Matrix Multiplication Using MPI

Mehedi Hasan Shakil
2010876138

Problem Statement

Write a program to multiply ***K*** different matrices ***A*** of dimension ***MxN*** with matrices ***B*** of dimension ***NxP*** dimension matrices. Where ***K*** is the number of matrices.

Where,
$K * M * N <= 10^6$; $K * N * P <= 10^6$; $K * M * P <= 10^6$

Code: Github

Message Passing Interface (MPI)

- library of routines to create parallel processes and exchange information among these processes
- uses operating system services to create parallel processes exchange information
- supports distributed program execution on heterogeneous hardware

Initialize the MPI Execution Environment

```
int MPI_Init(int *argc, char ***argv)
```

- `MPI_Init(&argc, &argv);`

argc

- Pointer to the number of arguments

argv

- Pointer to the argument vector

Initialize the MPI Execution Environment

```
int MPI_Init(int *argc, char ***argv)
```

- `MPI_Init(&argc, &argv);`

```
mpirun -np 3 ./matrix_multiplication
```

Initialize the MPI Execution Environment

```
int MPI_Init(int *argc, char ***argv)
```

- `MPI_Init(&argc, &argv);`

```
mpirun -np 3 ./matrix_multiplication
```

rank, size, K, M, N, P, startTime, endTime, A, B, R, localA, localB, localR
Rest of the code

rank, size, K, M, N, P, startTime, endTime, A, B, R, localA, localB, localR
Rest of the code

rank, size, K, M, N, P, startTime, endTime, A, B, R, localA, localB, localR
Rest of the code

Determine the Rank of the Calling Process in the Communicator

```
int MPI_Comm_rank(MPI_Comm comm, int *rank)
```

- `MPI_Comm_rank(MPI_COMM_WORLD, &rank);`

Determine the Rank of the Calling Process in the Communicator

```
int MPI_Comm_rank(MPI_Comm comm, int *rank)
```

- `MPI_Comm_rank(MPI_COMM_WORLD, &rank);`

rank=0, size, K, M, N, P, startTime, endTime, A, B, R, localA, localB, localR
Rest of the code

rank=1, size, K, M, N, P, startTime, endTime, A, B, R, localA, localB, localR
Rest of the code

rank=2, size, K, M, N, P, startTime, endTime, A, B, R, localA, localB, localR
Rest of the code

Determine the Rank of the Calling Process in the Communicator

```
int MPI_Comm_rank(MPI_Comm comm, int *rank)
```

- `MPI_Comm_rank(MPI_COMM_WORLD, &rank);`

```
MPI_Comm_size(MPI_COMM_WORLD, &size);
```

rank=0, size=3, K, M, N, P, startTime, endTime, A, B, R, localA, localB, localR
Rest of the code

rank=1, size=3, K, M, N, P, startTime, endTime, A, B, R, localA, localB, localR
Rest of the code

rank=2, size=3, K, M, N, P, startTime, endTime, A, B, R, localA, localB, localR
Rest of the code

User Input

```
if(rank == 0){
    // User Input
}
```

rank=0, size=3, K=9, M=3, N=3, P=3, startTime, endTime, A, B, R, localA, localB, localR
Rest of the code

User Input

```
if(rank == 0){
    // User Input
}
```

rank=0, size=3, K=9, M=3, N=3, P=3, startTime, endTime, A, B, R, localA, localB, localR
Rest of the code

rank=1, size=3, K, M, N, P, startTime, endTime, A, B, R, localA, localB, localR
Rest of the code

rank=2, size=3, K, M, N, P, startTime, endTime, A, B, R, localA, localB, localR
Rest of the code

Broadcast a Message from the Process with Rank "root"

```
int MPI_Bcast(void *buffer, int count, MPI_Datatype
datatype, int root, MPI_Comm comm)
```

rank=0, size=3, K=9, M=3, N=3, P=3, startTime, endTime, A, B, R, localA, localB, localR
Rest of the code

rank=1, size=3, K, M, N, P, startTime, endTime, A, B, R, localA, localB, localR
Rest of the code

rank=2, size=3, K, M, N, P, startTime, endTime, A, B, R, localA, localB, localR
Rest of the code

Broadcast a Message from the Process with Rank "root"

```
int MPI_Bcast(void *buffer, int count, MPI_Datatype
datatype, int root, MPI_Comm comm)
```

- `MPI_Bcast(&K, 1, MPI_INT, 0, MPI_COMM_WORLD);`
- `MPI_Bcast(&M, 1, MPI_INT, 0, MPI_COMM_WORLD);`
- `MPI_Bcast(&N, 1, MPI_INT, 0, MPI_COMM_WORLD);`
- `MPI_Bcast(&P, 1, MPI_INT, 0, MPI_COMM_WORLD);`

rank=0, size=3, K=9, M=3, N=3, P=3, startTime, endTime, A, B, R, localA, localB, localR
Rest of the code

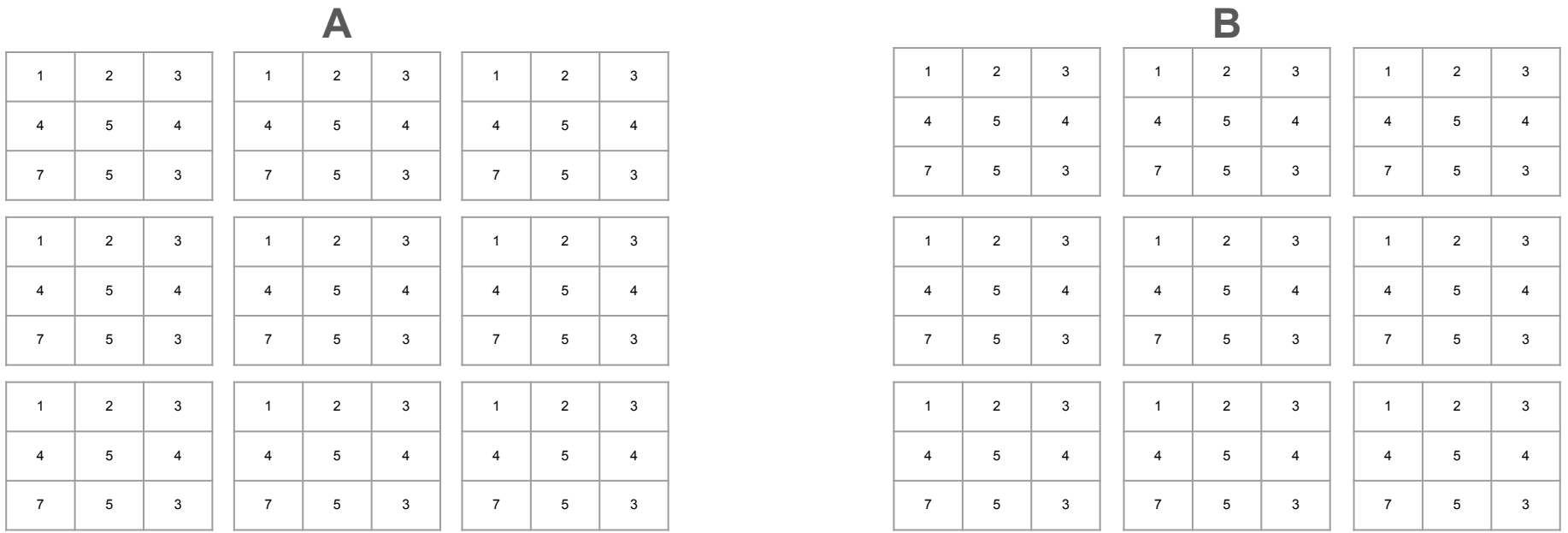
rank=1, size=3, K=9, M=3, N=3, P=3, startTime, endTime, A, B, R, localA, localB, localR
Rest of the code

rank=2, size=3, K=9, M=3, N=3, P=3, startTime, endTime, A, B, R, localA, localB, localR
Rest of the code

Initialize the Matrices in the “root” Process

```
if (rank == 0) {
    // Matrices Initialization
}
```

rank=0, size=3, K=9, M=3, N=3, P=3, startTime, endTime, A, B, R, localA, localB, localR



Send Data from “root” Process to all Processes

```
int MPI_Scatter(const void *sendbuf, int sendcount, MPI_Datatype
sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int
root, MPI_Comm comm)

MPI_Scatter(A, (K / size) * M * N, MPI_INT, localA, (K / size) * M
* N, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Scatter(B, (K / size) * N * P, MPI_INT, localB, (K / size) * N
* P, MPI_INT, 0, MPI_COMM_WORLD);
```

Send Data from “root” Process to all Processes

```
int MPI_Scatter(const void *sendbuf, int sendcount, MPI_Datatype
sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int
root, MPI_Comm comm)

MPI_Scatter(A, (K / size) * M * N, MPI_INT, localA, (K / size) * M
* N, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Scatter(B, (K / size) * N * P, MPI_INT, localB, (K / size) * N
* P, MPI_INT, 0, MPI_COMM_WORLD);
```

rank 0	rank 1	rank 3
localA = $[A_0, A_1, A_2]$ localB = $[B_0, B_1, B_2]$ localR	localA = $[A_3, A_4, A_5]$ localB = $[B_3, B_4, B_5]$ localR	localA = $[A_6, A_7, A_8]$ localB = $[B_6, B_7, B_8]$ localR

Matrix Multiplication by Each Process

```
for (int k = 0; k < (K / size); k++) {
        for (int i = 0; i < M; i++) {
            for (int j = 0; j < P; j++) {
                localR[k][i][j] = 0;
                for (int l = 0; l < N; l++) {
                    localR[k][i][j] += localA[k][i][l] * localB[k][l][j];
                }
            }
        }
    }
```

Matrix Multiplication by Each Process

```
for (int k = 0; k < (K / size); k++) {
        for (int i = 0; i < M; i++) {
            for (int j = 0; j < P; j++) {
                localR[k][i][j] = 0;
                for (int l = 0; l < N; l++) {
                    localR[k][i][j] += localA[k][i][l] * localB[k][l][j];
                }
            }
        }
    }
```

rank 0
localA = [A_0, A_1, A_2]
localB = [B_0, B_1, B_2]
localR = [lA_0*lB_0, lA_1*lB_1, lA_2*lB_2]

rank 1
localA = [A_3, A_4, A_5]
localB = [B_3, B_4, B_5]
localR = [lA_3*lB_3, lA_4*lB_4, lA_5*lB_5]

rank 2
localA = [A_6, A_7, A_8]
localB = [B_6, B_7, B_8]
localR = [lA_6*lB_6, lA_7*lB_7, lA_8*lB_8]

*l: local

Gather Result Matrices from all Processes to the “root” Process

```
int MPI_Gather(const void *sendbuf, int sendcount, MPI_Datatype
sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype,
int root, MPI_Comm comm)

MPI_Gather(localR, (K / size) * M * P, MPI_INT, R, (K / size) *
M * P, MPI_INT, 0, MPI_COMM_WORLD);
```

Gather Result Matrices from all Processes to the “root” Process

```
int MPI_Gather(const void *sendbuf, int sendcount, MPI_Datatype
sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype,
int root, MPI_Comm comm)

MPI_Gather(localR, (K / size) * M * P, MPI_INT, R, (K / size) *
M * P, MPI_INT, 0, MPI_COMM_WORLD);
```

Print Result Matrices from the “root”

```
if(rank == 0) {
    // Print Result Matrices
}
```

30	27	20		30	27	20		30	27	20
52	53	44		52	53	44		52	53	44
48	54	50		48	54	50		48	54	50
30	27	20		30	27	20		30	27	20
52	53	44		52	53	44		52	53	44
48	54	50		48	54	50		48	54	50
30	27	20		30	27	20		30	27	20
52	53	44		52	53	44		52	53	44
48	54	50		48	54	50		48	54	50

Synchronization and Timing

```
int MPI_Barrier(MPI_Comm comm)
```

- Blocks until all processes in the communicator have reached this routine

```
double MPI_Wtime(void)
```

- Returns an elapsed time on the calling processor

Terminate MPI execution environment

```
int MPI_Finalize(void)
```

- All processes must call this routine before exiting.

Thank You