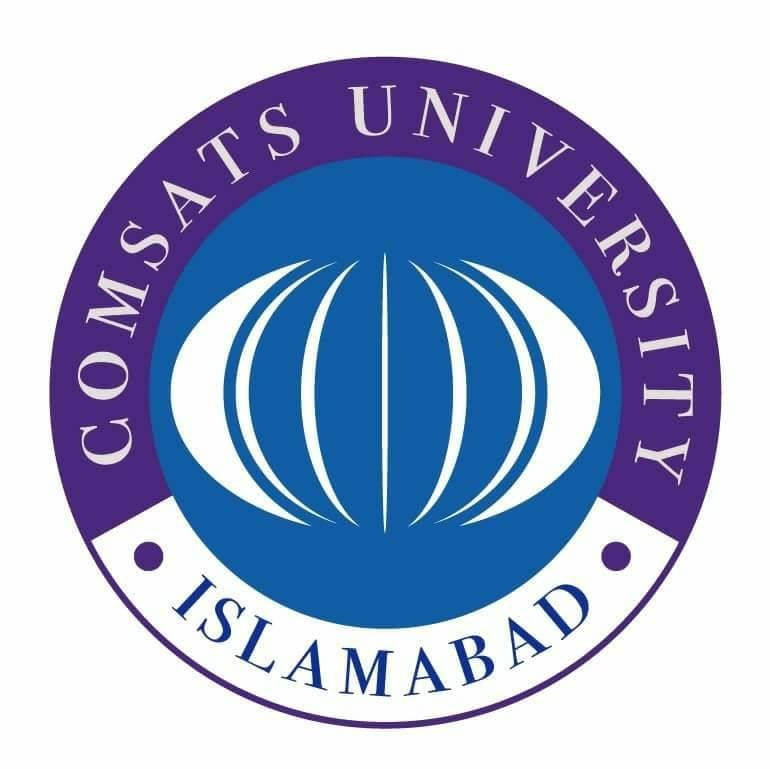
**COMSATS UNIVERSITY ISLAMABAD**

LAHORE CAMPUS



Lab Assignment 05

Submitted by : Shamama Aslam

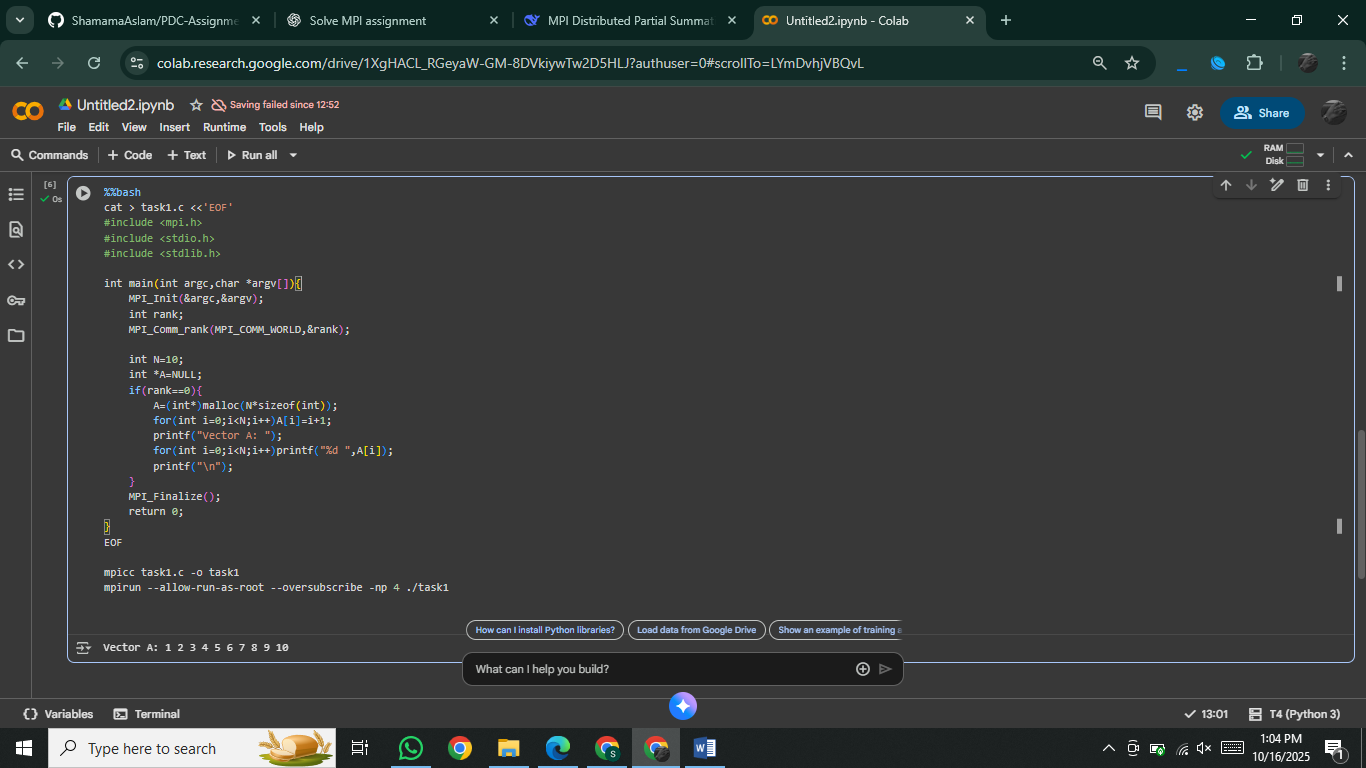
Submitted to : Sir Akhzar Nazir

Reg no : SP23-BCS-143

Section : C

Course Title : Parallel and Distributing Computing

Date : 16 Oct. 2025

**Task 01:**

**Task 02:**  
%%bash

cat > task2.c <<'EOF'

#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

int main(int argc,char \*argv[]){

MPI\_Init(&argc,&argv);

int rank,size;

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD,&size);

int N=12;

int each=N/size;

int \*A=NULL;

if(rank==0){

A=(int\*)malloc(N\*sizeof(int));

for(int i=0;i<N;i++)A[i]=i+1;

}

int \*part=(int\*)malloc(each\*sizeof(int));

MPI\_Scatter(A,each,MPI\_INT,part,each,MPI\_INT,0,MPI\_COMM\_WORLD);

printf("Process %d received:",rank);

for(int i=0;i<each;i++)printf(" %d",part[i]);

printf("\n");

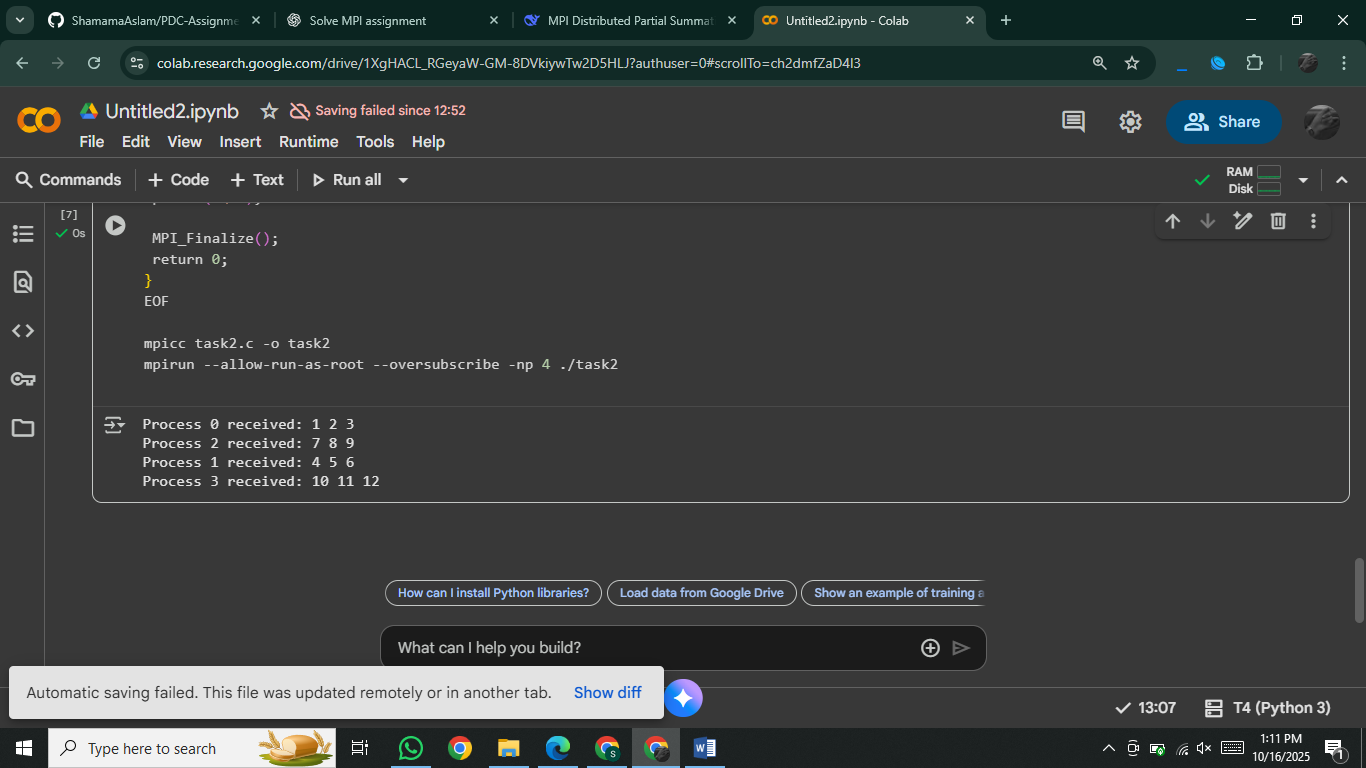
MPI\_Finalize();

return 0;

}

EOF

mpicc task2.c -o task2

mpirun --allow-run-as-root --oversubscribe -np 4 ./task2

**Task 03:**

%%bash

cat > task3.c <<'EOF'

#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

int main(int argc,char \*argv[]){

MPI\_Init(&argc,&argv);

int rank,size;

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD,&size);

int N=12;

int each=N/size;

int \*A=NULL;

if(rank==0){

A=(int\*)malloc(N\*sizeof(int));

for(int i=0;i<N;i++)A[i]=i+1;

}

int \*part=(int\*)malloc(each\*sizeof(int));

MPI\_Scatter(A,each,MPI\_INT,part,each,MPI\_INT,0,MPI\_COMM\_WORLD);

int local\_sum=0;

for(int i=0;i<each;i++)local\_sum+=part[i];

printf("Process %d local sum=%d\n",rank,local\_sum);

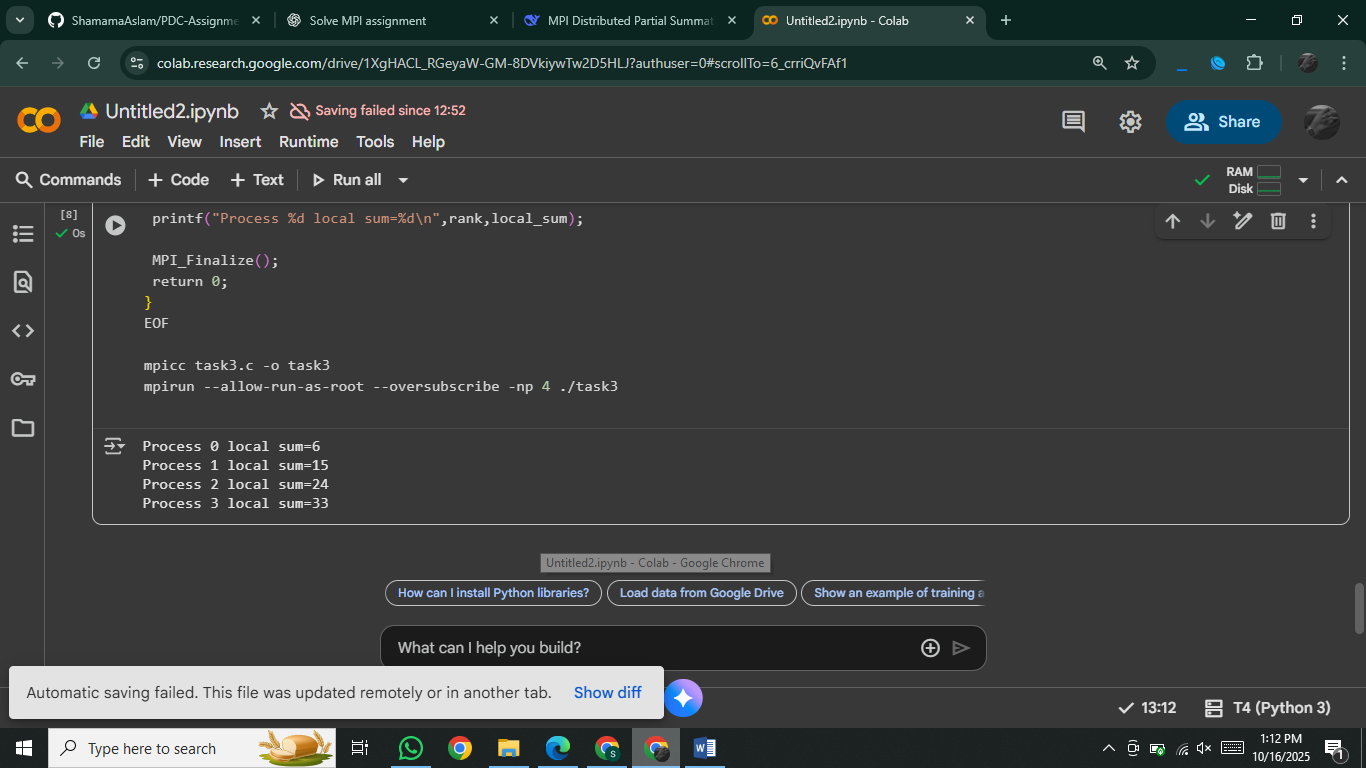
MPI\_Finalize();

return 0;

}

EOF

mpicc task3.c -o task3

mpirun --allow-run-as-root --oversubscribe -np 4 ./task3

**Task 04:**

%%bash

cat > task4.c <<'EOF'

#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

int main(int argc,char \*argv[]){

MPI\_Init(&argc,&argv);

int rank,size;

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD,&size);

int N=12;

int each=N/size;

int \*A=NULL;

if(rank==0){

A=(int\*)malloc(N\*sizeof(int));

for(int i=0;i<N;i++)A[i]=i+1;

}

int \*part=(int\*)malloc(each\*sizeof(int));

MPI\_Scatter(A,each,MPI\_INT,part,each,MPI\_INT,0,MPI\_COMM\_WORLD);

int local\_sum=0;

for(int i=0;i<each;i++)local\_sum+=part[i];

int total\_sum=0;

MPI\_Reduce(&local\_sum,&total\_sum,1,MPI\_INT,MPI\_SUM,0,MPI\_COMM\_WORLD);

if(rank==0)printf("Total Sum (MPI\_Reduce)=%d\n",total\_sum);

MPI\_Finalize();

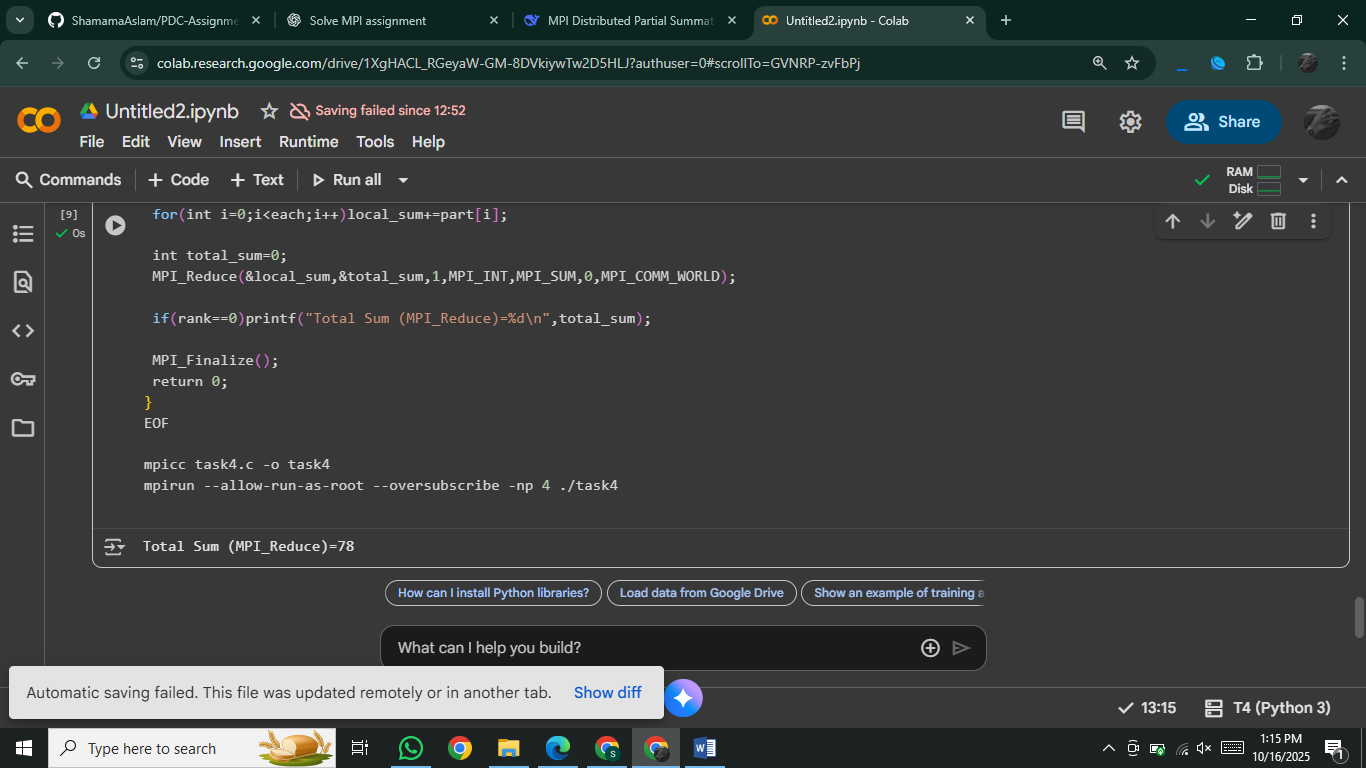
return 0;

}

EOF

mpicc task4.c -o task4

mpirun --allow-run-as-root --oversubscribe -np 4 ./task4



**Task 05:**

%%bash

cat > task5.c <<'EOF'

#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

int main(int argc,char \*argv[]){

MPI\_Init(&argc,&argv);

int rank,size;

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD,&size);

int N=12;

int each=N/size;

int \*A=NULL;

if(rank==0){

A=(int\*)malloc(N\*sizeof(int));

for(int i=0;i<N;i++)A[i]=i+1;

}

int \*part=(int\*)malloc(each\*sizeof(int));

MPI\_Scatter(A,each,MPI\_INT,part,each,MPI\_INT,0,MPI\_COMM\_WORLD);

int local\_sum=0;

for(int i=0;i<each;i++)local\_sum+=part[i];

int total\_sum=0;

MPI\_Reduce(&local\_sum,&total\_sum,1,MPI\_INT,MPI\_SUM,0,MPI\_COMM\_WORLD);

if(rank==0){

int expected=N\*(N+1)/2;

printf("Total Sum (MPI)=%d\nExpected Sum=%d\nDifference=%d\n",

total\_sum,expected,expected-total\_sum);

}

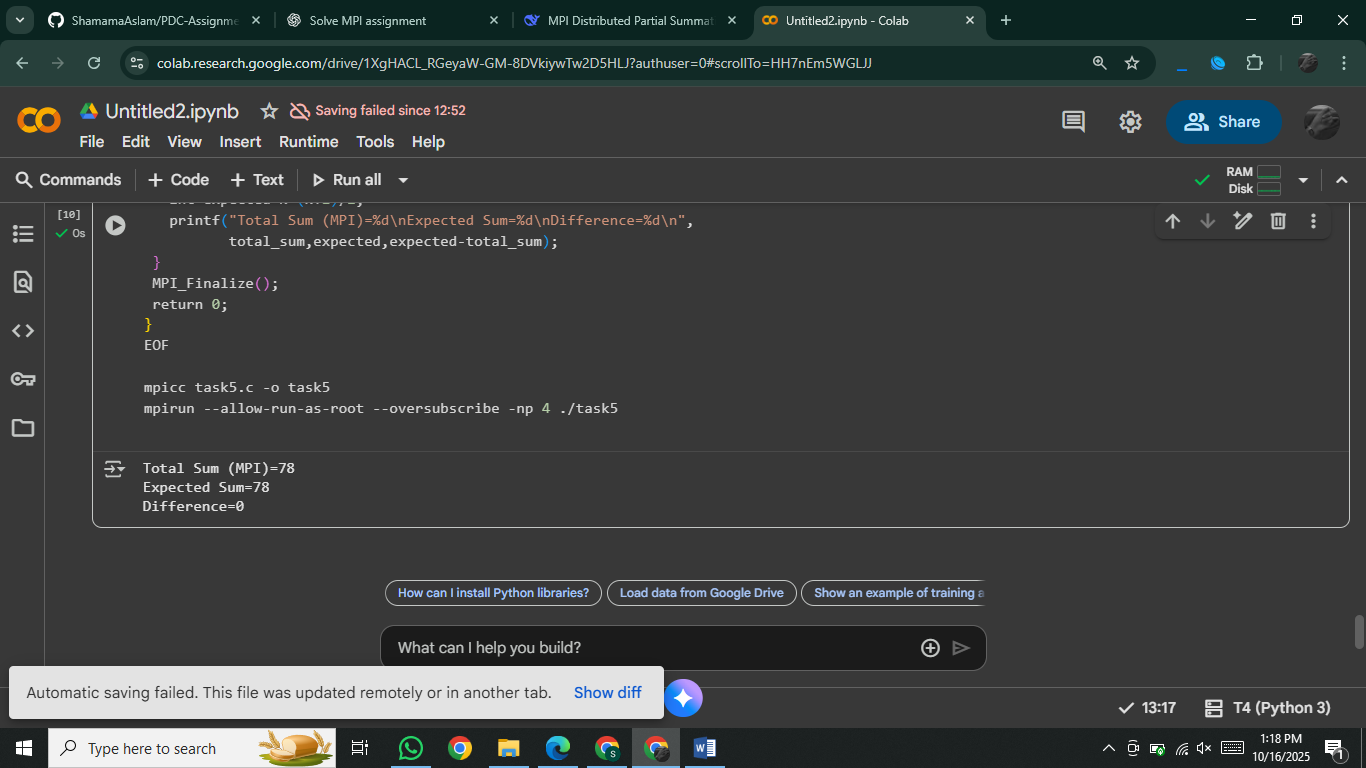
MPI\_Finalize();

return 0;

}

EOF

mpicc task5.c -o task5

mpirun --allow-run-as-root --oversubscribe -np 4 ./task5

**Answers to Discussion Questions**

**1. What happens if the vector size N is not divisible by the number of processes?**

If N is not divisible by the number of processes, using simple division (N/size) will leave some elements unprocessed. The original sample code would miss these remainder elements, leading to incorrect results.

**2. How can you modify the program to handle uneven partitions?**

The solution above handles uneven partitions by:

* Calculating remainder = N % size
* Giving one extra element to the first remainder processes
* Using MPI\_Scatterv instead of MPI\_Scatter to handle variable-sized chunks
* Each process calculates its own local\_n based on its rank

**3. How would performance differ between using MPI\_Reduce vs. MPI\_Gather + local summation?**

* **MPI\_Reduce**: More efficient - uses optimized reduction trees (O(log P) complexity), minimal memory usage at root
* **MPI\_Gather + local summation**: Less efficient - gathers all partial sums (O(P) memory at root), then root does final sum (O(P) operations), overall O(P) complexity

**4. How could this same approach be extended to matrix summation or averaging?**

For matrix operations:

* Distribute rows or columns using MPI\_Scatterv
* Each process computes partial sum/average of its sub-matrix
* Use MPI\_Reduce with appropriate datatypes
* For 2D decomposition, use MPI\_Cart\_create for grid topology

**Bonus Task :**

%%bash

cat > bonus.c <<'EOF'

#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

int main(int argc,char \*argv[]){

MPI\_Init(&argc,&argv);

int rank,size;

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD,&size);

int N=10; // not divisible by number of processes

int base=N/size;

int rem=N%size;

int local\_n=base+(rank<rem?1:0);

int \*A=NULL,\*sendcounts=NULL,\*displs=NULL;

if(rank==0){

A=(int\*)malloc(N\*sizeof(int));

sendcounts=(int\*)malloc(size\*sizeof(int));

displs=(int\*)malloc(size\*sizeof(int));

for(int i=0;i<N;i++)A[i]=i+1;

int offset=0;

for(int i=0;i<size;i++){

sendcounts[i]=base+(i<rem?1:0);

displs[i]=offset;

offset+=sendcounts[i];

}

}

int \*part=(int\*)malloc(local\_n\*sizeof(int));

MPI\_Scatterv(A,sendcounts,displs,MPI\_INT,part,local\_n,MPI\_INT,0,MPI\_COMM\_WORLD);

int local\_sum=0;

for(int i=0;i<local\_n;i++)local\_sum+=part[i];

int total\_sum=0;

MPI\_Reduce(&local\_sum,&total\_sum,1,MPI\_INT,MPI\_SUM,0,MPI\_COMM\_WORLD);

if(rank==0)printf("Total Sum (Uneven N)=%d\n",total\_sum);

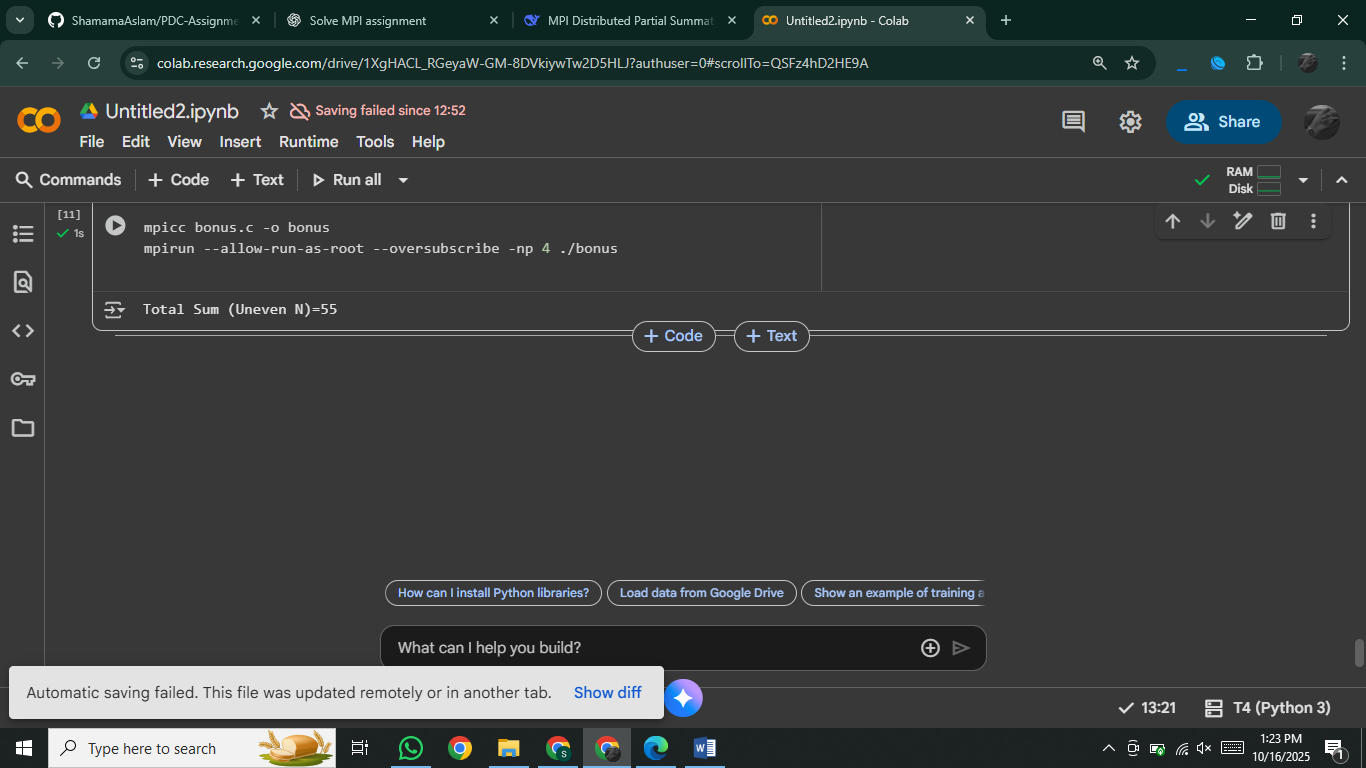
MPI\_Finalize();

return 0;

}

EOF

mpicc bonus.c -o bonus

mpirun --allow-run-as-root --oversubscribe -np 4 ./bonus