH -n 1             # Total # of mpi tasks
#SBATCH -t 45:30:00        # Run time (hh:mm:ss)
#SBATCH --mail-user=
#SBATCH --mail-type=all    # Send email at begin and end of job
#SBATCH -A CMPS-5433-MWSU  # class project/account code

# Other commands must follow all #SBATCH directives...

module list
pwd
date

# Launch MPI code...
module spider python3/3.7.0
python OkoyeLopezProjectNaiveParallel.py
# ----------------------------------------------------
```

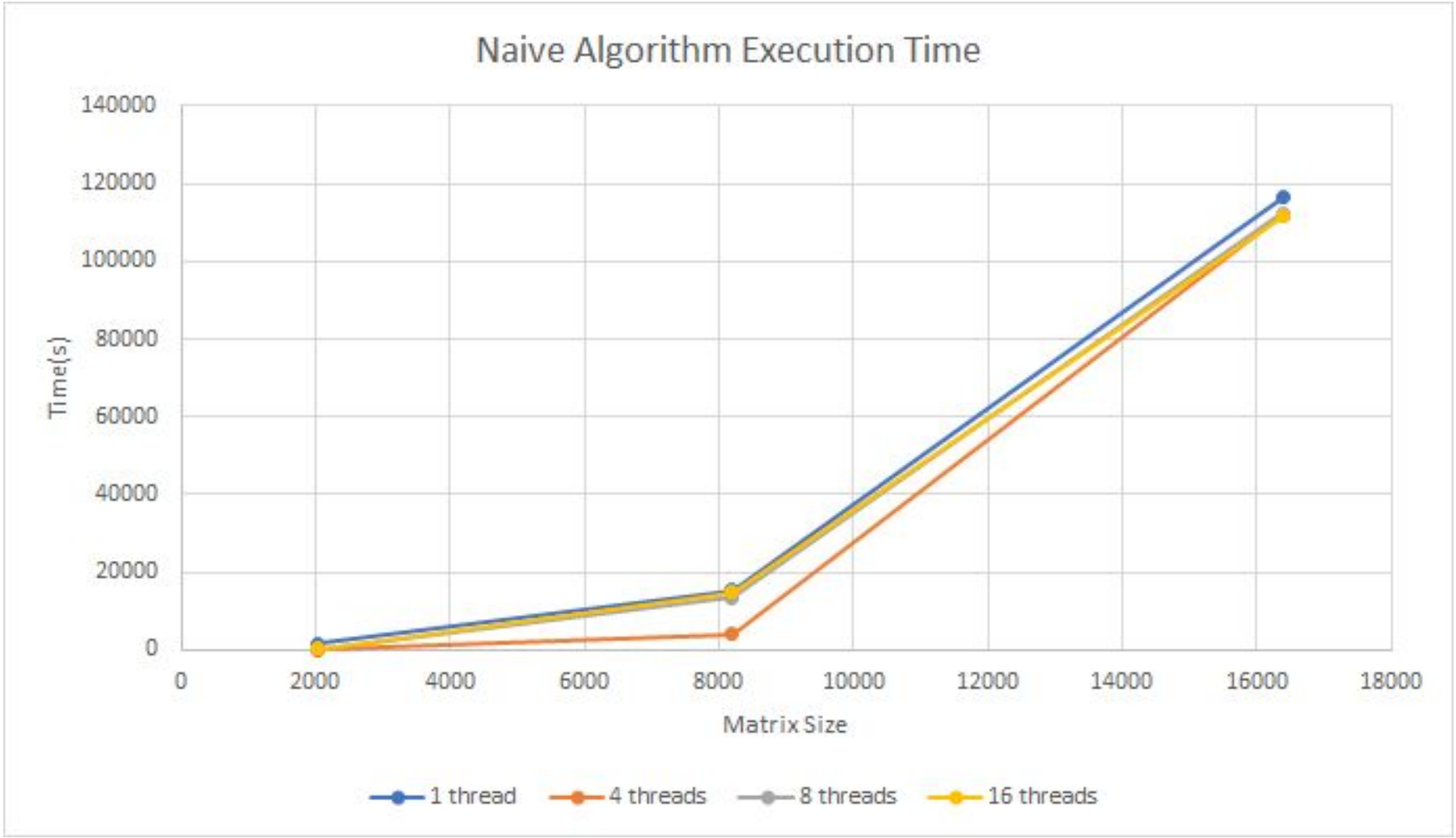
Execution Time(seconds): Naive

Matrix Size

Threads

	2048	**8192**	**16,384**
1	1722	15,299	116,464
4	68	4047	111,989
8	225	13,415	112,378
16	258	14,712	111,580

* 1 day = 86,400 seconds

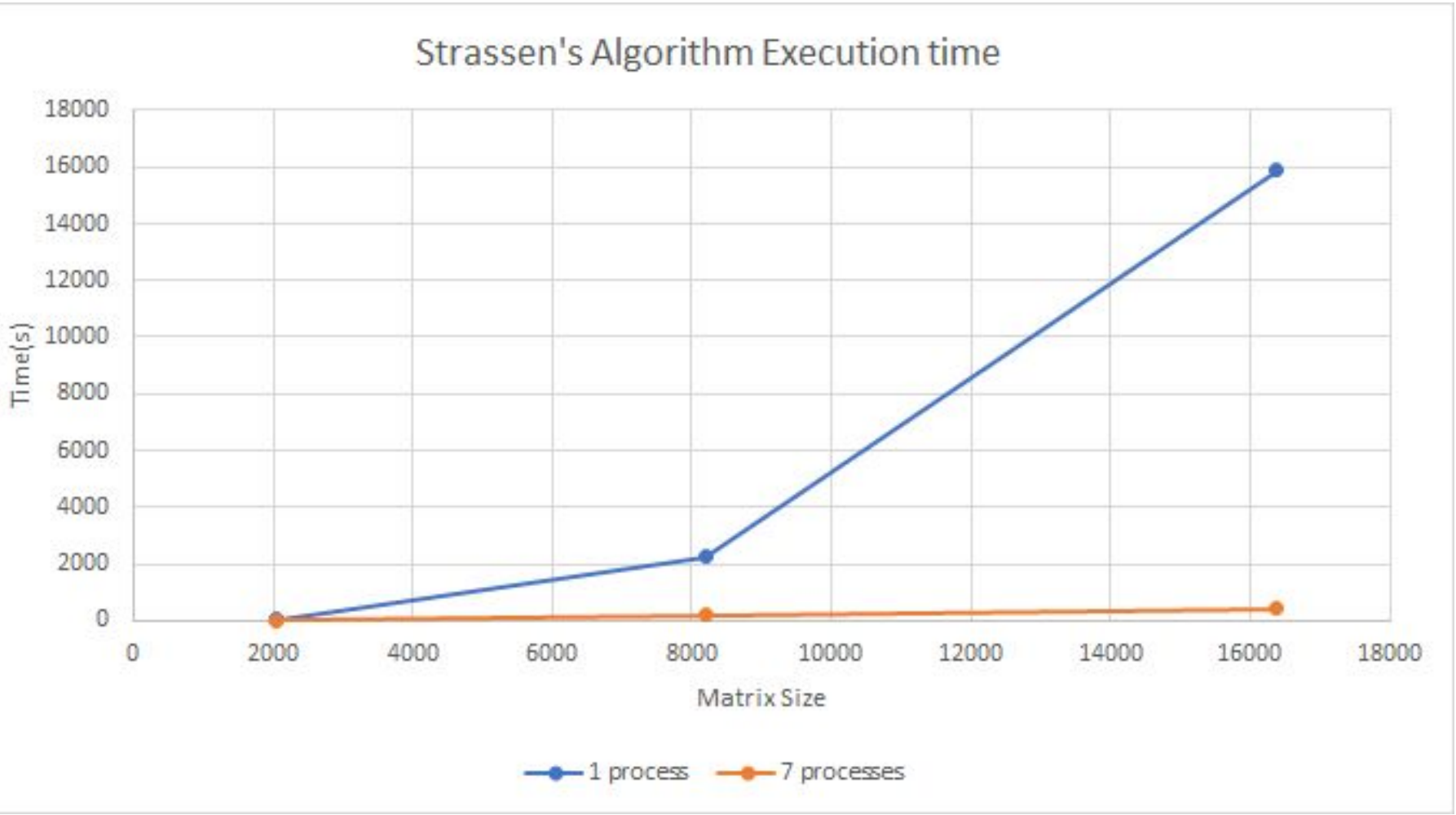

Naive Algorithm Execution Time
140000
120000
100000
80000
60000
40000
20000
0
Time(s)
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
Matrix Size
1 thread
4 threads
8 threads
16 threads

Execution Time(seconds): Strassen

Matrix Size

Processes

	2048	8192	16,384
1	45	2266	15,872
7	4	202	1407

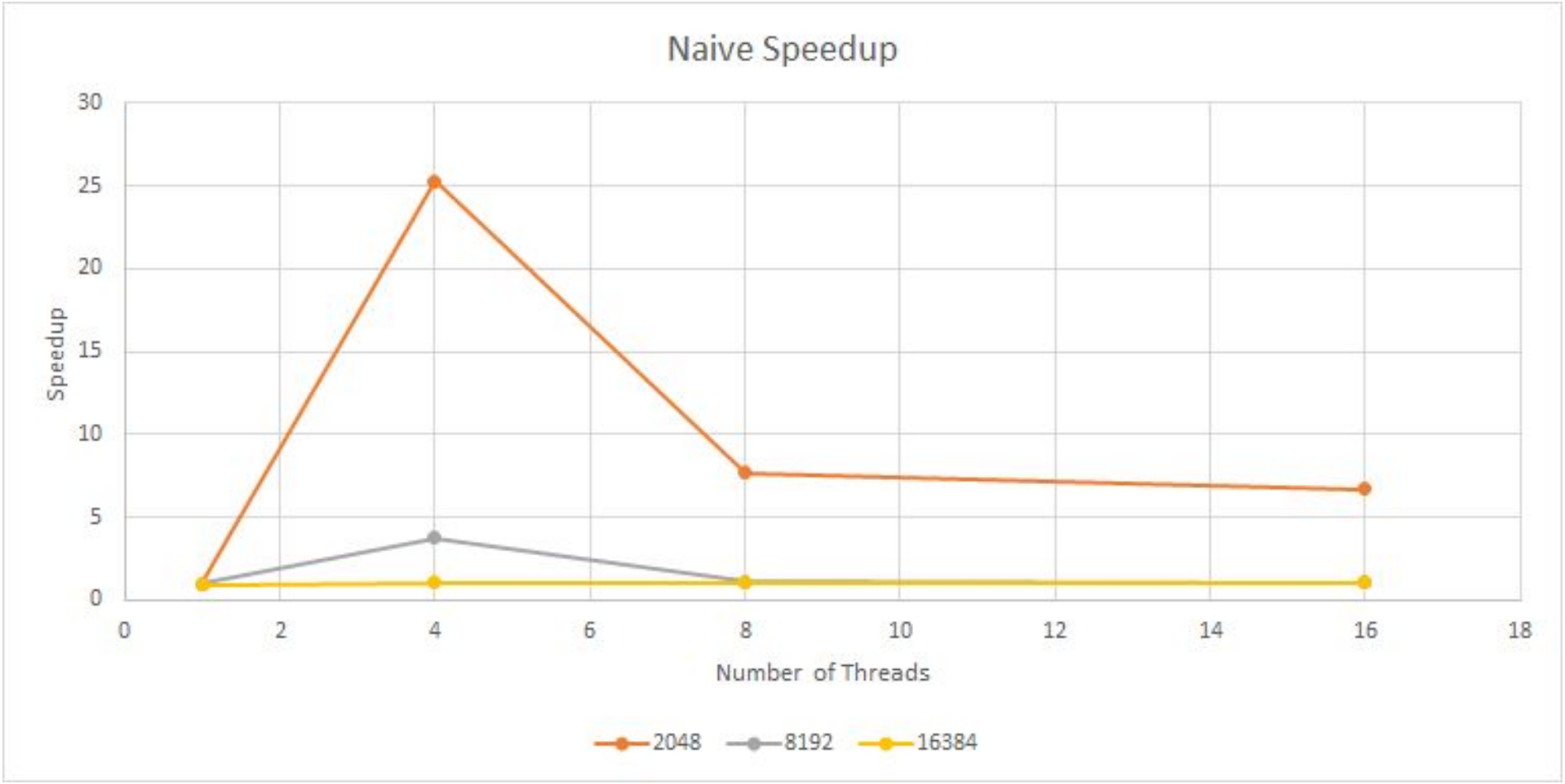

Strassen's Algorithm Execution time
Time(s)
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
Matrix Size
1 process
7 processes

Speedup: Naive

Matrix Size

Threads

	2048	8192	16,384
1	1	1	1
4	25.32353	3.780331	1.039959
8	7.653333	1.14044	1.036359
6	6.674419	1.039899	1.043771

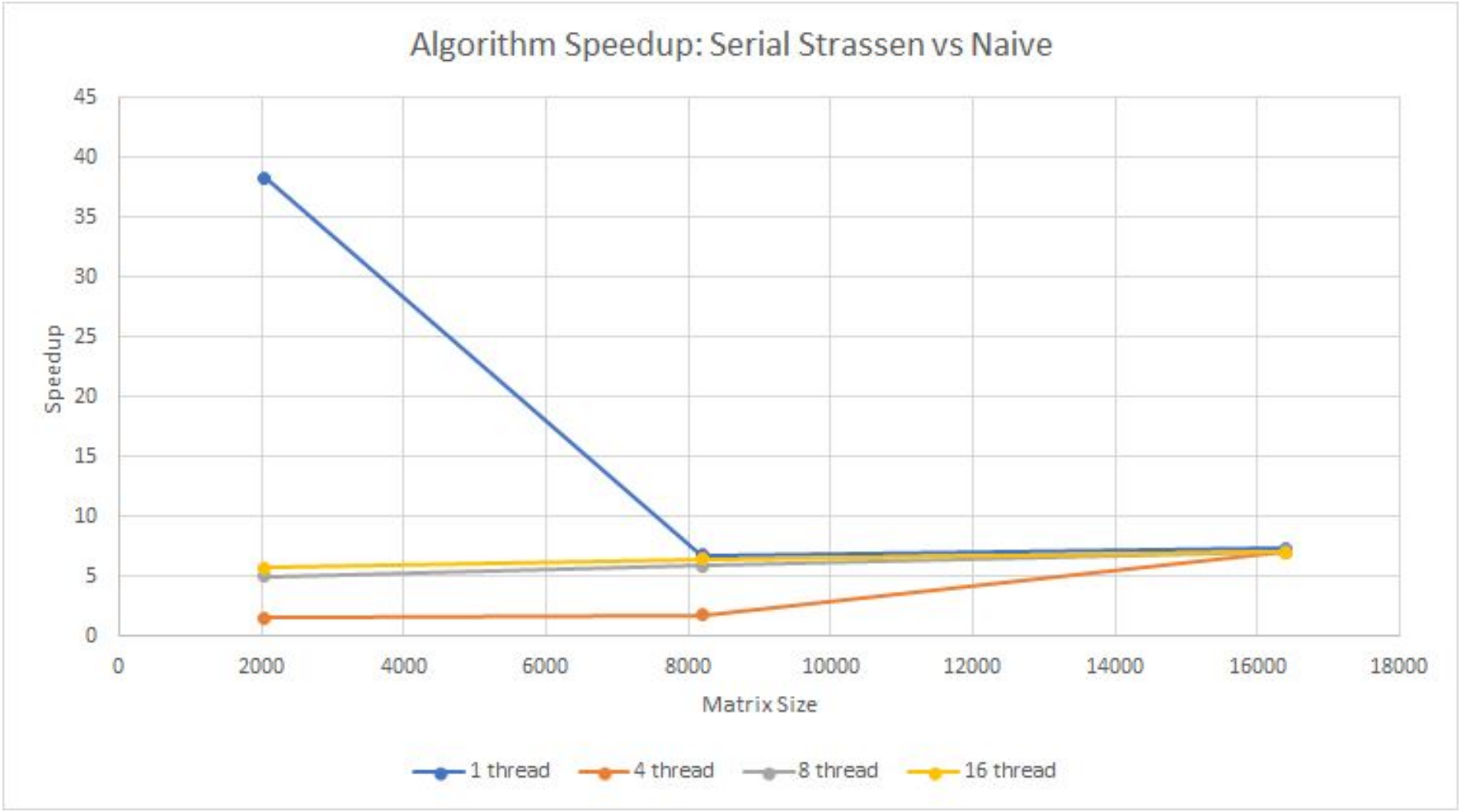

Naive Speedup
30
25
20
15
10
5
0
Speedup
0
2
4
6
8
10
12
14
16
18
Number of Threads
2048
8192
16384

Speedup: Serial Strassen

Matrix Size

Naive Threads

	2048	8192	16,384
1	38.26667	6.751544572	7.337702
4	1.511111	1.785966461	7.055759
8	5	5.920123566	7.080267
16	5.733333	6.492497793	7.02999

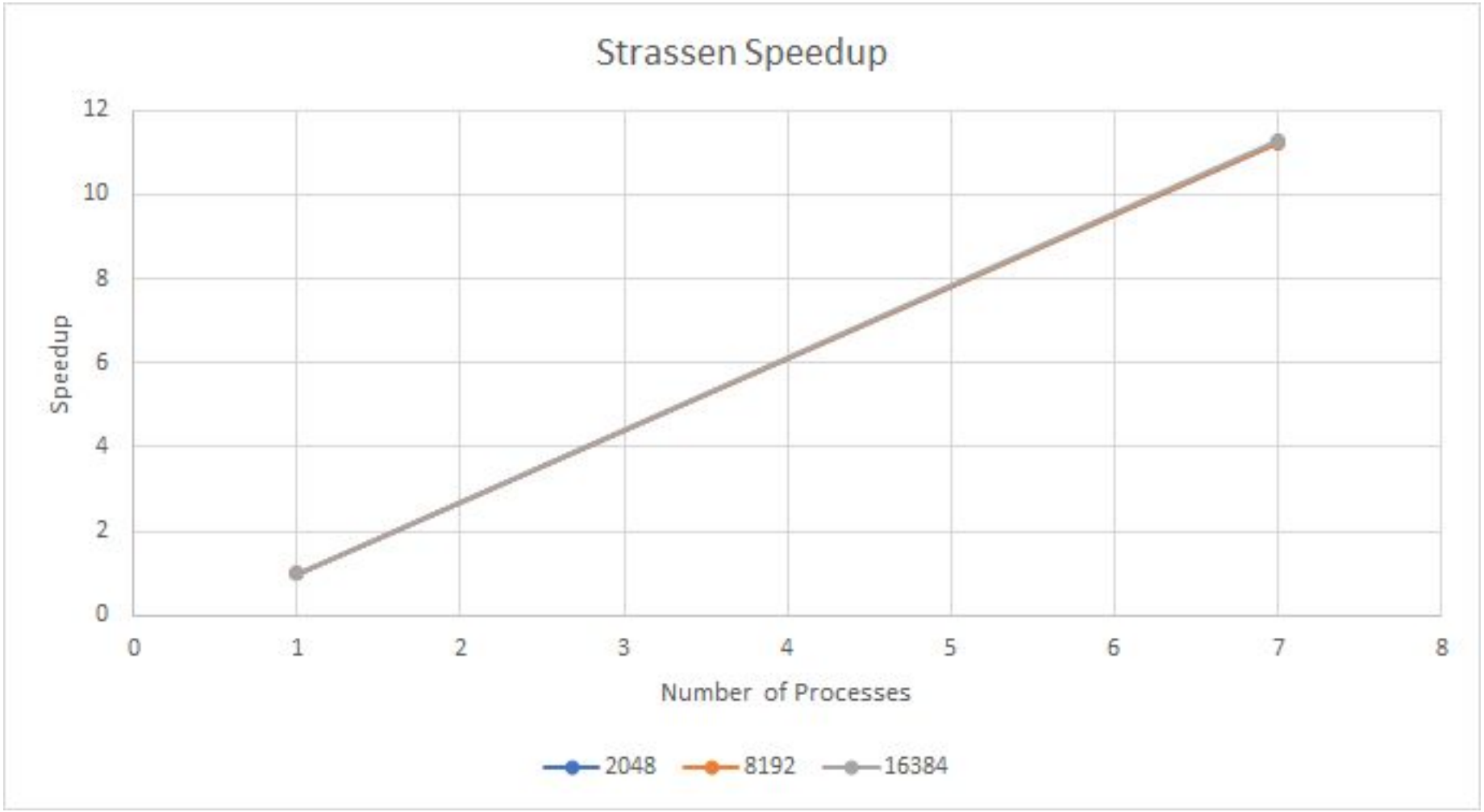

Algorithm Speedup: Serial Strassen vs Naive
Speedup
45
40
35
30
25
20
15
10
5
0
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
Matrix Size
1 thread
4 thread
8 thread
16 thread

Speedup: Strassen

Matrix Size

Processes

	2048	8192	16,384
1	1	1	1
7	11.25	11.21782	11.28074

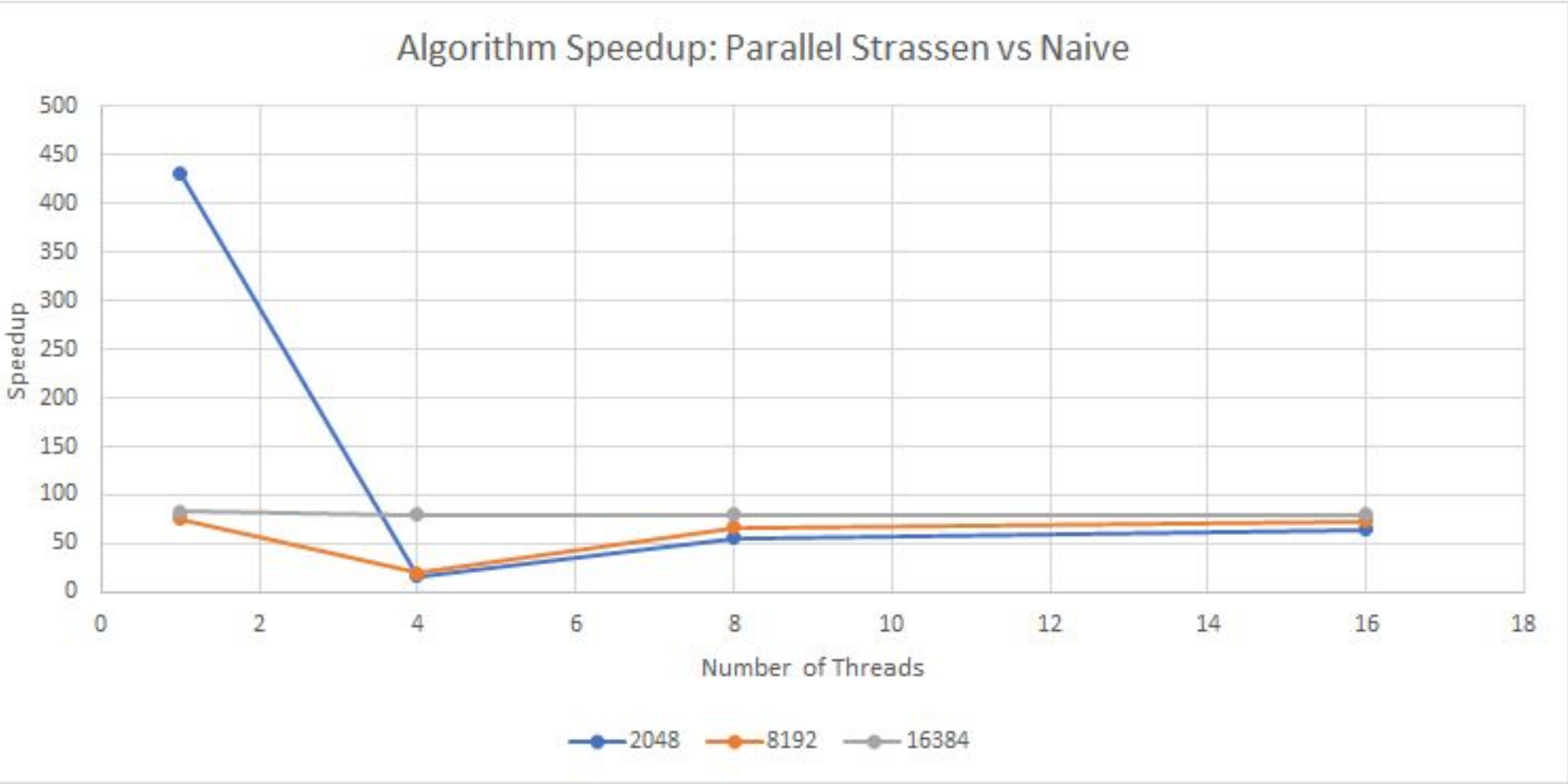

Strassen Speedup
12
10
8
6
4
2
0
Speedup
0
1
2
3
4
5
6
7
8
Number of Processes
2048
8192
16384

Speedup: Parallel Strassen

Matrix Size

Naive Threads

	2048	8192	16,384
1	430.5	75.73762	82.7747
4	17	20.03465	79.59417
8	56.25	66.41089	79.87065
16	64.5	72.83168	79.30348

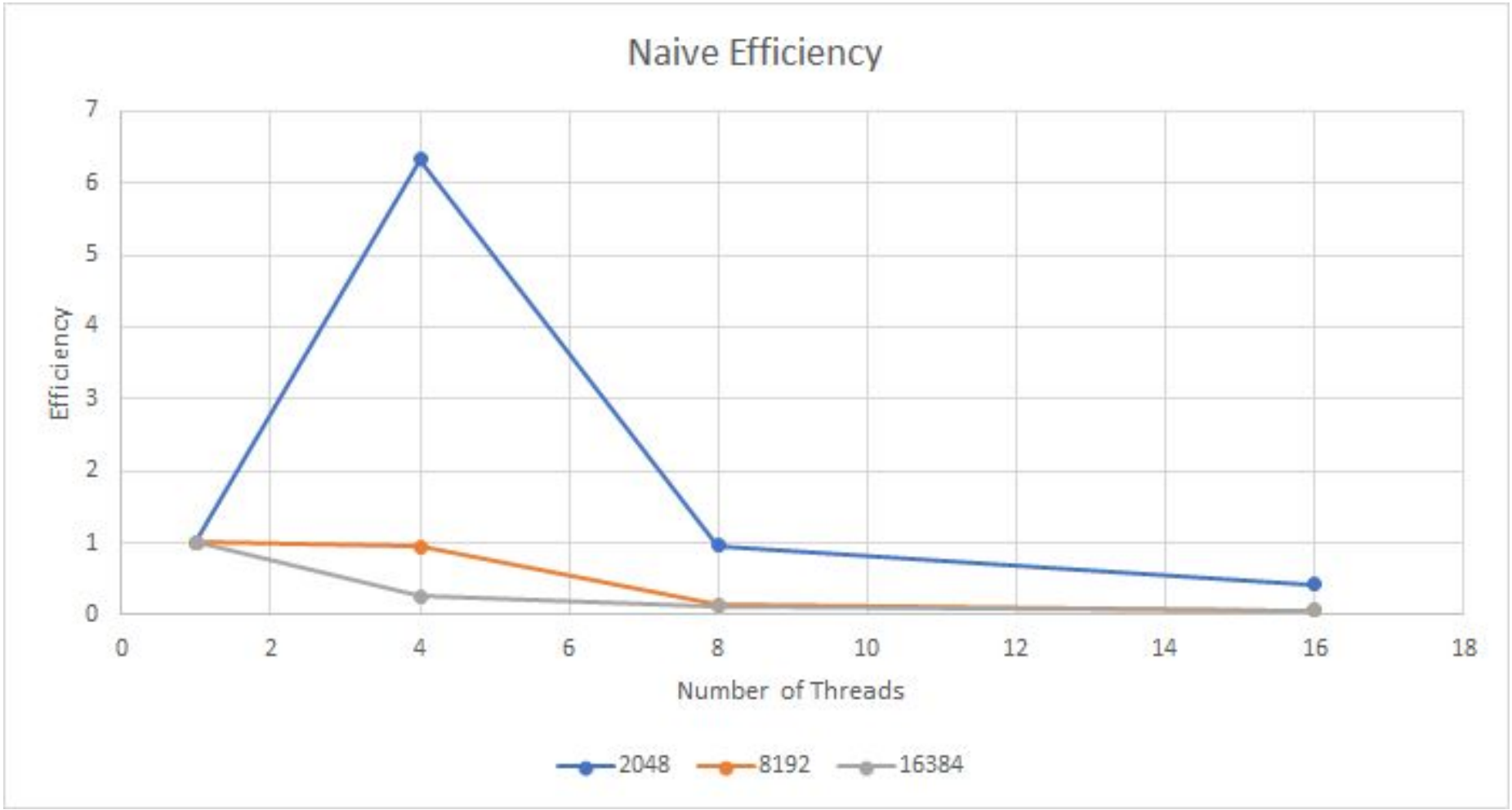

Algorithm Speedup: Parallel Strassen vs Naive
Speedup
500
450
400
350
300
250
200
150
100
50
0
0
2
4
6
8
10
12
14
16
18
Number of Threads
2048
8192
16384

Efficiency: Naive

Matrix Size

Threads

	2048	8192	16,384
1	1	1	1
4	6.330882	0.945083	0.25999
8	0.956667	0.142555	0.129545
16	0.417151	0.064994	0.065236

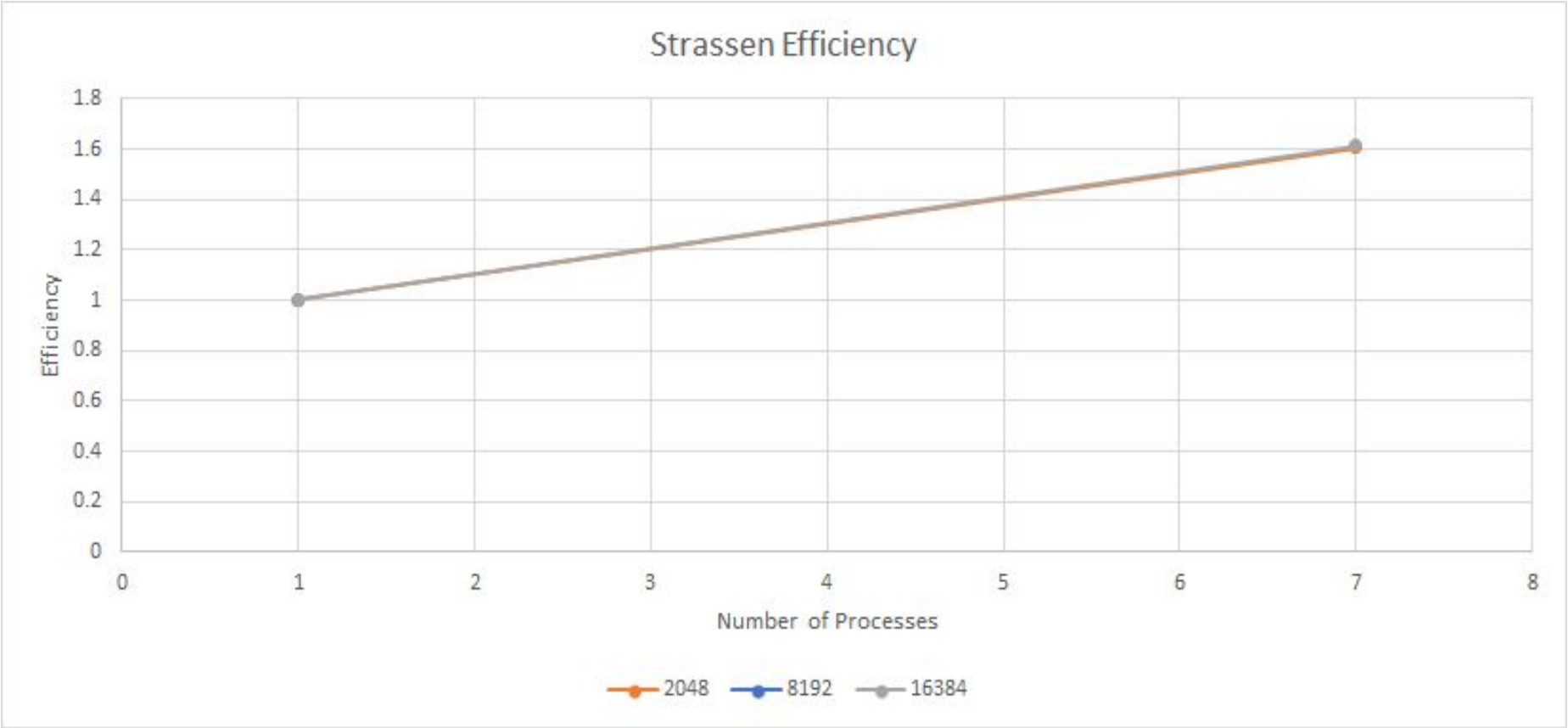

Naive Efficiency
Efficiency
Number of Threads
0
1
2
3
4
5
6
7
0
2
4
6
8
10
12
14
16
18
2048
8192
16384

Efficiency: Strassen

Matrix Size

Processes

	2048	8192	16,384
1	1	1	1
7	1.607143	1.602546	1.611534

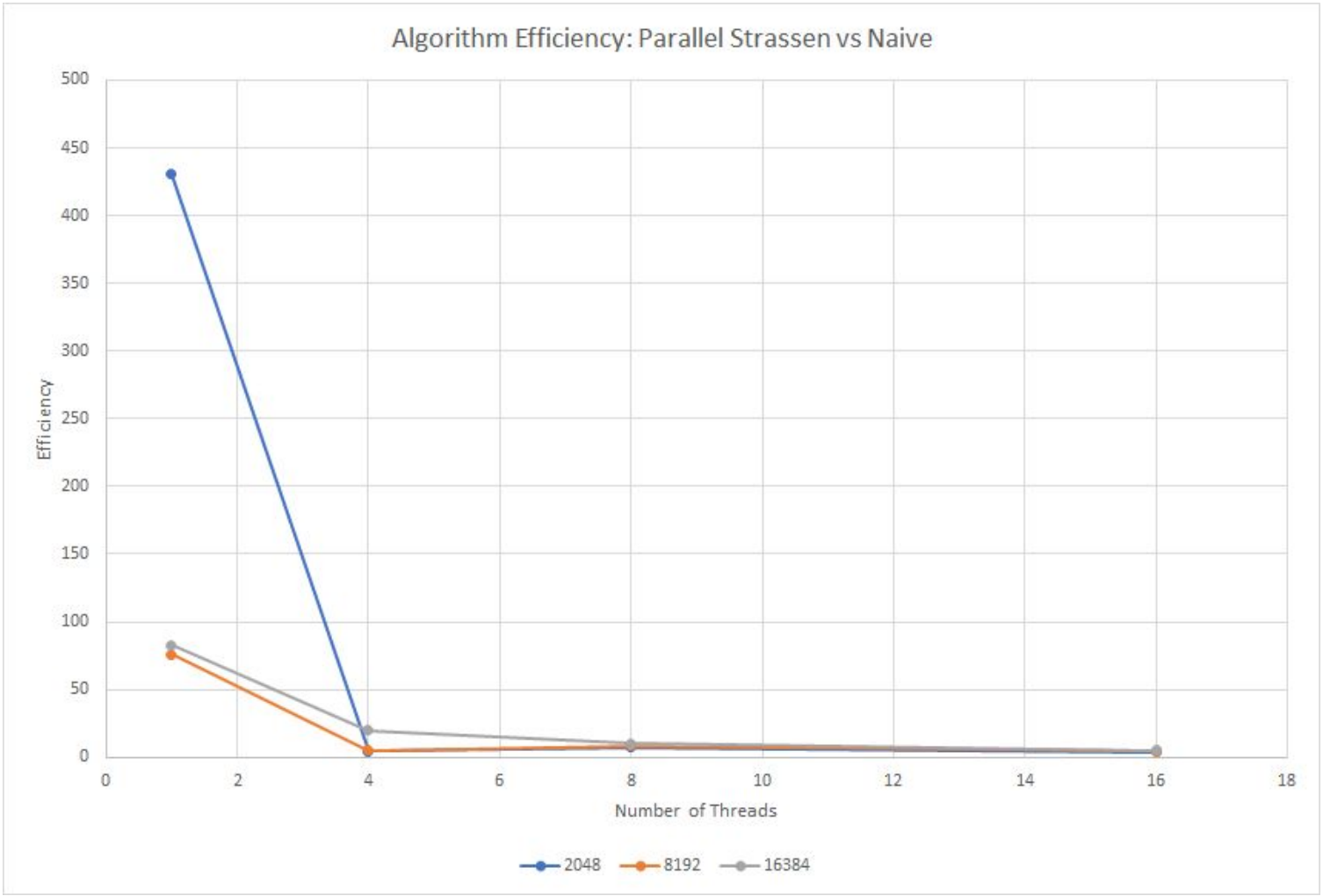

Strassen Efficiency
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
Efficiency
0
1
2
3
4
5
6
7
8
Number of Processes
2048
8192
16384

Efficiency: Parallel Strassen

Matrix Size

Naive Threads

	2048	8192	16,384
1	430.5	75.74	82.7747
4	4.25	5.009	19.89854
8	7.03125	8.301	9.983831
16	4.03125	4.552	4.956468


Algorithm Efficiency: Parallel Strassen vs Naive
Efficiency
500
450
400
350
300
250
200
150
100
50
0
0
2
4
6
8
10
12
14
16
18
Number of Threads
2048
8192
16384

Important Variables

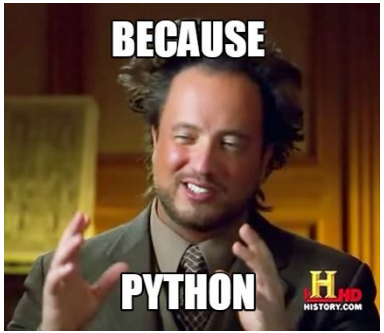
- Size of matrix
  - n
- Number of threads
  - num_workers
- Number of processes
  - Change in Parallel Strassen Script, however not advised

Obstacles

- New language for us to use in Stampede2
- Understanding Strassen’s Algorithm
- Massive waiting time
- Inefficiency with Microsoft Excel

Conclude

- Naive algorithm is not the only matrix multiplication algorithm
- Strassen's algorithm is the most practical big matrix multiplication technique
- Parallel processing is made easy by Python



https://github.com/JakobLopez/ParallelMatrixMultiplication

References

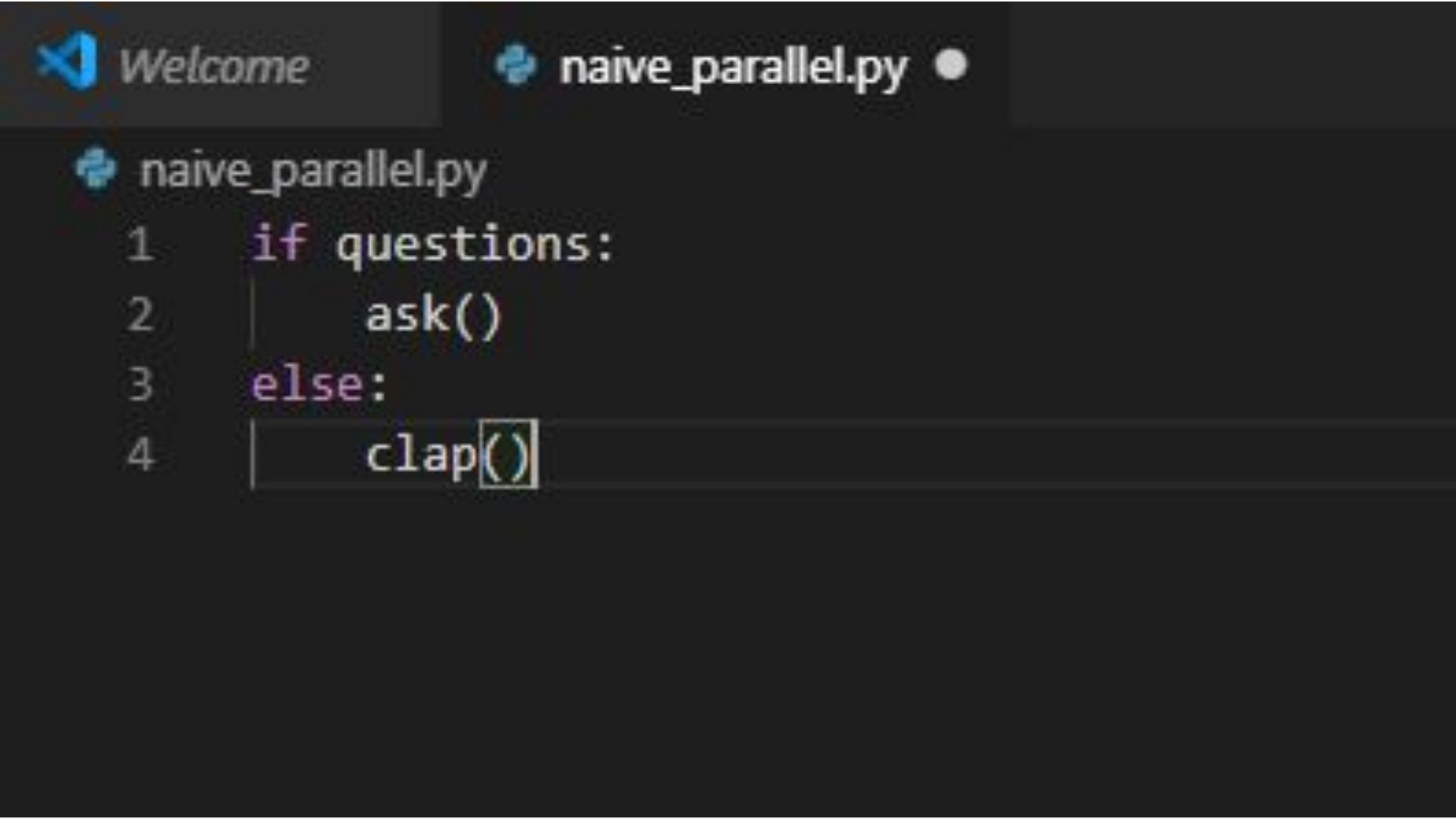
https://www.cac.cornell.edu/education/training/StampedeJan2017/Python_R_HPC_Workshop.pdf

https://mpi4py.readthedocs.io/en/stable/

https://docs.python.org/2/library/multiprocessing.html

https://en.wikipedia.org/wiki/Matrix_multiplication_algorithm

https://www.geeksforgeeks.org/strassens-matrix-multiplication/


Welcome
naive_parallel.py
naive_parallel.py
1 if questions:
2 ask()
3 else:
4 clap()