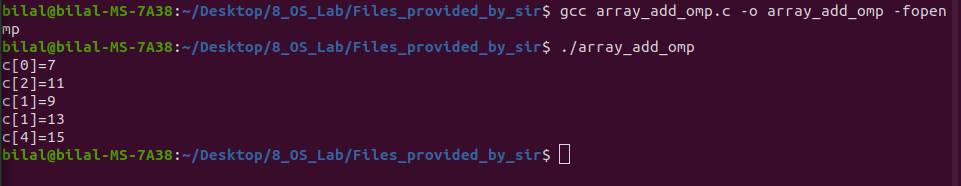
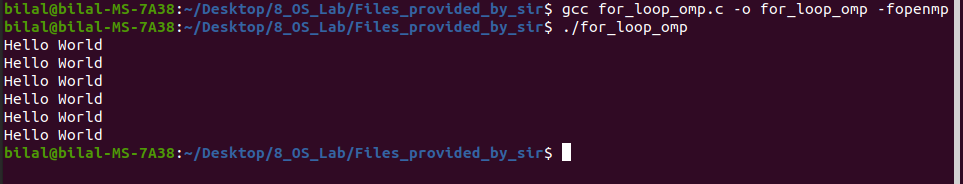
**Task 01**

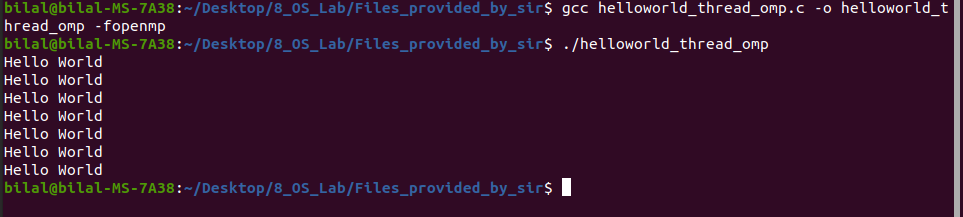
1. array\_add\_omp



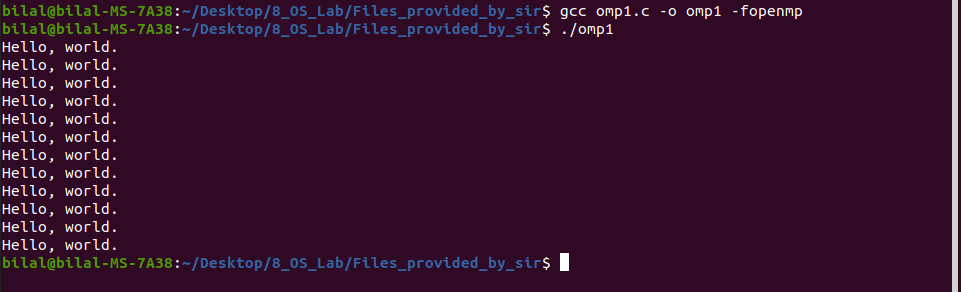
2. for\_loop\_omp



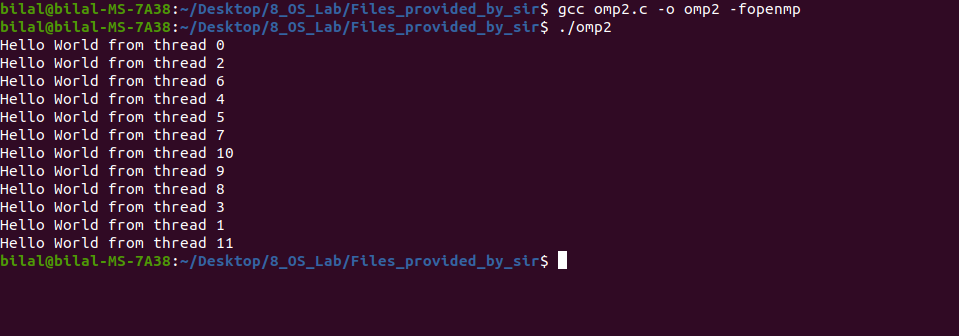
3. helloworld\_thread\_omp



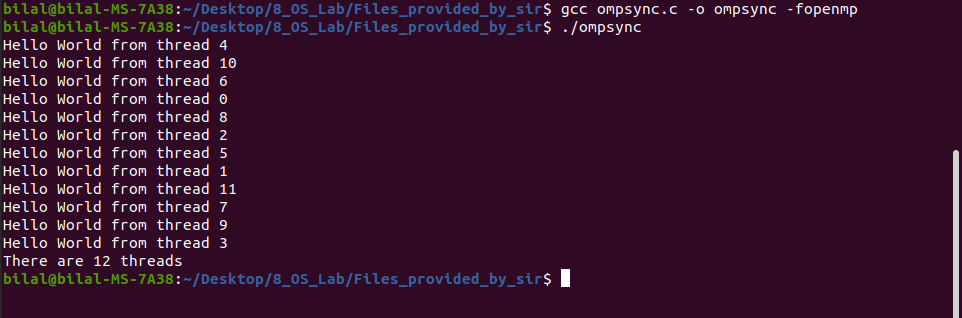
4. omp1



5. omp2

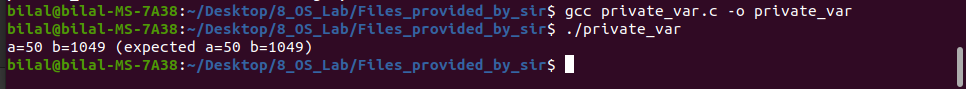


6. ompsync

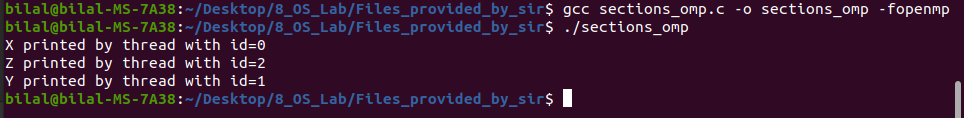


7. parallel\_loop

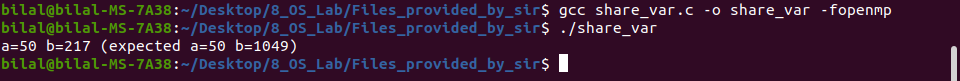


8. private\_var

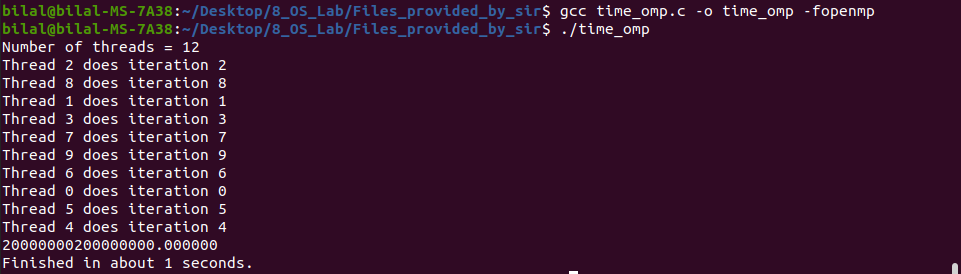
9. sections\_omp



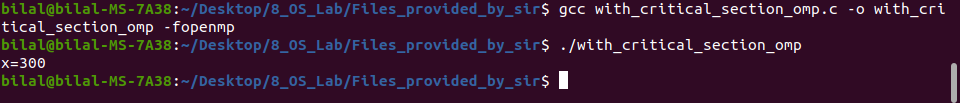
10. share\_var



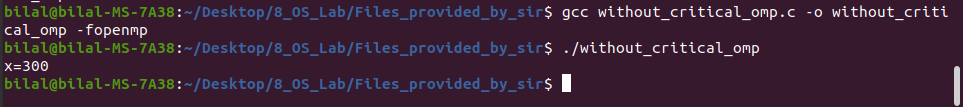
11. time\_omp



12. with\_critical\_section\_omp



13. without\_critical\_omp



**Task 02: part(a)**

**#include<omp.h>**

#include<stdio.h>

#include<time.h>

#include<stdlib.h>

#define N 2000000000

// 10 arab iterations

int main()

{

clock\_t t;

time\_t start, stop;

printf("This file will check the time period without openmp.\nWe are taking the value of N=2000000000.\n");

float sum=0;

t=clock();

time(&start);

for (int i = 1; i < N; i++)

{

sum=1/(float)i+sum;

// printf("1/i= %f\nsum= %f.\n\n", (float)1/i,(float)sum);

}

printf("%.5f\n", sum);

time(&stop);

t=clock()-t;

double time\_taken=((double)t)/CLOCKS\_PER\_SEC;

printf("It took %.0f seconds to perform this task without parallelism.\n", difftime(stop, start));

}

**Task 02: part(b)**

**#include <omp.h>**

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#define N 2000000000

// 10 arab iterations

int main()

{

clock\_t t;

time\_t start, stop;

long long int i;

printf("This file will check the time period without openmp.\nWe are taking the value of N=2000000000.\n");

float sum = 0;

t = clock();

time(&start);

#pragma omp parallel for reduction(+:sum)

for ( i = 1; i < N; i++)

{

sum+=1/(float)i;

}

printf("%f\n",sum);

time(&stop);

t = clock() - t;

// double time\_taken=((double)t)/CLOCKS\_PER\_SEC;

printf("It took %.0f seconds to perform this task with parallelism.\n", difftime(stop, start));

}

**Task 03: part(a)**

**#include <stdio.h>**

#include <omp.h>

#include <stdlib.h>

#include<time.h>

#define N 835

void printmatrix(int Matrix[][N])

{

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N; j++)

{

printf("%d ", Matrix[i][j]);

}

printf("\n");

}

}

void Assign\_values(int Matrix[][N])

{

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N; j++)

{

Matrix[i][j] = rand() % N;

}

}

}

void Add\_matrices(int Matrix01[][N], int Matrix02[][N], int Ans\_Matrix[][N])

{

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N; j++)

{

Ans\_Matrix[i][j] = Matrix01[i][j] + Matrix02[i][j];

}

}

}

int main()

{

time\_t start, end;

time(&start);

printf("Adding 2 NxN matrices without parallelism.\n");

int matrix\_01[N][N] = {}, matrix\_02[N][N] = {}, matrix\_ans[N][N] = {};

// Assigning random values to both matrices

Assign\_values(matrix\_01);

Assign\_values(matrix\_02);

// displayig both matrices

// printf("This is matrix 01\n");

// printmatrix(matrix\_01);

// printf("\nThis is matrix 02\n");

// printmatrix(matrix\_02);

// now adding both matrices

Add\_matrices(matrix\_01,matrix\_02,matrix\_ans);

printf("\nAfter adding matrix 01 and matrix 02 we get:\n");

printmatrix(matrix\_ans);

time(&end);

printf("It took %0.f seconds to add 2 matrices of 835x835 without parallelism.\n",difftime(end,start));

}

**Task 03: part(b)**

**#include <stdio.h>**

#include <omp.h>

#include <stdlib.h>

#include<time.h>

#define N 835

void printmatrix(int Matrix[][N])

{

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N; j++)

{

printf("%d ", Matrix[i][j]);

}

printf("\n");

}

}

void Assign\_values(int Matrix[][N])

{

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N; j++)

{

Matrix[i][j] = rand() % N;

}

}

}

void Add\_matrices(int Matrix01[][N], int Matrix02[][N], int Ans\_Matrix[][N])

{

#pragma omp parallel for

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N; j++)

{

Ans\_Matrix[i][j] = Matrix01[i][j] + Matrix02[i][j];

}

}

}

int main()

{

time\_t start, end;

time(&start);

printf("Adding 2 NxN matrices without parallelism.\n");

int matrix\_01[N][N] = {}, matrix\_02[N][N] = {}, matrix\_ans[N][N] = {};

// Assigning random values to both matrices

Assign\_values(matrix\_01);

Assign\_values(matrix\_02);

// displayig both matrices

// printf("This is matrix 01\n");

// printmatrix(matrix\_01);

// printf("\nThis is matrix 02\n");

// printmatrix(matrix\_02);

// now adding both matrices

Add\_matrices(matrix\_01,matrix\_02,matrix\_ans);

printf("\nAfter adding matrix 01 and matrix 02 we get:\n");

printmatrix(matrix\_ans);

time(&end);

printf("It took %0.f seconds to add 2 matrices of 835x835 with parallelism.\n",difftime(end,start));

}