University at Buffalo
Department of Computer Science and Engineering
School of Engineering and Applied Sciences

PARALLEL MATRIX MULTIPLICATION

Prepared by:

Malvika Sundaram Srinivasan

50290572

Guided by:

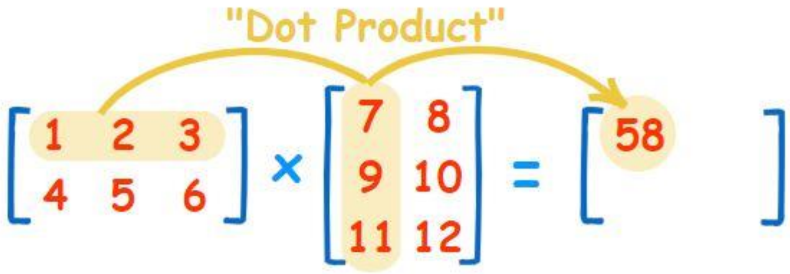
Professor Dr. Russ Miller

University at Buffalo
Department of Computer Science and Engineering
School of Engineering and Applied Sciences

TABLE OF CONTENTS

MATRIX MULTIPLICATION

- Given two matrices Matrix A of size **mxn** with elements a_{ij} and Matrix B of size **nxp** with elements b_{jk})
- Matrix C is the product of A and B with size mxp



$$c_{ij} = a_{i1}b_{1j} + \cdots + a_{in}b_{nj} = \sum_{k=1}^{n} a_{ik}b_{kj},$$

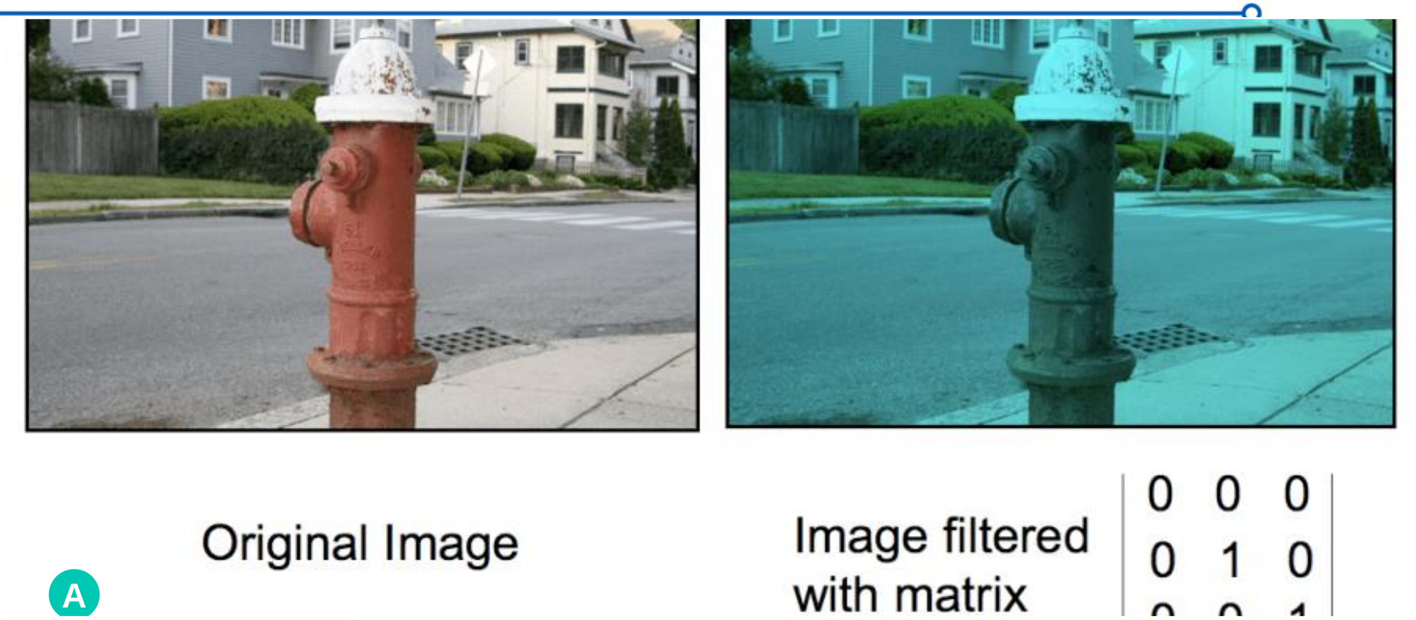
for $i = 1, \ldots, m$ and $j = 1, \ldots, p$.

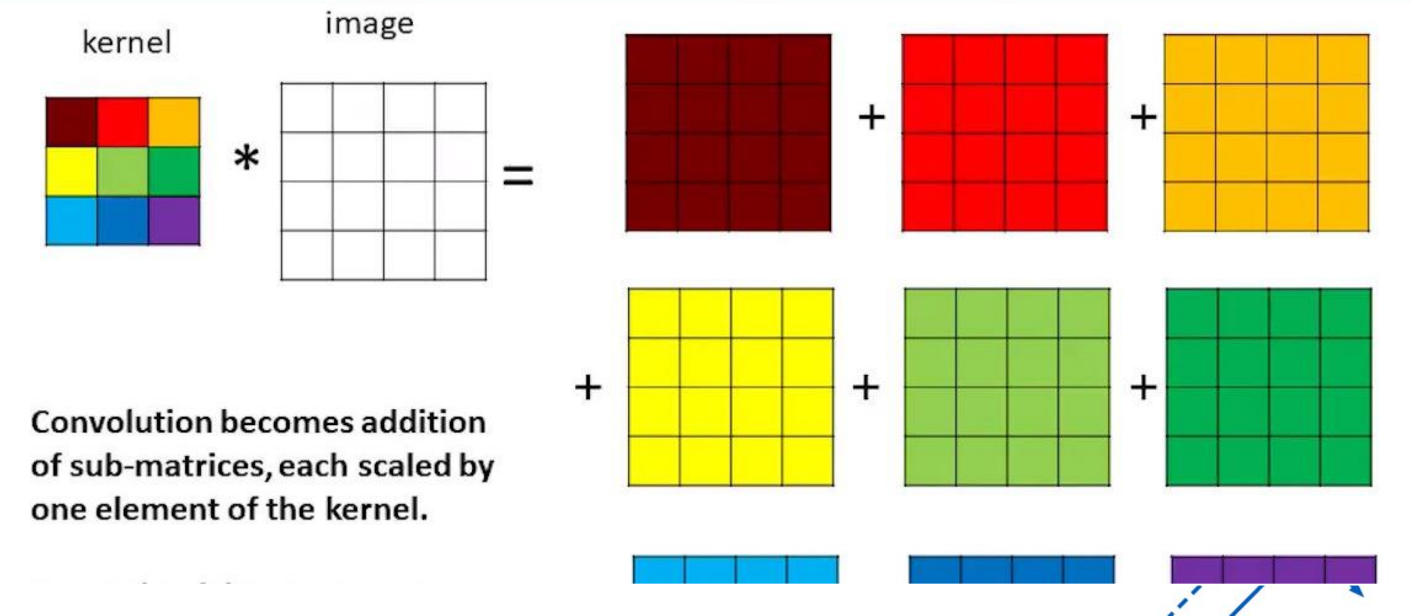
Number of Columns of A = Number of Rows of B

USE CASE

These are the final goals of the project

- *Perform some image filters*
- *Perform convolution using General Matrix Multiplication(GEMM) in parallel*





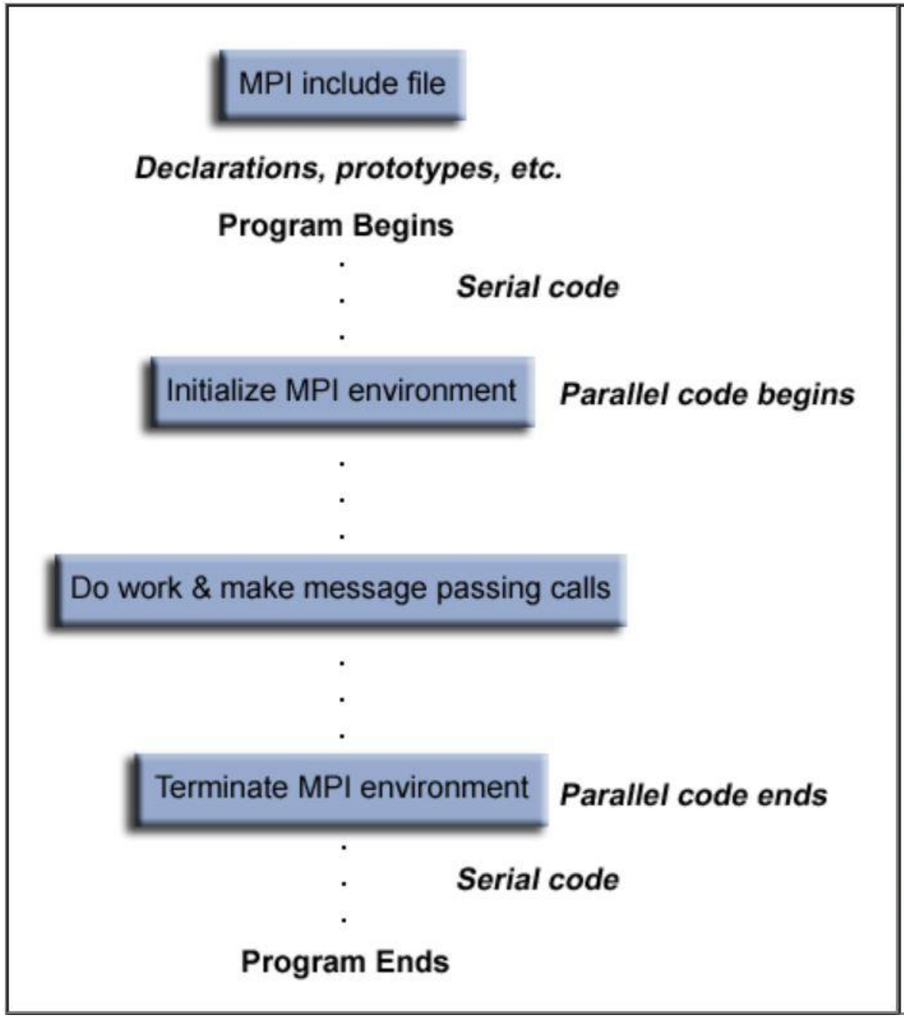
PROCESS

Using ***Open MPI*** to write matrix multiplication

The steps taken to run a program

1. *Write the configurations and module loading as a shell script (SLURM)*
2. *The shell script also contains program to run*
3. *Run the script with sbatch command*
4. *Monitor the status using squeue or the jobs dashboard*
5. *Run the test for 3 times in each configuration and compute the average*

MPI Program Structure




```c
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[])
{
int numtasks, rank, dest, source, rc, count, tag=1;
char inmsg, outmsg='x';
MPI_Status Stat;

MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);

if (rank == 0) {
  dest = 1;
  source = 1;
  rc = MPI_Send(&outmsg, 1, MPI_CHAR, dest, tag, MPI_COMM_WORLD);
  rc = MPI_Recv(&inmsg, 1, MPI_CHAR, source, tag, MPI_COMM_WORLD, &Stat);
  }

else if (rank == 1) {
  dest = 0;
  source = 0;
  rc = MPI_Recv(&inmsg, 1, MPI_CHAR, source, tag, MPI_COMM_WORLD, &Stat);
  rc = MPI_Send(&outmsg, 1, MPI_CHAR, dest, tag, MPI_COMM_WORLD);
  }

MPI_Finalize();
}
```

SEQUENTIAL APPROACH

ITERATIVE ALGORITHM

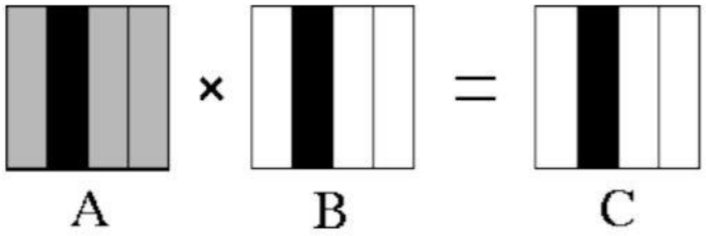
Complexity:

- The algorithm takes $\Theta(nmp)$ time.
- If input are square matrices of size **nxn**, the runtime is cubic i.e. $\Theta(n^3)$

- Input: matrices A and B
- Let C be a new matrix of the appropriate size
- For i from 1 to n:
    - For j from 1 to p:
        - Let $\text{sum} = 0$
        - For k from 1 to m:
            - Set $\text{sum} \leftarrow \text{sum} + A_{ik} \times B_{kj}$
        - Set $C_{ij} \leftarrow \text{sum}$
- Return C

PARALLEL APPROACH – 1D Decomposition

- 1-D column wise decomposition
- Each task:
  - Utilizes subset of cols of A, B, C.
  - Responsible for calculating its C_{ij}
  - Requires full copy of A
  - Requires $\frac{N^2}{P}$ data from each of the other $(P-1)$ tasks.
- # Computations: $\mathcal{O}\left(N^3/P\right)$
- $T_{mat-mat-1D} = (P-1)\left(t_{st} + t_{wall}\frac{N^2}{P}\right)$



PARALLEL APPROACH – Cannon's Algorithm

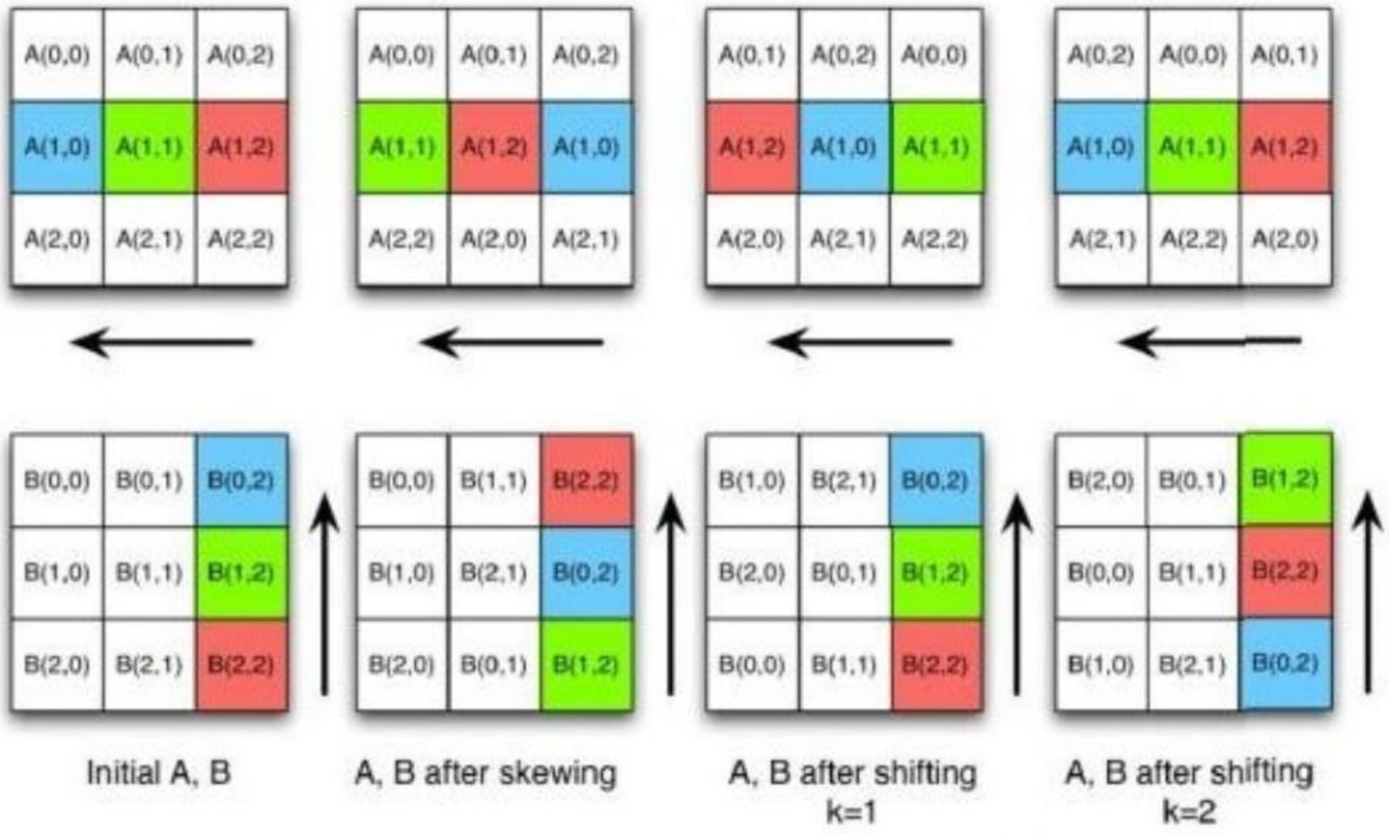
- It is especially suitable for computers laid out in an N × N mesh.
- **Storage requirements remain constant** and are **independent of the number of processors**

Algorithm overview

When multiplying two $N \times N$ matrices A and B, we need $N \times N$ processing nodes P arranged in a 2d grid. Initially $p_{i,j}$ is responsible for $a_{i,j}$ and $b_{i,j}$.

```
row i of matrix a is circularly shifted by i elements to the left.
col j of matrix b is circularly shifted by j elements up.
Repeat n times:
    p[i][j] multiplies its two entries and adds to running total.
    circular shift each row of a 1 element left
    circular shift each col of b 1 element up
```

PARALLEL APPROACH – Cannon's Algorithm



C(1,2) = A(1,0) * B(0,2) + A(1,1) * B(1,2) + A(1,2) * B (2,2)

PARALLEL APPROACH – Cannon's Algorithm

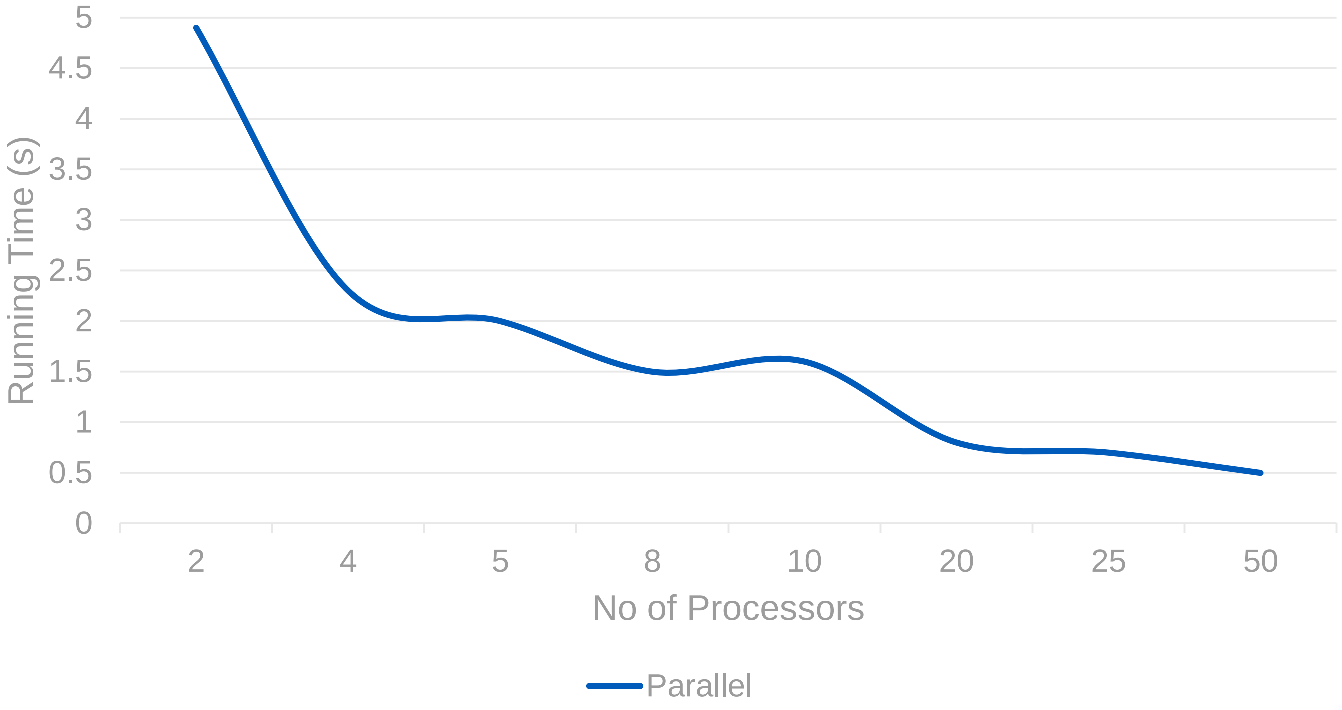
- *Consider two* $n \times n$ *matrices* $A(i, j)$ *and* $B(i, j)$ *partitioned into* p *blocks.*
- $0 \le i, j \le \sqrt{p}$ *and the size* $(n / \sqrt{p}) \times (n / \sqrt{p})$ *each.*
- *Process* $P(i, j)$ *initially stores* $A(i, j)$ *and* $B(i, j)$ *, computes block* $C(i, j)$ *of the result matrix.*
- *The initial step of the algorithm regards the alignment of the matrices*
    - *Align the blocks of A and B in such a way that each process can independently start multiplying its local submatrices.*
    - *This is done by shifting all submatrices* $A(i, j)$ *to the left (with wraparound) by i steps and all submatrices* $B(i, j)$ *up (with wraparound) by j steps.*
    - *Perform local block multiplication.*
    - *Each block of A moves one step left and each block of B moves one step up (again with wraparound)*
- *Perform next block multiplication, add to partial result, repeat until all blocks have been multiplied.*

RESULTS - SEQUENTIAL

No of Processors	Matrix Size	Runtime (s)
1	100 x 100	3.39
1	1000 x 1000	11.21
1	2000 x 2000	83.24
1	3000 x 3000	372.78
1	4000 x 4000	854.39
1	5000 x 5000	2003.24

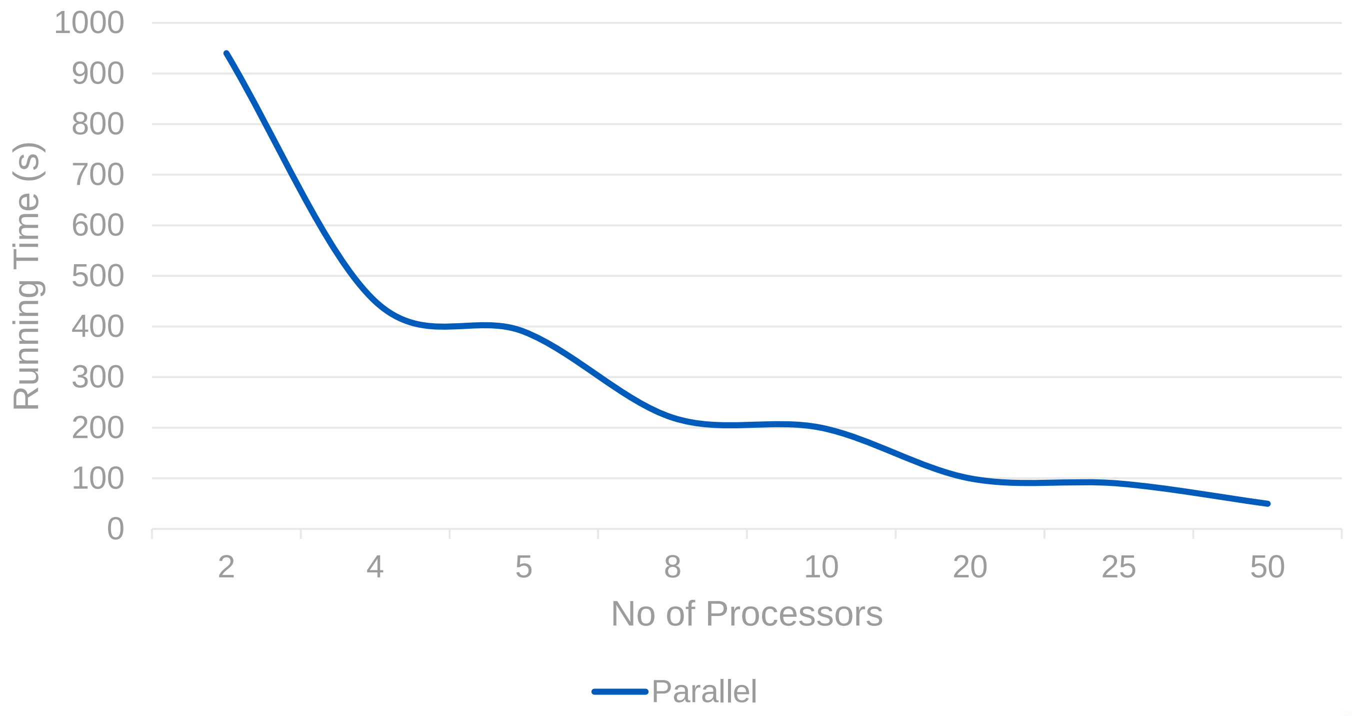
RESULTS - PARALLEL

Matrix size: 1000 x 1000



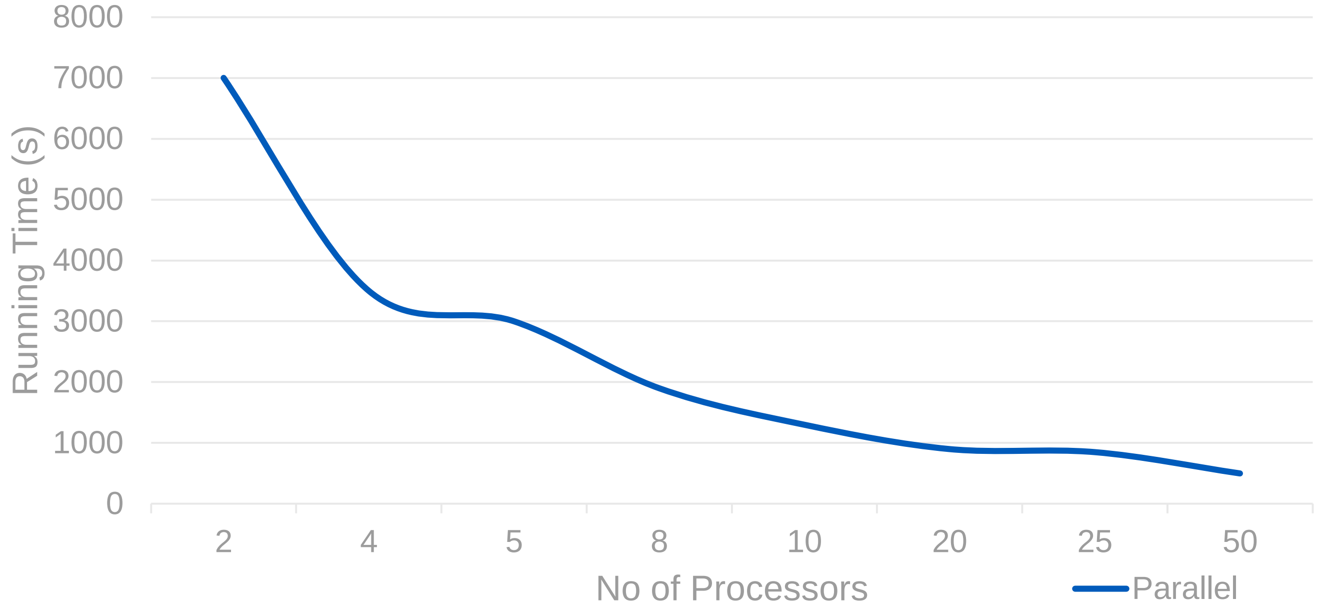
RESULTS - PARALLEL

Matrix size: 5000 x 5000



RESULTS - PARALLEL

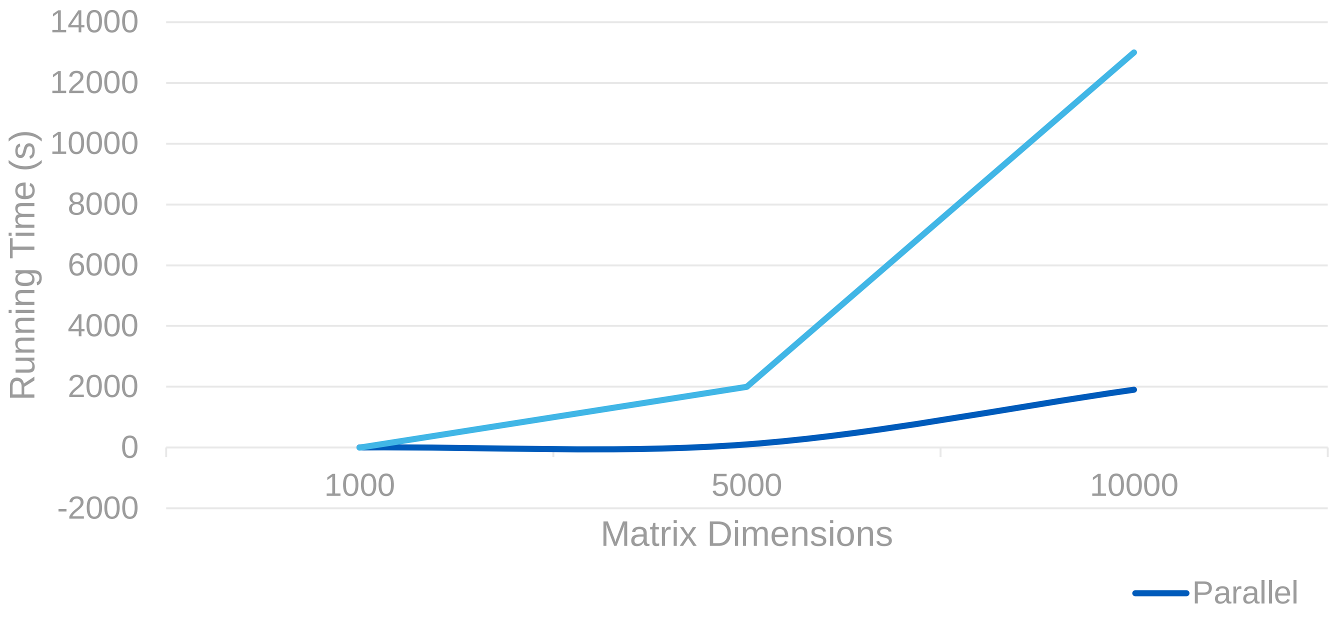
Matrix size: 10000 x 10000



University at Buffalo
Department of Computer Science and Engineering
School of Engineering and Applied Sciences

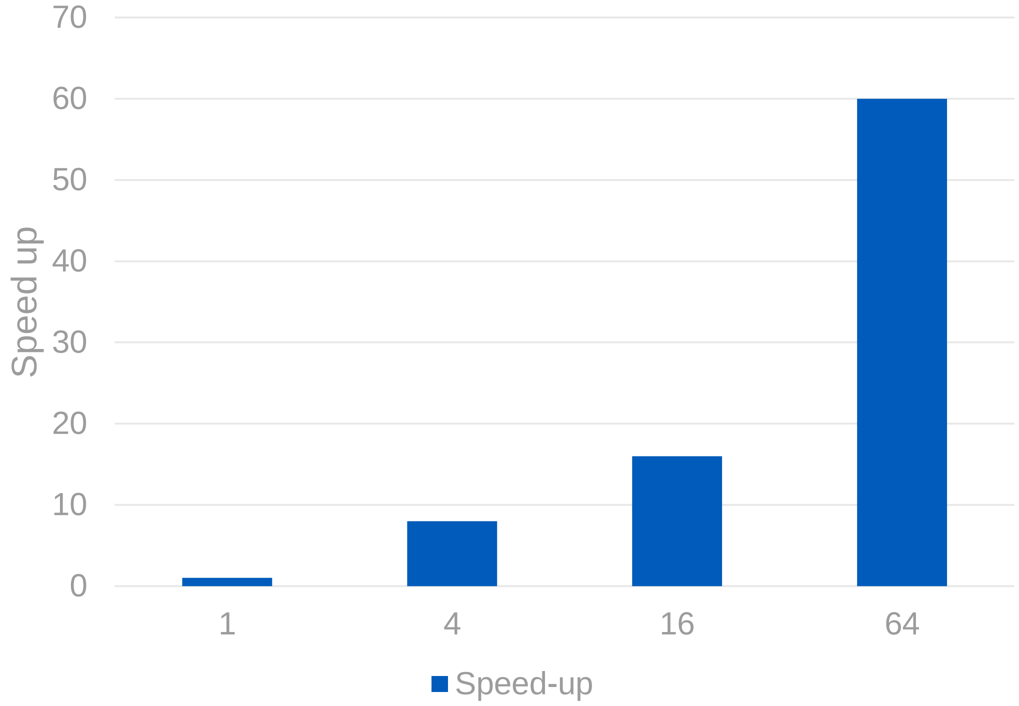
RESULTS – PARALLEL VS SEQUENTIAL

No of Processors in parallel = 10



RESULTS – SPEEDUP

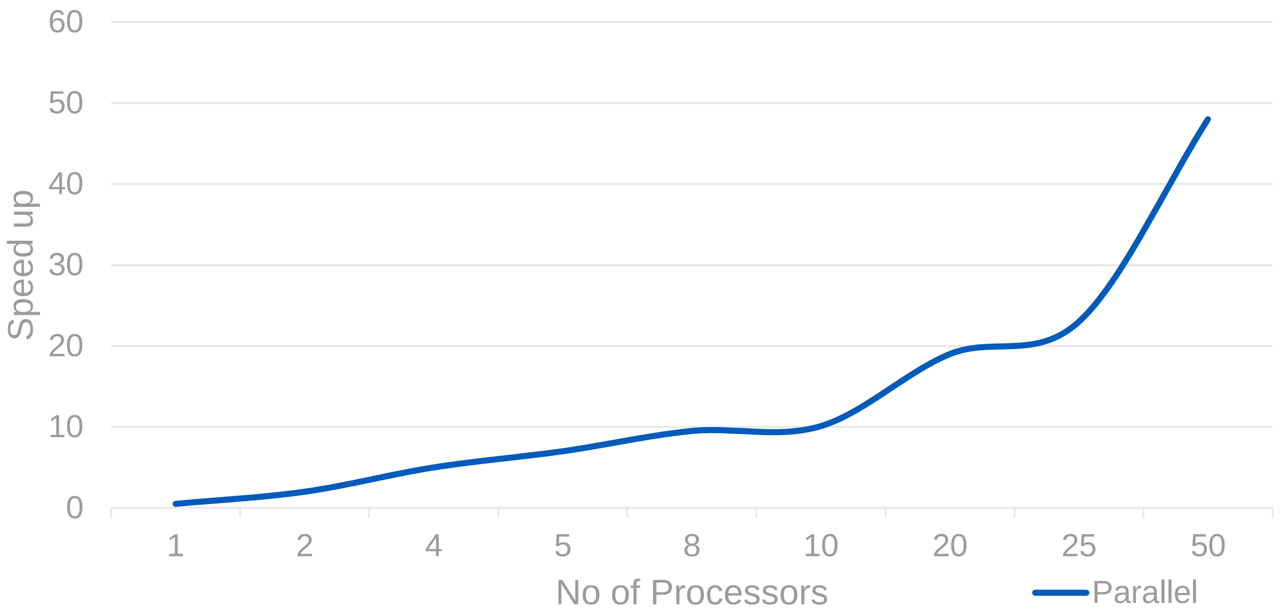
Matrix size: 100 x 100



No of Processors	Speedup
1	1
4	6.46
16	16.53
64	59.56

RESULTS – SPEEDUP

Matrix size: 5000 x 5000



RESULTS

1D Decomposition vs Cannon’s algorithm

No of Processors	Matrix Size	Runtime (s)
1	100 x 100	3.39
4	200 x 200	1.62
1	200 x 200	8.21
4	1000 x 1000	2.341

No of Processors	Matrix Size	Runtime (s)
1	100 x 100	2.89
4	200 x 200	1.13
1	200 x 200	7.8142
4	1000 x 1000	2.1896

LEARNING

- Understanding of Parallelization and writing MPI & SLURM script
- Increasing the processors doesn’t always reduce the running time
- At each stage doubling the data means quadrupling the number of processors
- Running times depend on how the nodes get allocated on CCR cluster

FUTURE WORK

- Try to implement in OpenMP and compare the results with Apache Spark

REFERENCES

- **Parallel Multi Channel Convolution using General Matrix Multiplication :** https://arxiv.org/pdf/1704.04428.pdf
- **2D Image Convolution using Three Parallel Programming Models on the Xeon Phi :** https://arxiv.org/pdf/1711.09791.pdf
- https://makezine.com/2011/03/30/codebox-create-image-filters-with-matrix-multiplication/
- https://en.wikipedia.org/wiki/Matrix_multiplication_algorithm
- https://computing.llnl.gov/tutorials/mpi/#Abstract
- https://edoras.sdsu.edu/~mthomas/sp17.605/lectures/MPI-MatMatMult.pdf