University at Buffalo The State University of New York

MATRIX MULTIPLICATION USING MPI

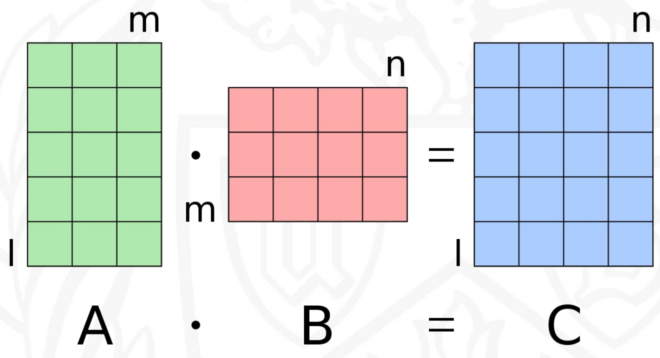
Adithya Raman

Final Review (12/02/2021)

University at Buffalo The State University of New York

Matrix Multiplication

- A matrix is linear transformation
- Applications in
  - Graphics:
    - Scaling, Translations and Rotations of vectors
  - Can represent a system of linear equations
- In general if A is (l x m) and B is (m x n) then the product is an (l x n) matrix whose elements are :
  - $C_{l*n} =$

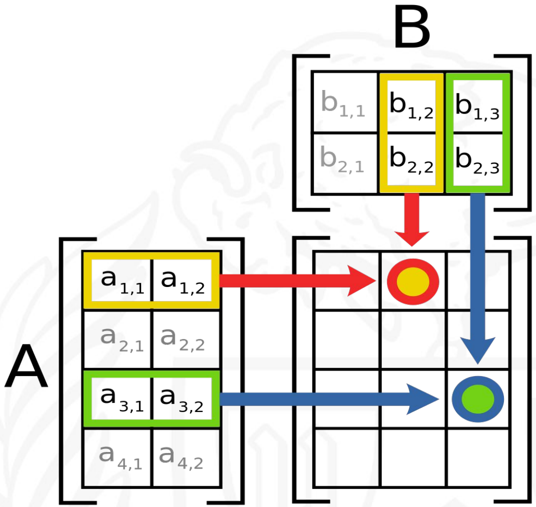


Straight forward single processor serial multiplication

Algorithm 1: The Naive Matrix Multiplication Algorithm

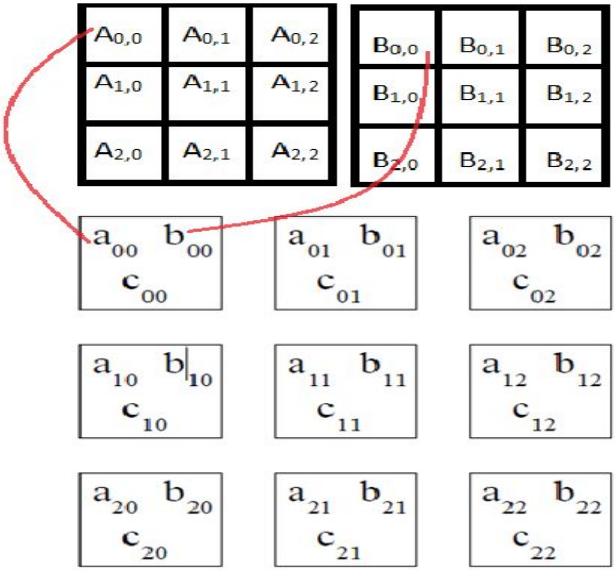
```
Data: S[A][B], P[G][H]
Result: Q[][]
if B == G then
    for m = 0; m < A; m++ do
        for r = 0; r < H; r++ do
            Q[m][r] = 0;
            for k = 0; k < G; k++ do
                Q[m][r] + = S[m][k] * P[k][r];
            end
        end
    end
end
```

Source:https://www.baeldung.com/cs/matrix-multiplication-algorithms



- Multiplying a matrix of size (AxB) with a matrix of size (BxC) using the naive approach gives a complexity of
    - *O(A*B*C)*

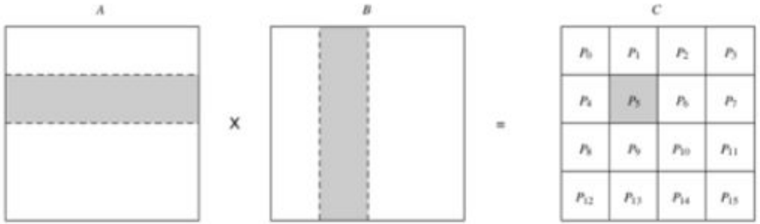
Cannon's Algorithm (working with square matrices)



- Let $A = [a_{ij}]_{nxn}$ and $B = [b_{ij}]_{nxn}$ be two matrices
- To compute C=AB using *'p'* processors:
  - Partition A and B into p square blocks $A_{i,j}$ and $B_{i,j}$ such that ($0 <= i,j <= p^{1/2}$)
  - Size of each block will be $(n/p^{1/2})x(n/p^{1/2})$
  - Initialize C sub-blocks at each processor with size $(n/p^{1/2})x(n/p^{1/2})$ and values as 0

- ***Algorithm:***
  1) At each processor compute the partial sum of the C sub-block in that processor using current A sub-block and B-sub-block
  2) Shift A sub-block one step to the left
  3) Shift B sub-block one step up

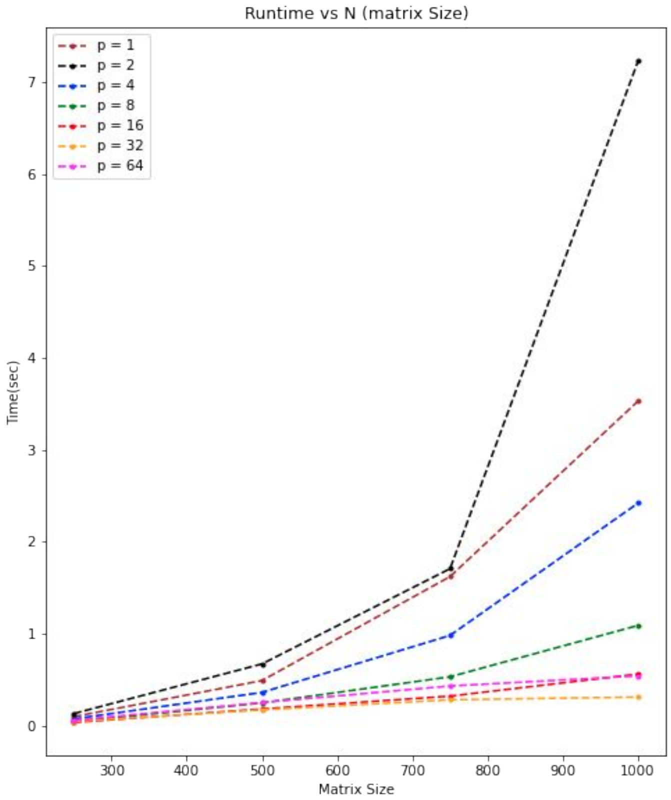
es

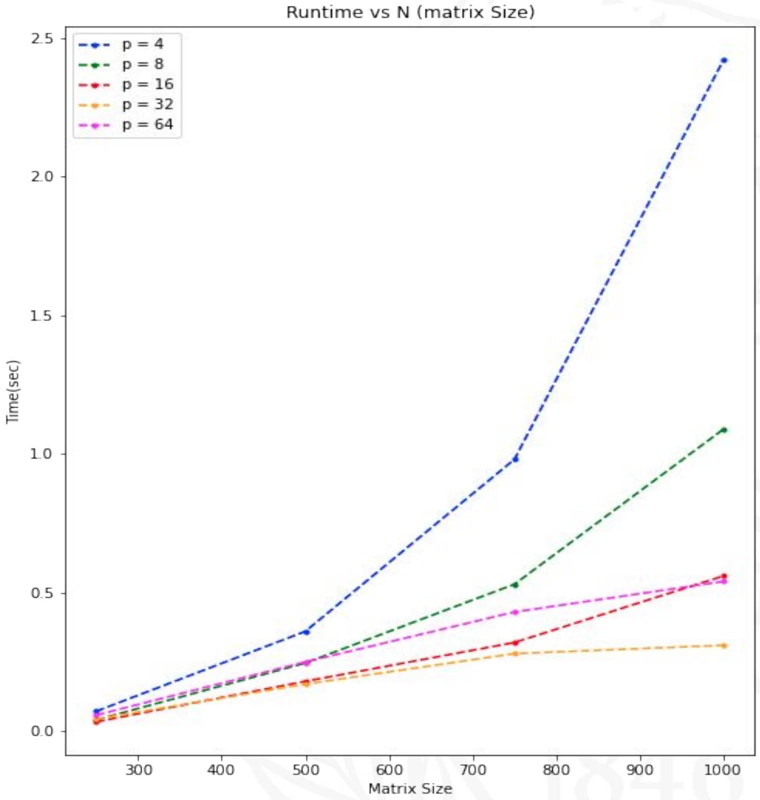


Parallel Algorithm Performance

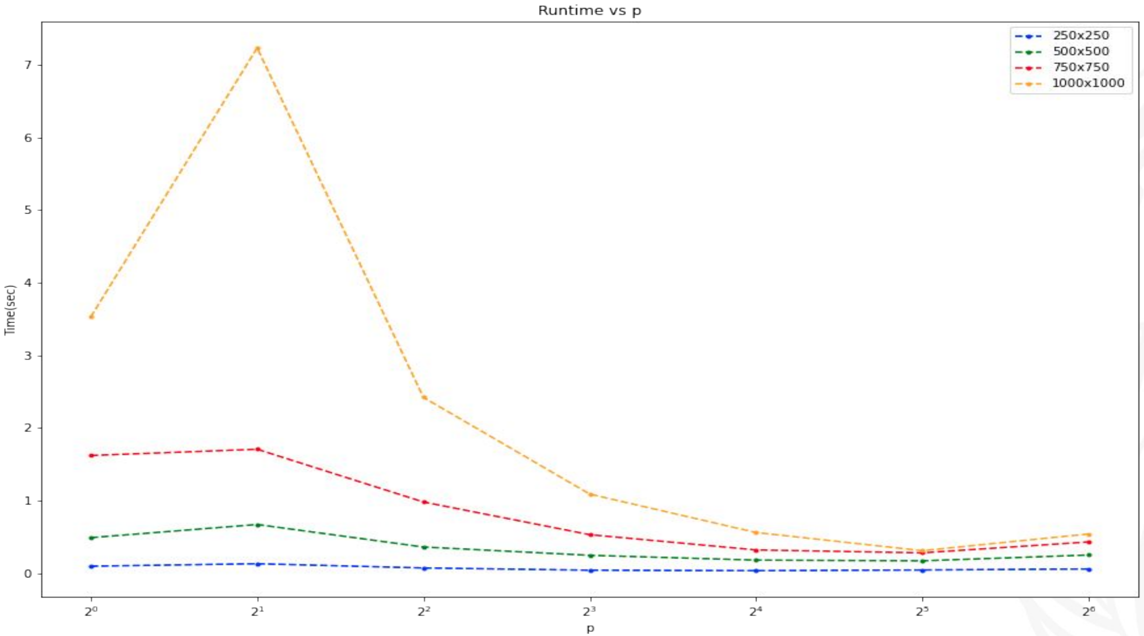
nodes	T(250x250)	T (500x500)	T(750x750)	T (1000x1000)	T (10000x1000 0)
1	0.096	0.489	1.62	3.53	~
2	0.130	0.67	1.706	7.23	~
4	0.072	0.36	0.98	2.42	~
8	0.041	0.246	0.53	1.089	~
16	0.034	0.18	0.32	0.56	580.794
32	0.044	0.17	0.28	0.31	367.178
64	0.058	0.25	0.43	0.54	161.535

Parallel Algorithm Performance

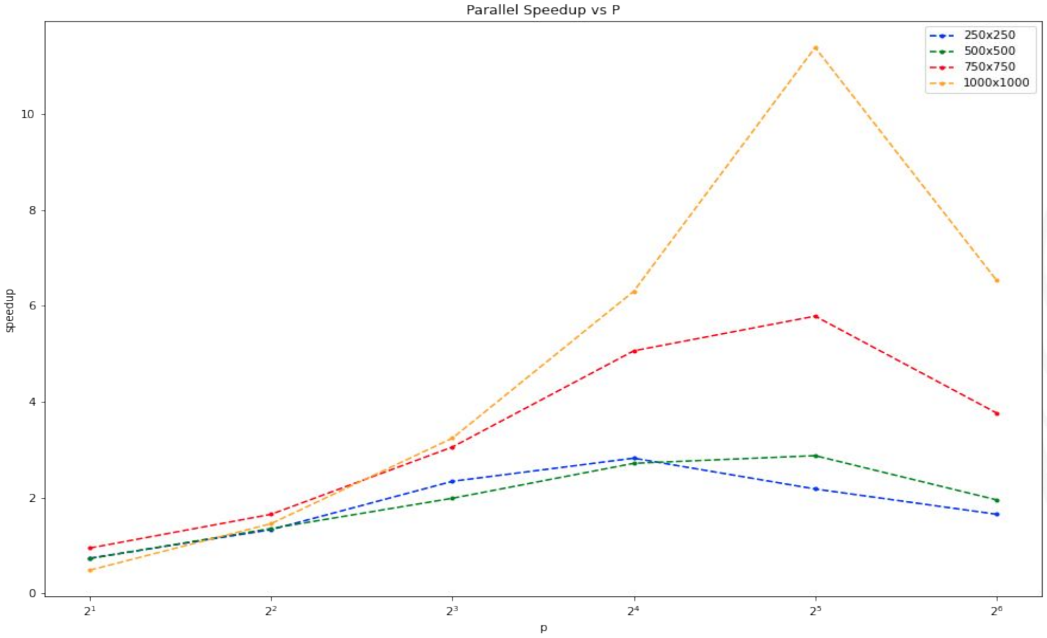




Parallel Algorithm Performance



Speedup



Note :

- We use the runtime of the best performing serial algorithm to calculate speedup
- The serial algorithm I used is NOT the most efficient.

Concluding Remarks

- Efficiency of the parallel algorithm decreases with increasing number of nodes
- Beyond 32 nodes the run-time of parallel algorithm increases
- Speedup is much higher for larger problem sizes
  - Comparing the 1000x1000 vs the 250x250 execution