

CS410 Assignment 3

Josyula V N Taraka Abhishek - 200010021

April 2023

Contents

1	SUMMA topology and communication	2
2	Algorithm	3
3	Pseudocode:	4
4	Coorectness of program	5
5	Experiments	6
5.1	SpeedUp	6
5.2	Parallel Efficiency	7

List of Figures

1	Topology and comm	2
2	SUMMA algorithm. Souce: slides	3
3	Summa check	5
4	Summa check results	5
5	SpeedUp	6
6	Parallel Efficiency	7

1 SUMMA topology and communication

- In SUMMA algorithm we split the matrix into tiles of size $\frac{n}{\sqrt{p}} \times \frac{n}{\sqrt{p}}$ where p is the number of available processors and n is the size of matrix.
- Each processor updates its own block.
- Processors are mapped to blocks using Cartesian topology.
- We use the MPI function *MPI_Cart_create(...)* to create the Cartesian topology.
- Each processor then calls the *SUMMA(...)* algorithm and updates its own block.
- We create a communication domain for each row and each column of the 2D - cartesian mesh.
- We use the *MPI_Cart_sub(...)* function to create the communication domains.
- The topology, rank and communication domains can be understood from the following figure.

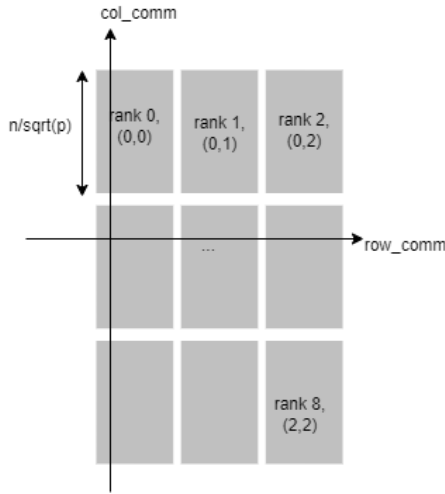


Figure 1: Topology and comm

- If processor's column coordinate equals to K th iteration of loop, it will broadcast its local portion of A within its 'row_comm' communicator.
- If processor's row coordinate equals to K th iteration of loop, it will broadcast its local portion of B within its 'col_comm' communicator.

- To find the span time we reduce the time taken by all processors and reduce it using MPI_MAX function.

2 Algorithm

- After broadcasting, we do naive multiplication with portions which each processor have received from others and store it in partial sum 'C_loc_tmp'.
- Finally we accumulate partials sums of 'C_loc_tmp' to 'C_loc' on each iteration
- This method minimises communication because at one processor of each row communicates within its row domain using binary tree, and vice versa for column communication.

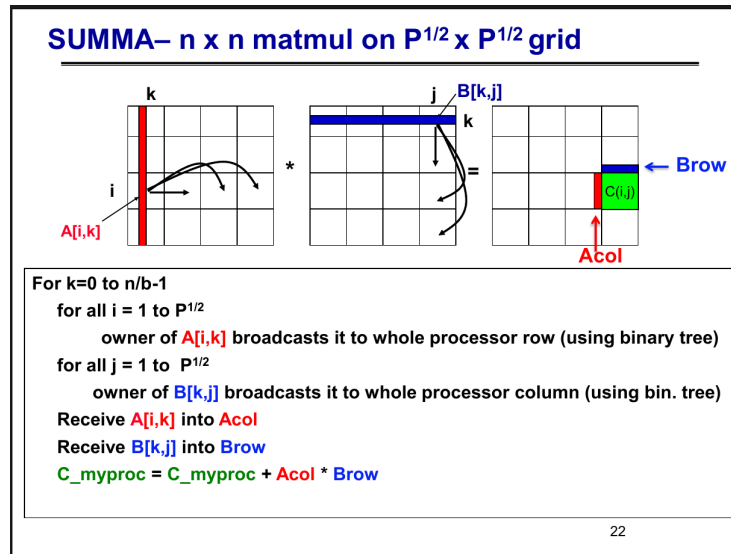


Figure 2: SUMMA algorithm. Souce: slides

3 Pseudocode:

SUMMA

```
for (int bcast_root = 0; bcast_root < nblks; ++bcast_root) {  
    // owner of A_loc[root_col,:] will broadcast its block within row comm  
    if (my_col == root_col) {  
        memcpy(A_loc, A_loc_save );  
    }  
    MPI_Bcast(A_loc, ....);  
  
    // owner of B_loc[:,root_row] will broadcast its block within col comm  
    if (my_row == root_row) {  
        memcpy(B_loc, B_loc_save);  
    }  
    MPI_Bcast(B_loc, ....);  
  
    // multiply local blocks A_loc, B_loc using matmul_naive  
    // and store in C_loc_tmp  
    matmul_naive(mb, nb, kb, A_loc, B_loc, C_loc_tmp);  
  
    // C_loc = C_loc + C_loc_tmp using plus_matrix  
}
```

SPAN time

```
MPI_Reduce(&etime, &max_etime, 1, MPLDOUBLE, MPLMAX, 0, MPLCOMM_WORLD);  
if (myrank == 0)  
{  
    printf("SUMMA took %f sec\n", max_etime);  
}
```

4 Coorrectness of program

SUMMA_check is used to verify if the mpi version is correct. It gathers all the A_loc, B_loc, C_loc versions and performs naive matmul on them.

It then prints the error.

```
[c200010021@iitdhmaster cs410assignment3-DefenseIsAlie-1]$ cat submit.sbatch
#!/bin/bash
#SBATCH --job-name=MPI_JOB           # Job name
#SBATCH --mail-type=END,FAIL         # Mail events (NONE, BEGIN, END, FAIL, ALL)
#SBATCH --mail-user=200010021@iitdh.ac.in # Where to send mail
#SBATCH --nodes=2                   # Run on 2 nodes
#SBATCH --ntasks-per-node=32        # 32 per node
#SBATCH --cpus-per-task=1           # 1 cpu per task
#SBATCH --time=00:5:00              # Time limit hrs:min:sec
#SBATCH --output=test_%j.log        # Standard output and error log
#SBATCH --partition=test            # Queue name
#SBATCH -n 64                       # Must be a perfect square

pwd; hostname; date
export I_MPI_PMI_LIBRARY=/usr/lib64/libpmi.so
srun ./summa_check 3360
date

[c200010021@iitdhmaster cs410assignment3-DefenseIsAlie-1]$ sbatch submit.sbatch
Submitted batch job 9561
```

Figure 3: Summa check

```
Makefile M submitsbatch M test_9561.log X
test_9561.log
1 /iitdh/Courses/cs410spring23/c200010021/cs410assignment3-DefenseIsAlie-1
2 cn01.iitdh.ac.in
3 Sat Apr 8 16:07:03 IST 2023
4 m, n, k = 3360, 3360, 3360
5 SUMMA took 0.845235 sec
6 Validating ...
7 SUMMA: OK: eps = 0.000000
8 Sat Apr 8 16:07:32 IST 2023
9
```

Figure 4: Summa check results

5 Experiments

Results of experiment.

NPROC	TIME	SpeedUp	Parallel Efficiency
1	215.78154	1	1
4	55.600979	3.880894615	0.9702236538
9	26.83926	8.039772334	0.8933080371
16	18.937714	11.39427599	0.7121422496
25	13.040093	16.54754609	0.6619018438
36	9.388849	22.98274687	0.6384096354
49	8.487743	25.4227231	0.5188310836
64	7.758794	27.81122169	0.4345503389

5.1 SpeedUp

SpeedUp vs NPROC

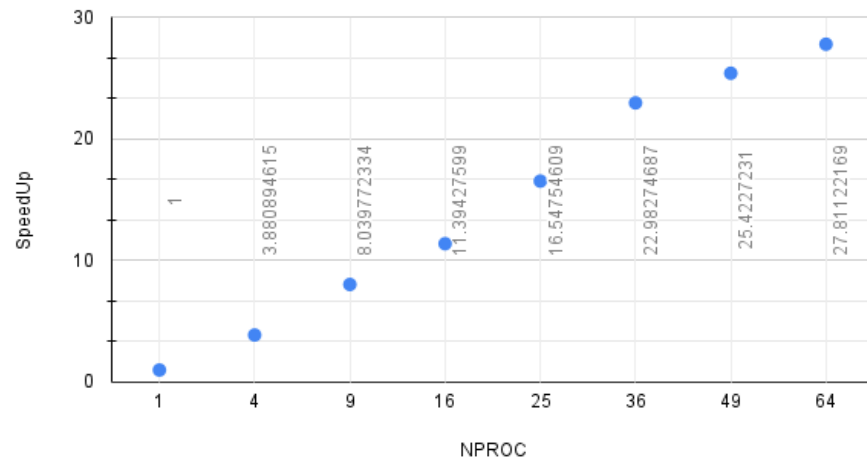


Figure 5: SpeedUp

5.2 Parallel Efficiency

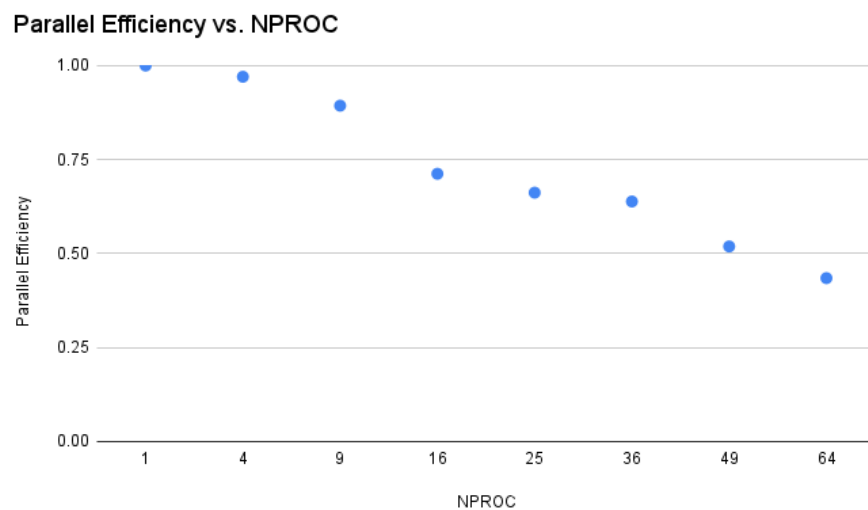


Figure 6: Parallel Efficiency

- I achieved a 27x speedup using 64 cores and the parallel efficiency is around 50
- This indicates that the program scales well.
- The outer k loop runs serially. This means that the span is of order n
- The span is characterized by $T_{\infty} = O(n)$
- Hence parallelsim achieved is $O(n^2)$.