# Day18

## Table of Contents


- 1. Scripts
    - 1.1. compile script
    - 1.2. run script
- 2. PI calculator serial
- 3. PI calculator serial long long
- 4. PI calculator parallel
- 5. Prime number count
- 6. Prime number count parallel
- 7. Serial Matrix Addition
- 8. Serial Matrix Addition static memory allocation
- 9. Parallel Matrix Addition
- 10. Parallel Matrix Addition Using MPI_Scatter and MPI_Gather
- 11. Serial Matrix Multiplication


## 1. Scripts

### 1.1. compile script

```
#!/bin/sh

#source /opt/ohpc/pub/apps/spack/share/spack/setup-env.sh
#spack load gcc/5i5y5cb
#spack load openmpi/c7kvqyq
source ~/git/spack/share/spack/setup-env.sh
spack load openmpi

inputFile=$1
outputFile="${1%.*}.out"      # extract the name of the file without extension and adding extension .out
#cmd=`mpicc $inputFile -o $outputFile`
cmd="mpicc $inputFile -o $outputFile -lm -fopenmp"     # running code using MPI
echo "------------------------------------------------------------------"
```

```
echo "Command executed: $cmd"
echo "-------------------------------------------------------------------"
$cmd

echo "Compilation successful. Check at $outputFile"
echo "-------------------------------------------------------------------"
```

## 1.2. run script

```
#!/bin/sh

#source /opt/ohpc/pub/apps/spack/share/spack/setup-env.sh
#spack load gcc/5i5y5cbc
source ~/git/spack/share/spack/setup-env.sh
spack load openmpi

cmd="mpirun -np $2 $1"
echo "-------------------------------------------------------------------"
echo "Command executed: $cmd"
echo "-------------------------------------------------------------------"
echo "###################################################################"
echo "##########                    OUTPUT                    ##########"
echo "###################################################################"
echo
mpirun -np $2 $1
echo
echo "###################################################################"
echo "##########                     DONE                     ##########"
echo "###################################################################"
```

# 2. PI calculator serial

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<sys/time.h>
#define N 999999999
int main()
{
    int i, j;
    double area, pi;
```

```
    double dx, y, x;
    double exe_time;
    struct timeval stop_time, start_time;
    dx = 1.0/N;
    x = 0.0;
    area = 0.0;
    gettimeofday(&start_time, NULL);
    for(i=0;i<N;i++){
        x = i*dx;
        y = sqrt(1-x*x);
        area += y*dx;
    }
    gettimeofday(&stop_time, NULL);
    exe_time = (stop_time.tv_sec+(stop_time.tv_usec/1000000.0)) - (start_time.tv_sec+(start_time.tv_usec/1000000.0));
    pi = 4.0*area;
    printf("\n Value of pi is = %.16lf\n Execution time is = %lf seconds\n", pi, exe_time);
    return 0;
}
```

- Compile the program:

```
gcc pi_serial.c -o pi_serial.out -lm
```

- Run the program:

```
./pi_serial.out
```

```

Value of pi is = 3.1415926555902138
Execution time is = 2.033164 seconds
```

# 3. PI calculator serial long long

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<sys/time.h>
int main()
```

```
{
    const long long N = 99999999999;
    long long i, j;
    double area, pi;
    double dx, y, x;
    double exe_time;
    struct timeval stop_time, start_time;
    dx = (1.0 * 1L)/N;
    x = 0.0;
    area = 0.0;
    gettimeofday(&start_time, NULL);
    for(i=0;i<N;i++){
        x = i*dx;
        y = sqrt(1-x*x);
        area += y*dx;
    }
    gettimeofday(&stop_time, NULL);
    exe_time = (stop_time.tv_sec+(stop_time.tv_usec/1000000.0)) - (start_time.tv_sec+(start_time.tv_usec/1000000.0));
    pi = 4.0*area;
    printf("\n Value of pi is = %.16lf\n Execution time is = %lf seconds\n", pi, exe_time);
    return 0;
}
```

- Compile the program:

```
gcc pi_serial_ll.c -o pi_serial_ll.out -lm
```

- Run the program:

```
./pi_serial_ll.out
```

```

Value of pi is = 3.1415926535490444
Execution time is = 197.083734 seconds
```

## 4. PI calculator parallel

```
#include<stdio.h>
```

```c
#include<mpi.h>
#include<stdlib.h>
#include<math.h>
#include<sys/time.h>
int main()
{
    MPI_Init(NULL, NULL);
    const long long N = 99999999999;
    long long i, j;
    double area, pi;
    double dx, y, x;
    double exe_time;
    struct timeval stop_time, start_time;
    dx = 1.0/N;
    x = 0.0;
    area = 0.0;
    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    long long chunksize = N / size;
    long long start = rank * chunksize;
    long long end = start + chunksize;
    if(rank == size - 1) end = N;
    if(rank == 0)
        gettimeofday(&start_time, NULL);
    for(i=start;i<end;i++){
        x = i*dx;
        y = sqrt(1-x*x);
        area += y*dx;
    }
    double finalarea;
    MPI_Reduce(&area, &finalarea, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
    if(rank == 0){
        gettimeofday(&stop_time, NULL);
        exe_time = (stop_time.tv_sec+(stop_time.tv_usec/1000000.0)) - (start_time.tv_sec+(start_time.tv_usec/1000000.0));
        pi = 4.0*finalarea;
        printf("\n Value of pi is = %.16lf\n Execution time is = %lf seconds\n", pi, exe_time);
    }
    MPI_Finalize();
    return 0;
}
```

- Compile the program

```
bash compile.sh pi_parallel.c
```

```
-----------------------------------------------------------------
Command executed: mpicc pi_parallel.c -o pi_parallel.out -lm -fopenmp
-----------------------------------------------------------------
Compilation successful. Check at pi_parallel.out
-----------------------------------------------------------------
```

- Run the program:

```
bash run.sh ./pi_parallel.out 10
```

```
  -----------------------------------------------------------------
  Command executed: mpirun -np 10 ./pi_parallel.out
  -----------------------------------------------------------------
  #################################################################
  ##########                  OUTPUT                   ##########
  #################################################################


   Value of pi is = 3.1415926536117809
   Execution time is = 77.906474 seconds

  #################################################################
  ##########                   DONE                    ##########
  #################################################################
```

# 5. Prime number count

```c
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<time.h>
#include<sys/time.h>

#define N 1000000
/*
                N  PRIME_NUMBER

                1           0
               10           4
              100          25
```

```
             1,000         168
            10,000       1,229
           100,000       9,592
         1,000,000      78,498
        10,000,000     664,579
       100,000,000   5,761,455
     1,000,000,000  50,847,534

*/

int main()
{
    int i, j;
    int count, flag;
    double exe_time;
    struct timeval stop_time, start_time;

    count = 1; // 2 is prime. Our loop starts from 3

    gettimeofday(&start_time, NULL);


    for(i=3;i<N;i++)
    {
        flag = 0;
        for(j=2;j<i;j++)
        {
            if((i%j) == 0)
            {
                flag = 1;
                break;
            }
        }
        if(flag == 0)
        {
            count++;
        }
    }

    gettimeofday(&stop_time, NULL);
    exe_time = (stop_time.tv_sec+(stop_time.tv_usec/1000000.0)) - (start_time.tv_sec+(start_time.tv_usec/1000000.0));

    printf("\n Number of prime numbers = %d \n Execution time is = %lf seconds\n", count, exe_time);

}
```

- Compile the program:

```
gcc prime_count_serial.c -o prime_count_serial.out -lm
```

- Run the program:

```
./prime_count_serial.out
```

```
Number of prime numbers = 78498
Execution time is = 52.188054 seconds
```

# 6. Prime number count parallel

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<time.h>
#include<sys/time.h>
#include<mpi.h>

#define N 100000

int main()
{
    MPI_Init(NULL, NULL);
    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    int i, j;
    int count, flag;
    double exe_time;
    struct timeval stop_time, start_time;
    int chunksize = N / size;
    int start = rank * chunksize;
    int end = start + chunksize;
    if(rank == size - 1) end = N;
    count = 0;
    if(rank == 0){
```

```
        count = 1;
        start = 3;
    }
    gettimeofday(&start_time, NULL);
    for(i=start;i<end;i++)
    {
        flag = 0;
        for(j=2;j<i;j++)
        {
            if((i%j) == 0)
            {
                flag = 1;
                break;
            }
        }
        if(flag == 0)
        {
            count++;
        }
    }
    int total_count = 0;
    MPI_Reduce(&count, &total_count, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

    if(rank == 0){
        gettimeofday(&stop_time, NULL);
        exe_time = (stop_time.tv_sec+(stop_time.tv_usec/1000000.0)) - (start_time.tv_sec+(start_time.tv_usec/1000000.0));
        printf("\n Number of prime numbers = %d \n Execution time is = %lf seconds\n", total_count, exe_time);
    }
    MPI_Finalize();

}
```

- Compile the program:

```
bash compile.sh prime_count_parallel.c
```

```
------------------------------------------------------------------
Command executed: mpicc prime_count_parallel.c -o prime_count_parallel.out -lm -fopenmp
------------------------------------------------------------------
Compilation successful. Check at prime_count_parallel.out
------------------------------------------------------------------
```

- Run the program:

```
bash run.sh ./prime_count_parallel.out 10
```

```
------------------------------------------------------------------
Command executed: mpirun -np 10 ./prime_count_parallel.out
------------------------------------------------------------------
##################################################################
##########                    OUTPUT                    ##########
##################################################################


 Number of prime numbers = 9592
 Execution time is = 0.190704 seconds

##################################################################
##########                     DONE                     ##########
##################################################################
```

# 7. Serial Matrix Addition

```c
#include<stdio.h>
#include<omp.h>
#include<stdlib.h>

int main(int argc, char **argv){
    int i, j, n = 400;
    int **m1, **m2, **sumMat;
    m1 = (int**)malloc(sizeof(int*) * n);
    m2 = (int**)malloc(sizeof(int*) * n);
    sumMat = (int**)malloc(sizeof(int*) * n);
    for(i = 0; i < n; i++){
        sumMat[i] = (int*)malloc(sizeof(int) * n);
        m1[i] = (int*)malloc(sizeof(int) * n);
        m2[i] = (int*)malloc(sizeof(int) * n);
        for(j = 0; j < n; j++){
            m1[i][j] = 1;
            m2[i][j] = 1;
            sumMat[i][j] = 0;
        }
    }

    for(i = 0; i < n; i++){
        for(j = 0; j < n; j++){
```

```
            sumMat[i][j] = m1[i][j] + m2[i][j];
        }
    }

    for(i = 0; i < n; i++){
        for(j = 0; j < n; j++){
            printf("%d ",sumMat[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

```
#bash compile.sh serial_mat_add.c
gcc serial_mat_add.c -fopenmp
```

```
#bash run.sh ./serial_mat_add.out 10 > output.txt
./a.out > output2.txt
```

# 8. Serial Matrix Addition static memory allocation

```
#include<stdio.h>
#include<stdlib.h>

int main(int argc, char **argv){
    const int n = 400;
    int i, j;
    int m1[n][n], m2[n][n], sumMat[n][n];
    for(i = 0; i < n; i++){
        for(j = 0; j < n; j++){
            m1[i][j] = 1;
            m2[i][j] = 1;
        }
    }
    /*
    for(i = 0; i < n; i++){
        for(j = 0; j < n; j++){
            printf("%d ",m1[i][j]);
        }
        printf("\n");
```

```
    }
    for(i = 0; i < n; i++){
        for(j = 0; j < n; j++){
            printf("%d ",m2[i][j]);
        }
        printf("\n");
    }*/

    for(i = 0; i < n; i++){
        for(j = 0; j < n; j++){
            sumMat[i][j] = m1[i][j] + m2[i][j];
        }
    }
    for(i = 0; i < n; i++){
        for(j = 0; j < n; j++){
            printf("%d ",sumMat[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

```
bash compile.sh serial_mat_add_static.c
```

```
------------------------------------------------------------------
Command executed: mpicc serial_mat_add_static.c -o serial_mat_add_static.out -lm -fopenmp
------------------------------------------------------------------
Compilation successful. Check at serial_mat_add_static.out
------------------------------------------------------------------
```

```
bash run.sh ./serial_mat_add_static.out 10 > output1.txt
```

# 9. Parallel Matrix Addition

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

int main(int argc, char **argv){
```

```
MPI_Init(NULL, NULL);
int n = 400;
int **m1, **m2, **sumMat;
m1 = (int**)malloc(sizeof(int*) * n);
m2 = (int**)malloc(sizeof(int*) * n);
sumMat = (int**)malloc(sizeof(int*) * n);
int rank, size;
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);

if(rank == 0){
    for(int i = 0; i < n; i++){
        sumMat[i] = (int*)malloc(sizeof(int) * n);
        m1[i] = (int*)malloc(sizeof(int) * n);
        m2[i] = (int*)malloc(sizeof(int) * n);
        for(int j = 0; j < n; j++){
            m1[i][j] = 1;
            m2[i][j] = 1;
            sumMat[i][j] = 0;
        }
    }
}
int chunksize = n / size;
int start = rank * chunksize;
int end = start + chunksize;
if(rank == size - 1) end = n;

MPI_Datatype contiguous_type;

// Create a contiguous datatype
MPI_Type_contiguous(count, MPI_INT, &contiguous_type);
MPI_Type_commit(&contiguous_type);

int **localArr1 = (int**) malloc(sizeof(int*) * chunksize);
int **localArr2 = (int**) malloc(sizeof(int*) * chunksize);
int **sumArr= (int**) malloc(sizeof(int*) * chunksize);
for(int i = 0; i < chunksize; i++){
    localArr1[i] = (int*) malloc(sizeof(int) * n);
    localArr2[i] = (int*) malloc(sizeof(int) * n);
    sumArr[i] = (int*) malloc(sizeof(int) * n);
}
for(int i = start; i < end; i++){
    MPI_Scatter(m1[i], n, MPI_INT, localArr1[i], n, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Scatter(m2[i], n, MPI_INT, localArr2[i], n, MPI_INT, 0, MPI_COMM_WORLD);
}


for(int i = start; i < end; i++){
```

```
        for(int j = 0; j < n; j++){
            sumArr[i][j] = localArr1[i][j] + localArr2[i][j];
        }
    }
    for(int i = 0; i < chunksize; i++){
        MPI_Gather(sumArr[i], chunksize, MPI_INT, sumMat, chunksize * n, MPI_INT, 0, MPI_COMM_WORLD);
    }


        for(int i = start; i < end; i++){
            for(int j = 0; j < n; j++){
                printf("%d ",sumMat[i][j]);
            }
            printf("\n");
    }


    MPI_Finalize();
    return 0;
}
```

```
bash compile.sh parallel_matrix_add.c
```

```
------------------------------------------------------------------
Command executed: mpicc parallel_matrix_add.c -o parallel_matrix_add.out -lm -fopenmp
------------------------------------------------------------------
Compilation successful. Check at parallel_matrix_add.out
------------------------------------------------------------------
```

```
bash run.sh ./parallel_matrix_add.out 10 > output3.txt
```

# 10. Parallel Matrix Addition Using MPI_Scatter and MPI_Gather

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

int main() {
    int i, j, rank, size, n = 10000;
    int *m1, *m2, *sumMat, *sub_m1, *sub_m2, *sub_sumMat;
```

```c
    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    int chunksize = (n * n) / size;

    // Allocate memory for the full matrices on the root process
    if (rank == 0) {
        m1 = (int*)malloc(n * n * sizeof(int));
        m2 = (int*)malloc(n * n * sizeof(int));
        sumMat = (int*)malloc(n * n * sizeof(int));
        for (i = 0; i < n * n; i++) {
            m1[i] = 1;
            m2[i] = 1;
        }
    }

    // Allocate memory for the submatrices on each process
    sub_m1 = (int*)malloc(chunksize * sizeof(int));
    sub_m2 = (int*)malloc(chunksize * sizeof(int));
    sub_sumMat = (int*)malloc(chunksize * sizeof(int));

    double startTime = MPI_Wtime();
    // Scatter the elements of the matrices to all processes
    MPI_Scatter(m1, chunksize, MPI_INT, sub_m1, chunksize, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Scatter(m2, chunksize, MPI_INT, sub_m2, chunksize, MPI_INT, 0, MPI_COMM_WORLD);

    // Perform the addition on the submatrices
    for (i = 0; i < chunksize; i++) {
        sub_sumMat[i] = sub_m1[i] + sub_m2[i];
    }

    // Gather the results from all processes
    MPI_Gather(sub_sumMat, chunksize, MPI_INT, sumMat, chunksize, MPI_INT, 0, MPI_COMM_WORLD);

    double endTime = MPI_Wtime();
    // Print the result on the root process
    if (rank == 0) {
        int flag = 1;
        for (i = 0; i < n * n; i++) {
            if (sumMat[i] != 2) {
                flag = 0;
                break;
            }
        }
        if (flag){
          printf("_____PASS_____\n");
```

```
            printf("Execution time: %lf\n", endTime - startTime);
        }
        else printf("_____FAIL_____\n");
        // Free the allocated memory
        free(m1);
        free(m2);
        free(sumMat);
    }
    free(sub_m1);
    free(sub_m2);
    free(sub_sumMat);
    MPI_Finalize();
    return 0;
}
```

- Compile

```
bash compile.sh mpi_matrix_addition1.c
```

```
------------------------------------------------------------------
Command executed: mpicc mpi_matrix_addition1.c -o mpi_matrix_addition1.out -lm -fopenmp
------------------------------------------------------------------
Compilation successful. Check at mpi_matrix_addition1.out
------------------------------------------------------------------
```

- Run

```
bash run.sh ./mpi_matrix_addition1.out 10
```

```
------------------------------------------------------------------
Command executed: mpirun -np 10 ./mpi_matrix_addition1.out
------------------------------------------------------------------
##################################################################
##########                    OUTPUT                    ##########
##################################################################

_____PASS_____
Execution time: 0.513867

##################################################################
```

```
##########                    DONE                    ##########
################################################################
```

# 11. Serial Matrix Multiplication

```c
#include <stdio.h>
#include <stdlib.h>
#include<omp.h>

int main() {
    int n = 2000;
    int i, j, k;

    // Allocate memory for matrices
    int **A = (int **)malloc(n * sizeof(int *));
    int **B = (int **)malloc(n * sizeof(int *));
    int **C = (int **)malloc(n * sizeof(int *));
    for (i = 0; i < n; i++) {
        A[i] = (int *)malloc(n * sizeof(int));
        B[i] = (int *)malloc(n * sizeof(int));
        C[i] = (int *)malloc(n * sizeof(int));
    }

    // Initialize matrices
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            A[i][j] = 1;
            B[i][j] = 1;
            C[i][j] = 0;
        }
    }

    double starttime = omp_get_wtime();
    // Matrix multiplication
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            for (k = 0; k < n; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
    double endtime = omp_get_wtime();

    // Print result
```

```c
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            printf("%d ", C[i][j]);
        }
        printf("\n");
    }
    printf("execution time: %lf\n", endtime - starttime);

    // Free allocated memory
    for (i = 0; i < n; i++) {
        free(A[i]);
        free(B[i]);
        free(C[i]);
    }
    free(A);
    free(B);
    free(C);

    return 0;
}
```

```
bash compile.sh serial_matrix_multiplication.c
```

```
------------------------------------------------------------------
Command executed: mpicc serial_matrix_multiplication.c -o serial_matrix_multiplication.out -lm -fopenmp
------------------------------------------------------------------
Compilation successful. Check at serial_matrix_multiplication.out
------------------------------------------------------------------
```

```
bash run.sh ./serial_matrix_multiplication.out 10 > output.txt
```

Author: Abhishek Raj
Created: 2025-01-13 Mon 17:36