Matrix Multiplication in Parallel

Presented by: Varsha Ganesh

Guided by:
Professor Dr. Russ Miller
Dr. Matt Jones

University at Buffalo The State University of New York

Table of Contents

- Project Goals
- Amdahl’s Law
- Problem Definition
- Sequential Algorithms
- Sequential Algorithm - Run time Analysis
- Parallel Algorithm
- Results
- Re-evaluation of Amdahl’s Law
- Challenges Faced

Project Goals

- Design, implement, and analyze the parallel solution of interest on modern large-scale multiprocessor/multi-core systems.

- Getting Accustomed to real life high performance multiprocessor computing environment.

- Solve Matrix Multiplication Problem in Parallel and compare the results with a sequential implementation to comment on Amdahl’s law.

Amdahl's Law

- The maximum speedup achievable by an *n*-processor machine is given by $S_n \leq 1/[f + (1 - f)/n]$, where f is the fraction of operations in the computation that must be performed sequentially.
- So, for example, if five percent of the operations in a given computation must be performed sequentially, then the speedup can never be greater than 20, *regardless of how many processors are used.*
- Therefore, just a small number of sequential operations can significantly limit the speedup of an algorithm on a parallel machine.

Problem Definition - Matrix Multiplication

Given a matrix A(N ×N) and a matrix B(N × N), the matrix C(N x N) resulting from the multiplication of matrices A and B, C = A × B is computed as Follows.

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

Note: To Compute One Value in C matrix, we have to peform N multiplications and (N-1) additions. Thus to compute the N^2 values in C Matrix, we have to peform O(N^3) operations.

Sequential Algorithms

Brute Force 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
```

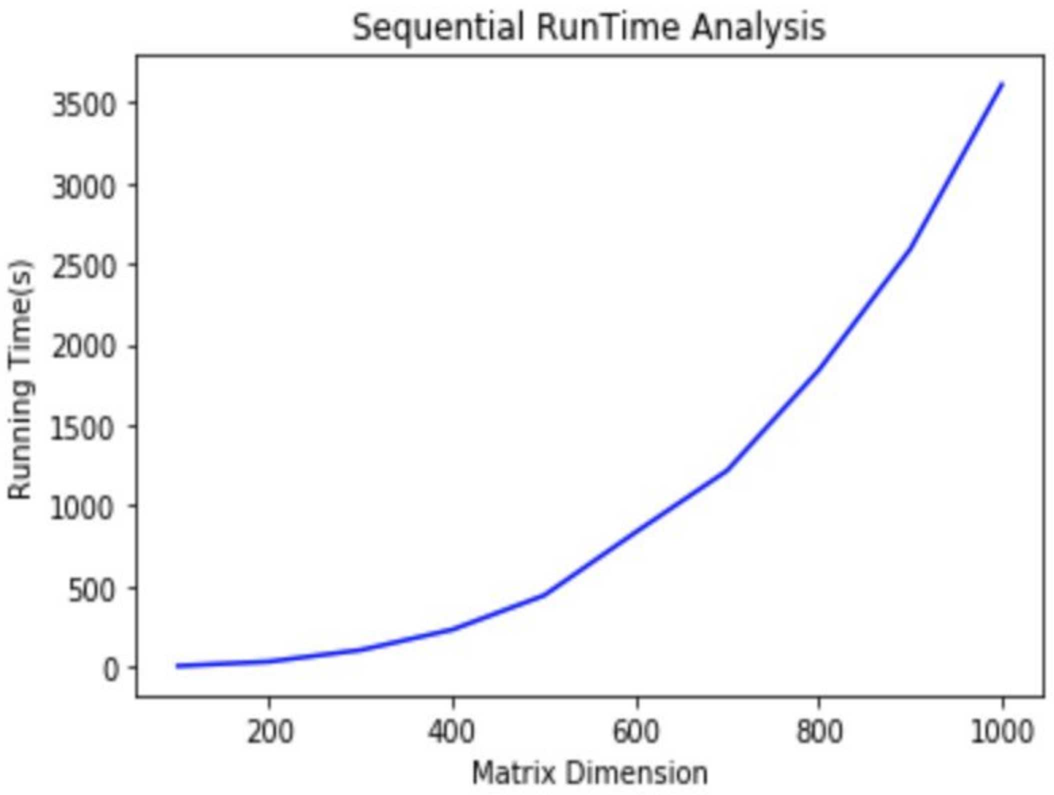
Strassen's Improvised Algorithm

$$\underbrace{\begin{bmatrix} a & b \\ c & d \end{bmatrix}}_{A} \times \underbrace{\begin{bmatrix} e & f \\ g & h \end{bmatrix}}_{B} = \underbrace{\begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}}_{C}$$

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

Sequential Algorithm - Runtime Analysis

No of Processors	Matrix Dimension	Running Time(s)
1	100 X 100	3.36
1	200 x 200	30.49
1	300 x 300	102.11
1	400 x 400	228.93
1	500 x 500	440.95
1	600 x 600	833.29
1	700 x 700	1216.29
1	800 x 800	1839.21
1	900 x 900	2591.42
1	1000 x 1000	3612.70

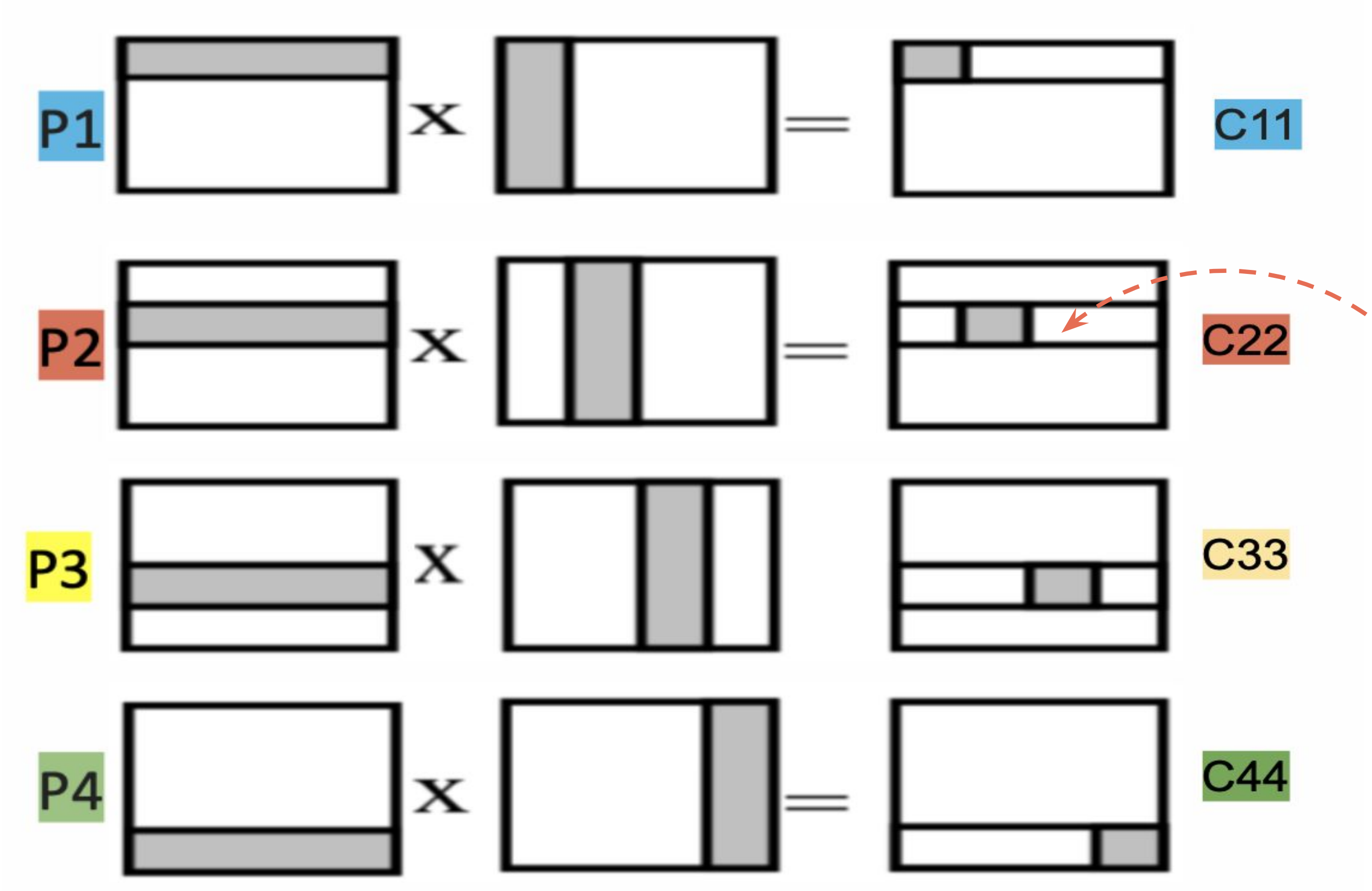


Block Striped Matrix Decomposition - A Parallel Approach

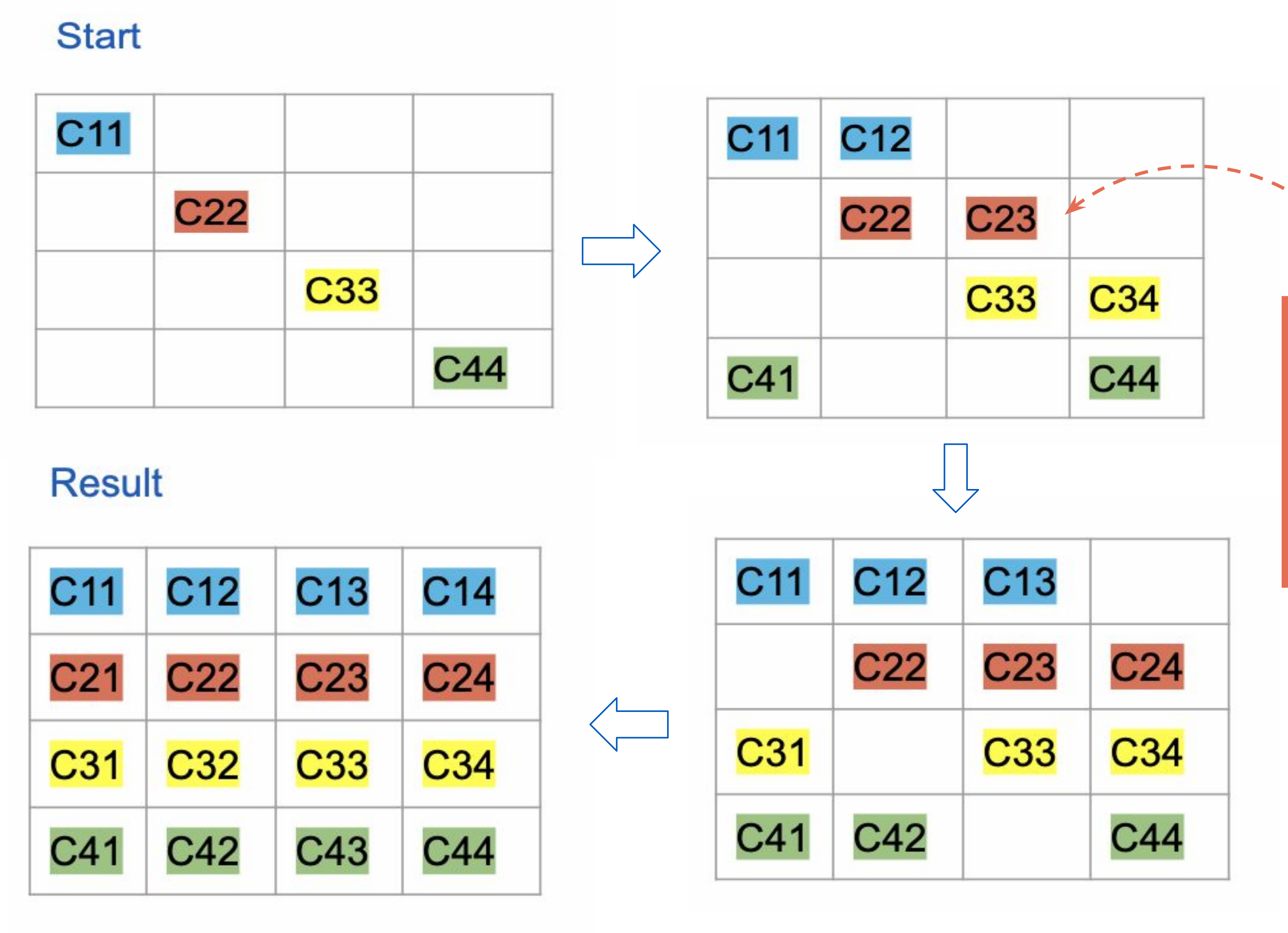
```
1.   Divide A_matrix along its Rows as per Number of Processors
2.   Divide B_matrix along its Columns as per Number of Processors
3.   All Processors in Parallel loads A[rank] and B[rank]
4.   For i in (rank,N):
          C[rank,i] = A_matrix[rank] *B_matrix[i]
          send(rank,B_matrix[i])
          B_matrix[i] = receive(rank)
5.   For i in(0, rank):
          C[rank,i] = A_matrix[rank] *B_matrix[i]
          send(rank,B_matrix[i])
          B_matrix[i] = receive(rank)
6.    Write the Results of C

def send(rank,B):
     if rank==0 : send(B,send = N-1, tag= N-1)
     else: send(B,send = rank-1, tag= rank-1)

def receive(rank):
     if rank==N-1:
          B_matrix = receive(source=0,tag = rank)
     else:
          B_matrix = receive(source=rank+1,tag = rank)
```

Note: This is a cyclic operation, even though there are two loops the Number of iterations is only N. This will become more clear with the pictorial representation. Here N is the number of Processors.

Data Distribution - After First Iteration



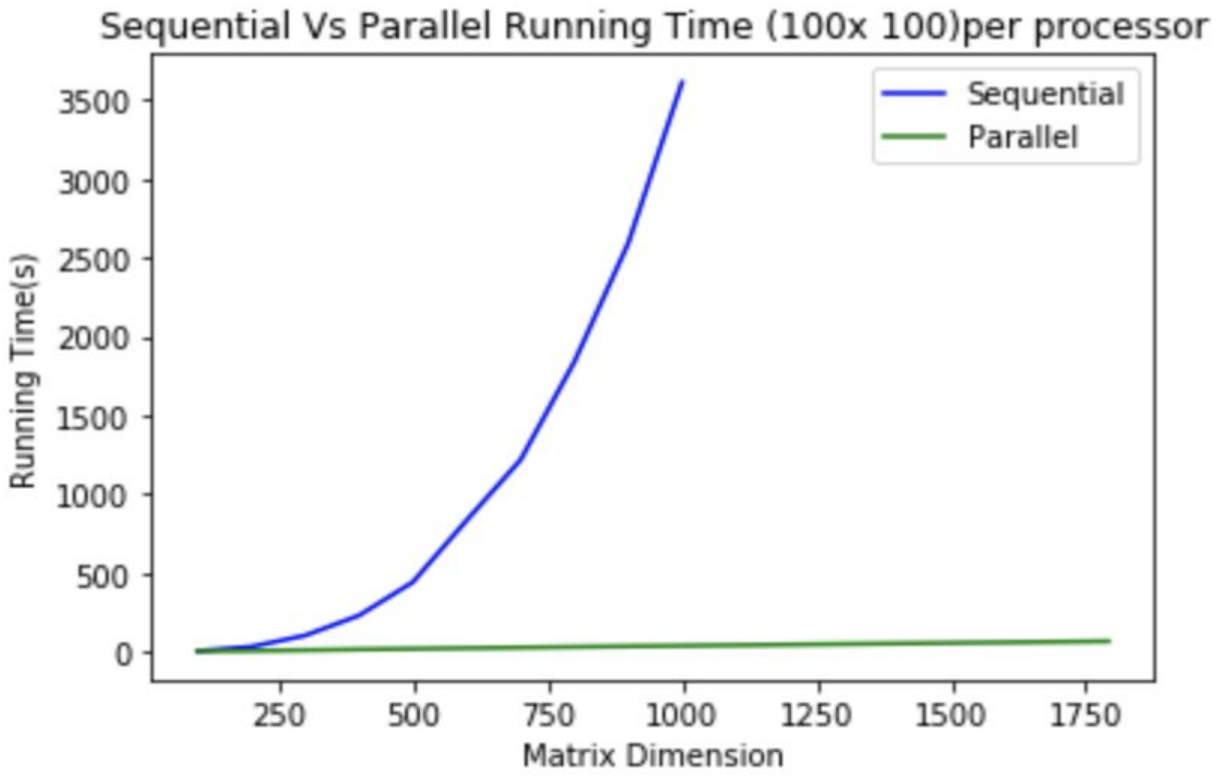
At Every step, each of the four processors compute the next block of C in their row in a cyclic fashion .To produce C, as depicted in the following slide.



At Every step, each of the fou processors compute the next block of C in their row in a cyclic fashion . The Number of steps = No of Processors.

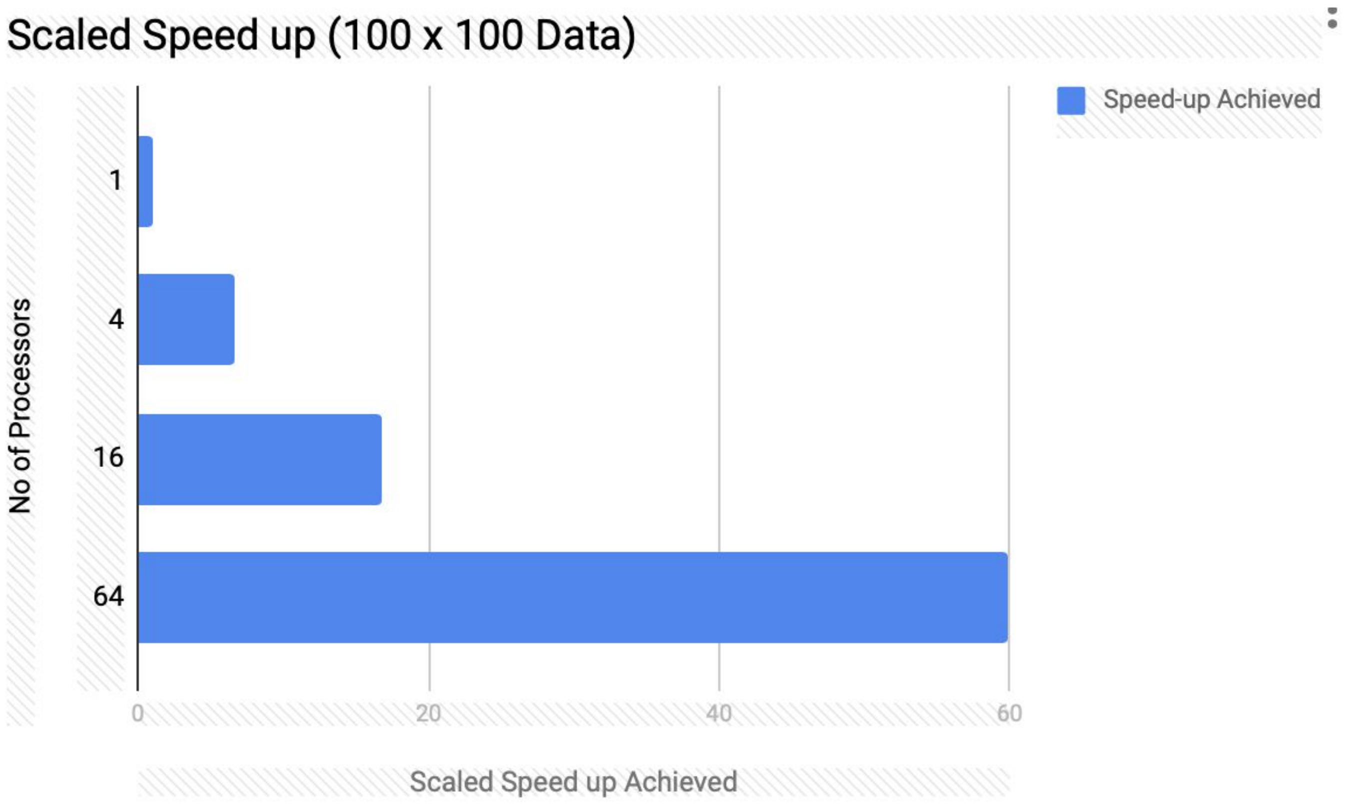
Parallel Approach (100 x 100 Data Items/ Processor)

No of Processors	Matrix Size	Running Time(s)
1	100 x 100	3.36
4	200 x 200	4.58
16	400 x 400	13.65
64	832 x 832	30.77
256	1792 x 1792	65.34



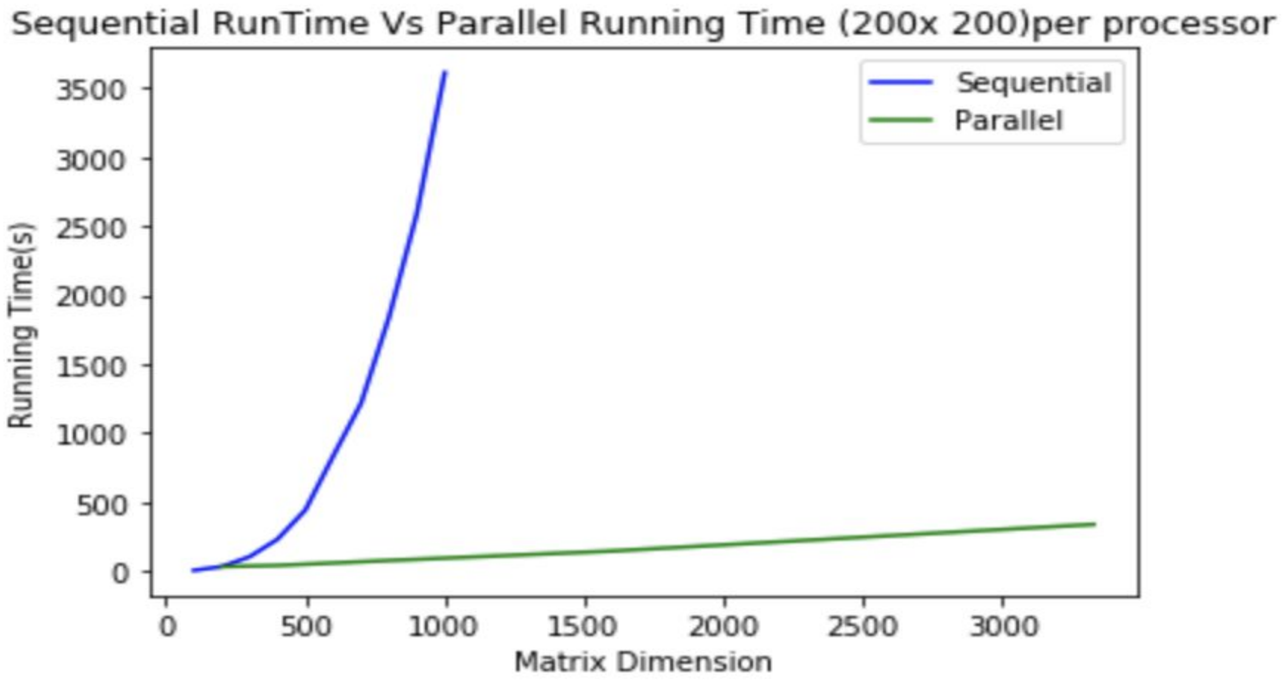
Scaled Speed-up Achieved (100 x 100 Data Items/ Processor)

No of Processors	Speed-up Achieved
1	1
4	6.65
16	16.77
64	59.77



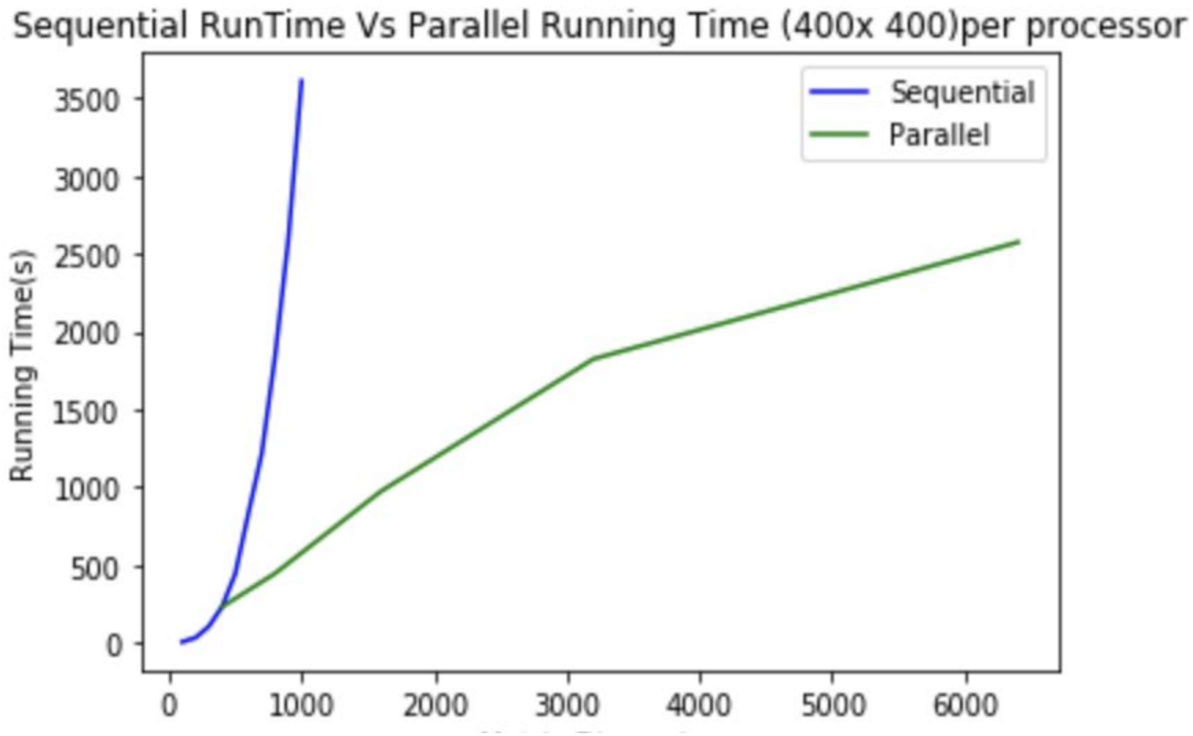
Parallel Approach (200 x 200 Data Items/ Processor)

No of Processors	Matrix Size	Running Time(s)
1	200 x 200	30.49
4	400 x 400	38.99
16	800 x 800	75.01
64	1600 x 1600	143.56
256	3328 x 3328	336.87



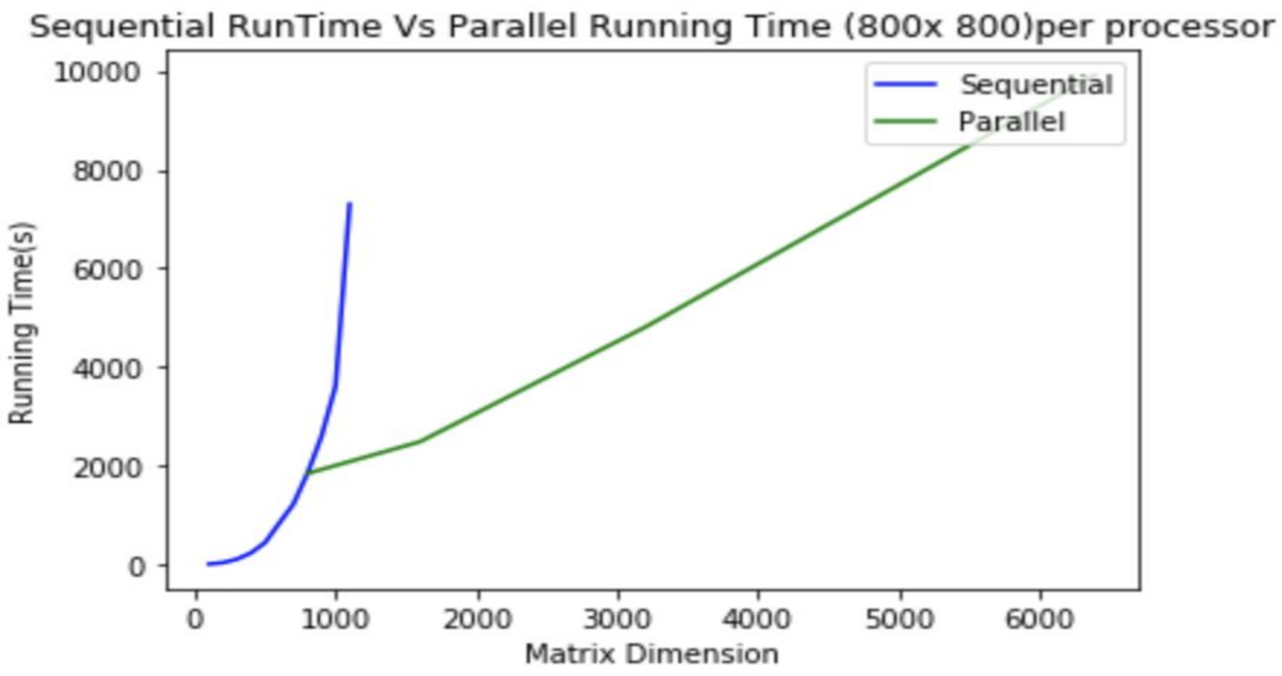
Parallel Approach (400 x 400 Data Items/ Processor)

No of Processors	Matrix Size	Running Time(s)
1	400 x 400	228.93
4	800 x 800	444.03
16	1600 x 1600	973.56
64	3200 x 3200	1823.54
256	6400 x 6400	2574.67



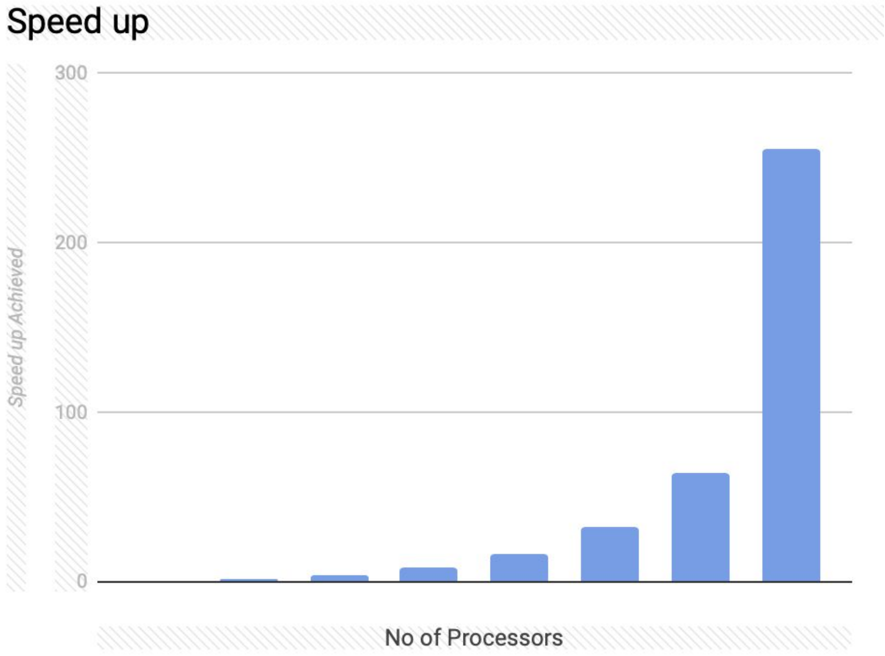
Parallel Approach (800 x 800 Data Items/ Processor)

No of Processors	Matrix Size	Running Time(s)
1	800 x 800	1839.21
4	1600 x 1600	2481.57
16	3200 x 3200	4802.58
64	6400 x 6400	9919.36



Speed up Factor while keeping Data Fixed (800 x 800)

No of Processors	Matrix Size	Running Time(s)	Speed up Factor
1	800 x 800	1839.21	1
2	800 x 800	586.26	3.138
4	800 x 800	288.48	6.37
8	800 x 800	157.72	11.66
16	800 x 800	72.93	25.21
32	800 x 800	40.35	45.58
64	832 x 832	30.77	59.77
256	768 x 768	5.846	314.60

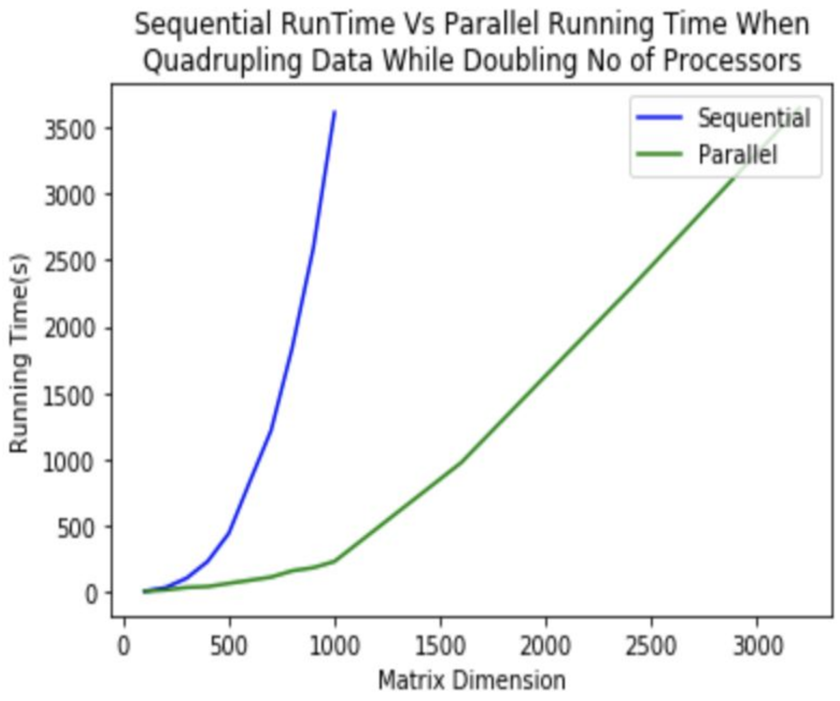


Note: This is not very different from the scaled speed up Factor, which means that we need to re-evaluate Amdahl's law.

Quadrupling Data While Doubling No of Processors

No of Processors	Matrix Size	Running Time(s)
1	100 x 100	3.36
2	200 x 200	13.27
3	300 x 300	32.02
4	400 x 400	38.99
5	500 x 500	63.10
6	600 x 600	86.45
7	700 x 700	110.184
8	800 x 800	157.72

No of Processors	Matrix Size	Running Time(s)
9	900 x 900	181.445
10	1000 x 1000	228.20
16	1600 x 1600	973.56
24	2400 x 2400	2285.12
32	3200 x 3200	3646.03
64	6400 x 6400	9919.36
100	10000 x 10000	23163.69



Reevaluation of Amdahl's Law

- Amdahl’s Law overlooks the fact that for many algorithms, the percentage of required sequential operations decreases as the size of the *problem* increases and hence the speed up achieved will also increase with the size of the problem.

- However, this is not applicable for all problems. For problems such as Matrix multiplication, as the size of problem increases larger volume have data have to be communicated among processors and will affect the speed up factor after a point.

- To solve Matrix multiplication more effectively we need to come up with a better algorithm as well a better communication system among processors.

Challenges Faced

- The Matrix dimensions should be completely divisible by Number of Processors.
- Doubling the Matrix dimension essentially means quadrupling the size of data.
- Thus No of Processors should also be quadrupled to achieve desired results.
- Matrix multiplication is a heavily communication dependant Problem and thus with increase in No of Processors/ data heavily affects the running Time.

Questions ??

Thank you!