VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Parallel and Distributed Computing-CSE4001L
S. DHANYA ABHIRAMI
16BCE0965

Lab Slots: L9+L10 **Date:** 4th September 2018

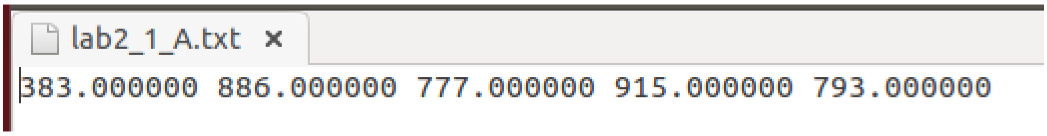
ASSESSMENT 2

1. Write a OpenMP program to show data environmental clauses variable scope using one dimensional array addition (private, first private, Last private and Shared).

Code

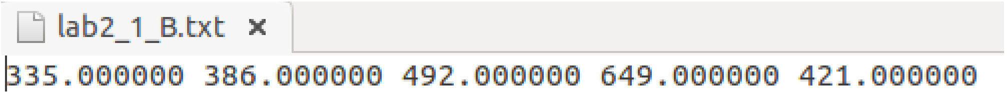
Values in files

A



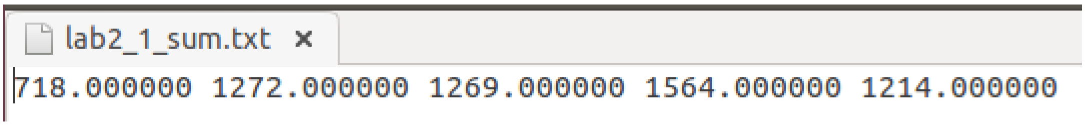
```
383.000000 886.000000 777.000000 915.000000 793.000000
```

B



```
335.000000 386.000000 492.000000 649.000000 421.000000
```

Sum



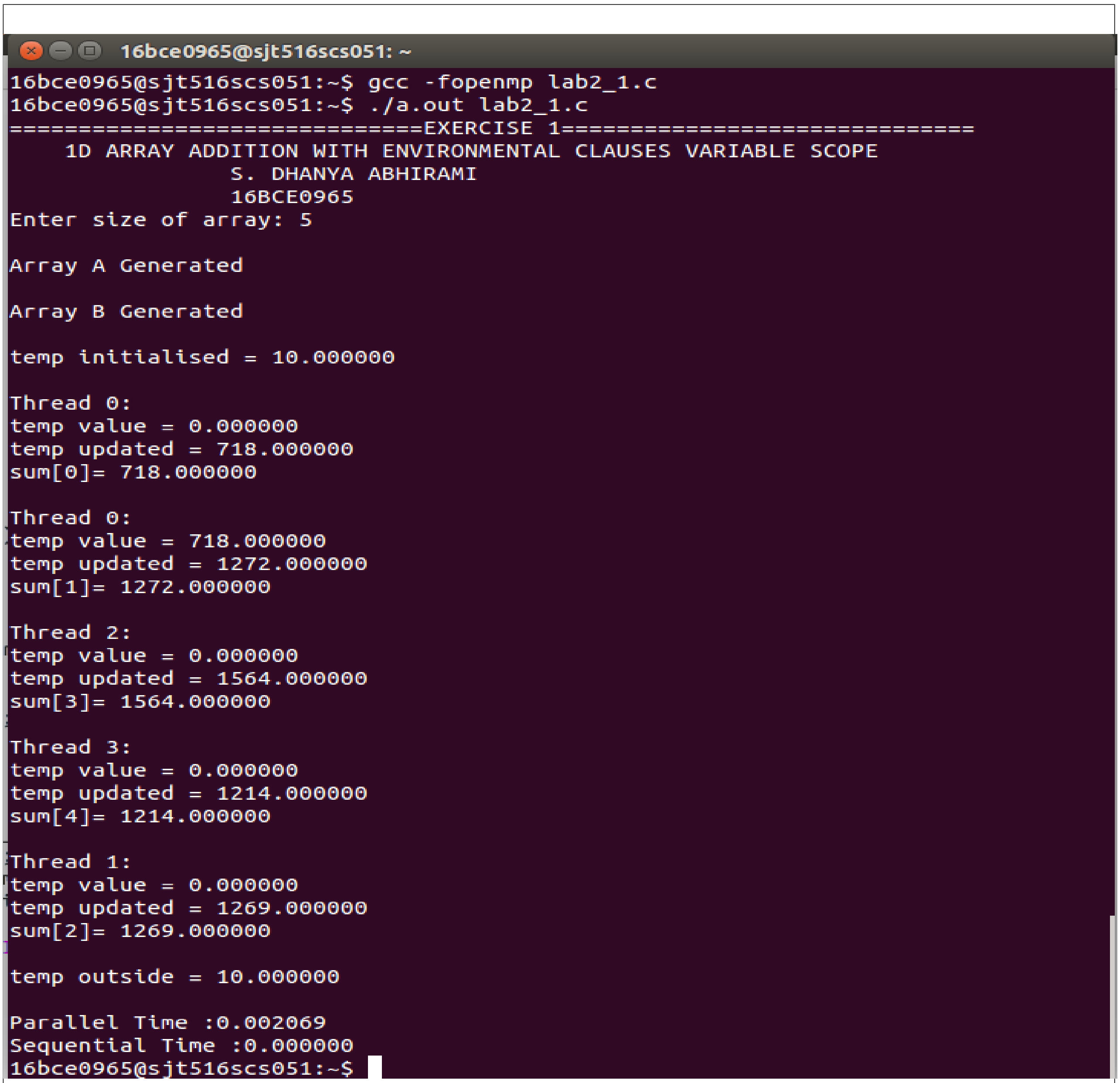
```
718.000000 1272.000000 1269.000000 1564.000000 1214.000000
```

Private

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include<time.h>
int main ()
{
	printf("============================EXERCISE
1============================\n");
   printf("    1D ARRAY ADDITION WITH ENVIRONMENTAL CLAUSES VARIABLE
SCOPE\n\t\tS. DHANYA ABHIRAMI\n\t\t16BCE0965\n");
	int i,n;
	float m;
	double parallel_start, parallel_end,seq_start,seq_end,parallel_time,seq_time;
```

```
        FILE *fptr;
        printf("Enter size of array: ");
        scanf("%d",&n);
        float *a=(float *) malloc (n*sizeof(float));
        float *b=(float *) malloc (n*sizeof(float));
        float *sum=(float *) malloc (n*sizeof(float));
        fptr = fopen("/home/likewise-open/VITUNIVERSITY/16bce0965/lab2_1_A.txt", "w");
  if(fptr == NULL)
  {
    printf("Error!");
    exit(1);
  }
  for (i = 0; i < n; ++i)
  {
        m = rand()%1000;
      fprintf(fptr,"%f ", m);
  }

  fclose(fptr);
  printf("\nArray A Generated\n");

 fptr = fopen("/home/likewise-open/VITUNIVERSITY/16bce0965/lab2_1_B.txt", "w");
  if(fptr == NULL)
  {
    printf("Error!");
    exit(1);
  }
  for (i = 0; i < n; ++i)
  {
    m = rand()%1000;
    fprintf(fptr,"%f ", m);
  }
  fclose(fptr);
  printf("\nArray B Generated\n\n");
  // Reading arrays from files
 fptr = fopen("/home/likewise-open/VITUNIVERSITY/16bce0965/lab2_1_A.txt", "r");
 if(fptr == NULL)
 {
   printf("Error!");
   exit(1);
 }
 for (i = 0; i < n; ++i)
 {
   fscanf(fptr,"%f",&m);
    a[i]=m;
}
fclose(fptr);

 fptr = fopen("/home/likewise-open/VITUNIVERSITY/16bce0965/lab2_1_B.txt", "r");
 if(fptr == NULL)
 {
   printf("Error!");
```

```
    exit(1);
  }

  for (i = 0; i < n; ++i)
  {
    fscanf(fptr,"%f",&m);
    b[i]=m;
 }
 fclose(fptr);
seq_start = omp_get_wtime();
 for (i=0; i<n; i++)
  {
    sum[i] = a[i] + b[i];
  }
  seq_end = omp_get_wtime();
seq_time = seq_end - seq_start;
 float temp=10.0;
 printf("temp initialised = %f\n",temp );
 parallel_start = omp_get_wtime();
 #pragma omp parallel for private(temp)
 for (i=0; i<n; i++)
  {
    printf("\nThread %d: \n",omp_get_thread_num());
    printf("temp value = %f\n",temp );
    temp = a[i] + b[i];
    printf("temp updated = %f\n",temp );
    sum[i] = temp;
    printf("sum[%d]= %f\n",i,sum[i]);
  }
parallel_end = omp_get_wtime();
parallel_time = parallel_end - parallel_start;
printf("\ntemp outside = %f\n\n",temp );
printf("Parallel Time :%lf",parallel_time);
printf("\nSequential Time :%lf\n",seq_time);
  // Storing Output in file
    fptr = fopen("/home/likewise-open/VITUNIVERSITY/16bce0965/lab2_1_sum.txt", "w");

    for (i = 0; i < n; ++i)
    {
      fprintf(fptr,"%f ",sum[i]);
   }
   fclose(fptr);
            return (0);
 }
```

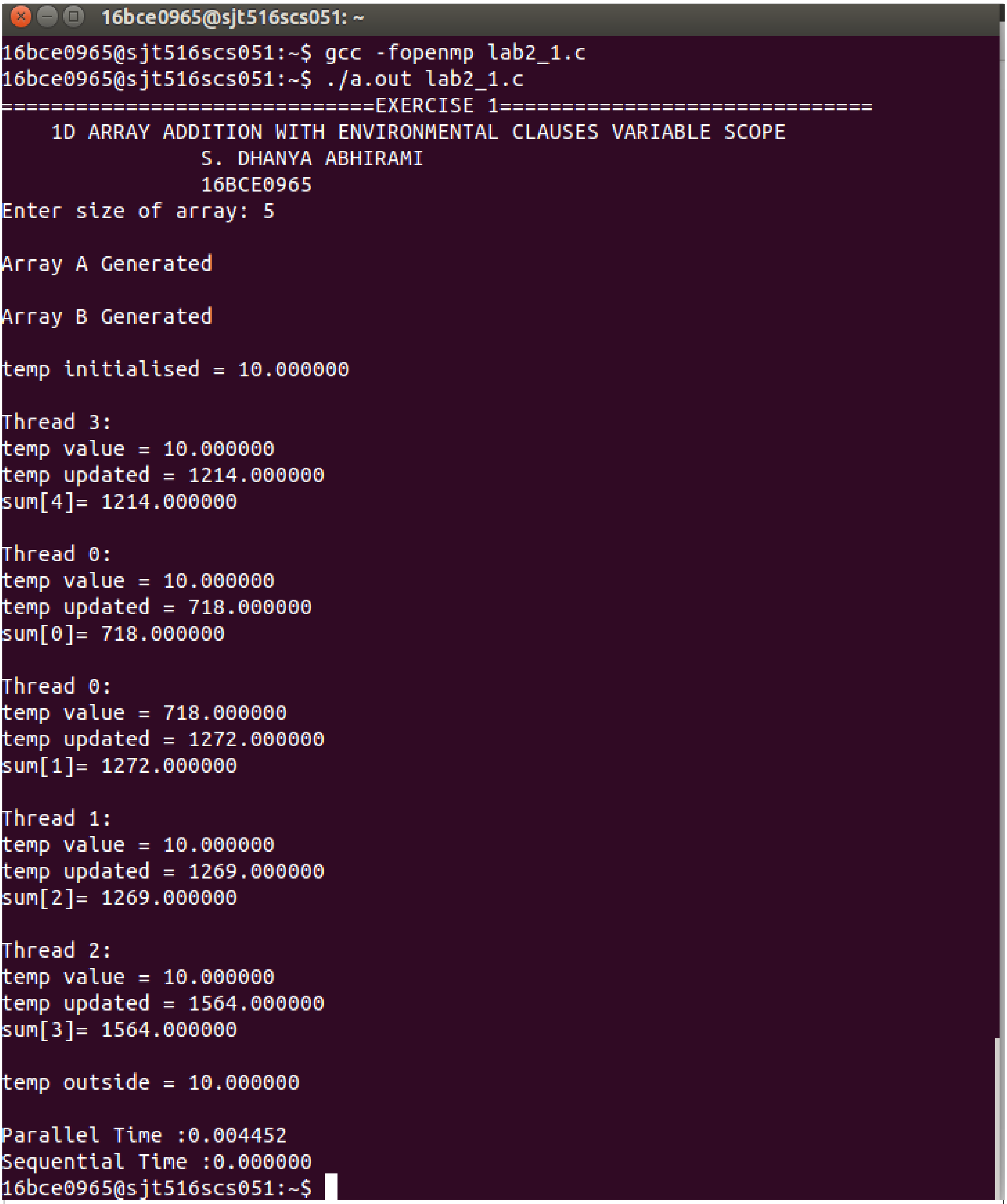


```
16bce0965@sjt516scs051:~$ gcc -fopenmp lab2_1.c
16bce0965@sjt516scs051:~$ ./a.out lab2_1.c
===============================EXERCISE 1===============================
    1D ARRAY ADDITION WITH ENVIRONMENTAL CLAUSES VARIABLE SCOPE
                S. DHANYA ABHIRAMI
                16BCE0965
Enter size of array: 5

Array A Generated

Array B Generated

temp initialised = 10.000000

Thread 0:
temp value = 0.000000
temp updated = 718.000000
sum[0]= 718.000000

Thread 0:
temp value = 718.000000
temp updated = 1272.000000
sum[1]= 1272.000000

Thread 2:
temp value = 0.000000
temp updated = 1564.000000
sum[3]= 1564.000000

Thread 3:
temp value = 0.000000
temp updated = 1214.000000
sum[4]= 1214.000000

Thread 1:
temp value = 0.000000
temp updated = 1269.000000
sum[2]= 1269.000000

temp outside = 10.000000

Parallel Time :0.002069
Sequential Time :0.000000
16bce0965@sjt516scs051:~$
```

First Private

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include<time.h>
int main ()
{
	printf("===============================EXERCISE 
1===============================\n");
   printf("    1D ARRAY ADDITION WITH ENVIRONMENTAL CLAUSES VARIABLE 
SCOPE\n\t\tS. DHANYA ABHIRAMI\n\t\t16BCE0965\n");
	int i,n;
	float m;
	double parallel_start, parallel_end,seq_start,seq_end,parallel_time,seq_time;
	FILE *fptr;
```

```
        printf("Enter size of array: ");
        scanf("%d",&n);
        float *a=(float *) malloc (n*sizeof(float));
        float *b=(float *) malloc (n*sizeof(float));
        float *sum=(float *) malloc (n*sizeof(float));
        fptr = fopen("/home/likewise-open/VITUNIVERSITY/16bce0965/lab2_1_A.txt", "w");
  if(fptr == NULL)
  {
     printf("Error!");
     exit(1);
  }
  for (i = 0; i < n; ++i)
  {
        m = rand()%1000;
      fprintf(fptr,"%f ", m);
  }

  fclose(fptr);
  printf("\nArray A Generated\n");

 fptr = fopen("/home/likewise-open/VITUNIVERSITY/16bce0965/lab2_1_B.txt", "w");
  if(fptr == NULL)
  {
     printf("Error!");
     exit(1);
  }
  for (i = 0; i < n; ++i)
  {
     m = rand()%1000;
     fprintf(fptr,"%f ", m);
  }
  fclose(fptr);
  printf("\nArray B Generated\n\n");
  // Reading arrays from files
 fptr = fopen("/home/likewise-open/VITUNIVERSITY/16bce0965/lab2_1_A.txt", "r");
 if(fptr == NULL)
 {
   printf("Error!");
   exit(1);
 }
 for (i = 0; i < n; ++i)
 {
   fscanf(fptr,"%f",&m);
    a[i]=m;
}
fclose(fptr);

 fptr = fopen("/home/likewise-open/VITUNIVERSITY/16bce0965/lab2_1_B.txt", "r");
 if(fptr == NULL)
 {
   printf("Error!");
   exit(1);
  }

  for (i = 0; i < n; ++i)
  {
    fscanf(fptr,"%f",&m);
    b[i]=m;
 }
 fclose(fptr);
seq_start = omp_get_wtime();
 for (i=0; i<n; i++)
  {
    sum[i] = a[i] + b[i];
  }
  seq_end = omp_get_wtime();
seq_time = seq_end - seq_start;
 float temp=10.0;
 printf("temp initialised = %f\n",temp );
 parallel_start = omp_get_wtime();
 #pragma omp parallel for firstprivate(temp)
 for (i=0; i<n; i++)
  {
    printf("\nThread %d: \n",omp_get_thread_num());
    printf("temp value = %f\n",temp );
    temp = a[i] + b[i];
    printf("temp updated = %f\n",temp );
    sum[i] = temp;
    printf("sum[%d]= %f\n",i,sum[i]);
  }
parallel_end = omp_get_wtime();
parallel_time = parallel_end - parallel_start;
printf("\ntemp outside = %f\n\n",temp );
printf("Parallel Time :%lf",parallel_time);
printf("\nSequential Time :%lf\n",seq_time);
  // Storing Output in file
    fptr = fopen("/home/likewise-open/VITUNIVERSITY/16bce0965/lab2_1_sum.txt", "w");

    for (i = 0; i < n; ++i)
    {
      fprintf(fptr,"%f ",sum[i]);
   }
   fclose(fptr);
          return (0);
 }
```

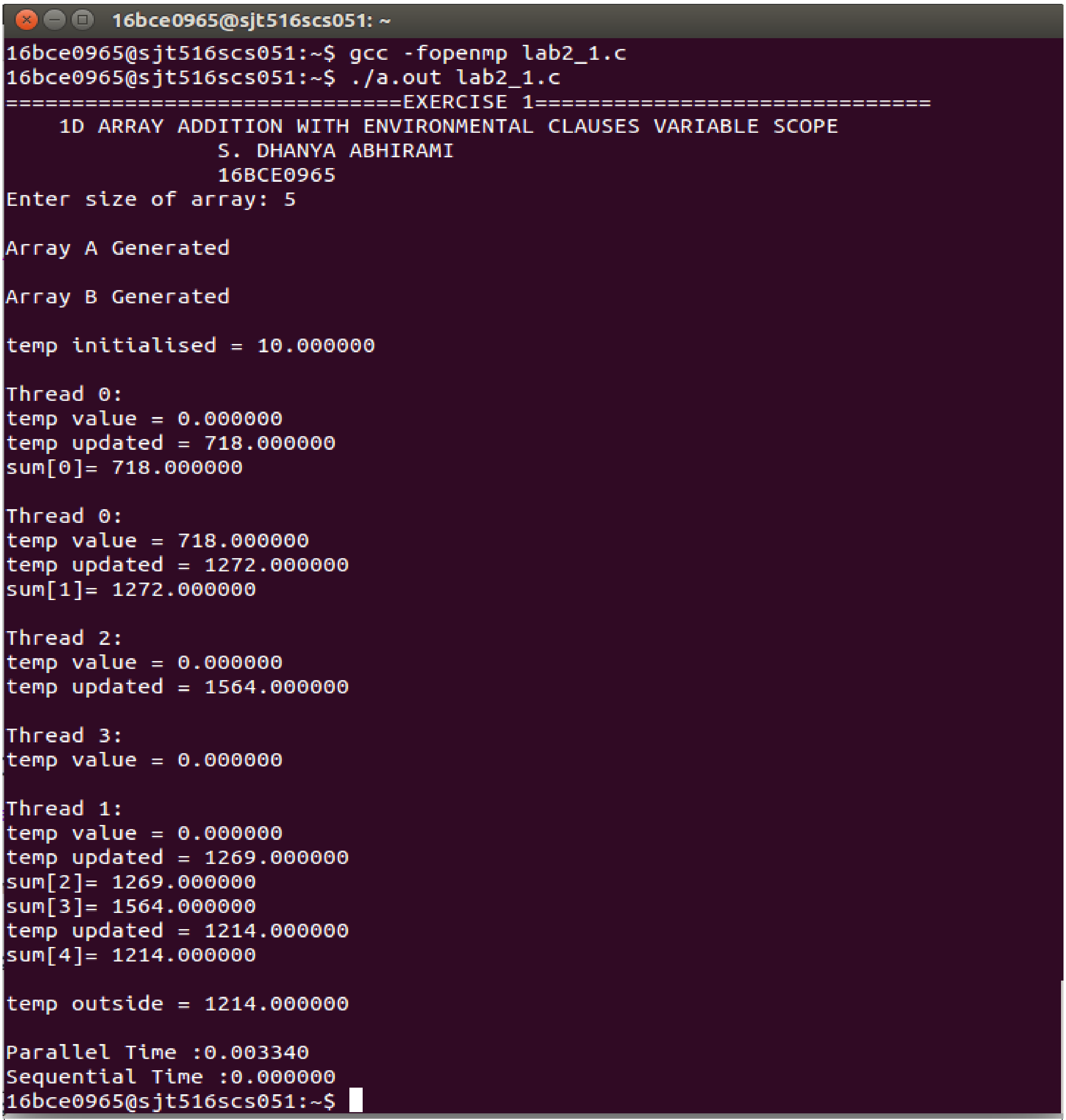


```
16bce0965@sjt516scs051:~$ gcc -fopenmp lab2_1.c
16bce0965@sjt516scs051:~$ ./a.out lab2_1.c
==============================EXERCISE 1==============================
    1D ARRAY ADDITION WITH ENVIRONMENTAL CLAUSES VARIABLE SCOPE
                S. DHANYA ABHIRAMI
                16BCE0965
Enter size of array: 5

Array A Generated

Array B Generated

temp initialised = 10.000000

Thread 3:
temp value = 10.000000
temp updated = 1214.000000
sum[4]= 1214.000000

Thread 0:
temp value = 10.000000
temp updated = 718.000000
sum[0]= 718.000000

Thread 0:
temp value = 718.000000
temp updated = 1272.000000
sum[1]= 1272.000000

Thread 1:
temp value = 10.000000
temp updated = 1269.000000
sum[2]= 1269.000000

Thread 2:
temp value = 10.000000
temp updated = 1564.000000
sum[3]= 1564.000000

temp outside = 10.000000

Parallel Time :0.004452
Sequential Time :0.000000
16bce0965@sjt516scs051:~$ 
```

Last Private

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include<time.h>
int main ()
{
        printf("==============================EXERCISE 
1==============================\n");
   printf("    1D ARRAY ADDITION WITH ENVIRONMENTAL CLAUSES VARIABLE
SCOPE\n\t\tS. DHANYA ABHIRAMI\n\t\t16BCE0965\n");
            int i,n;
            float m;
            double parallel_start, parallel_end,seq_start,seq_end,parallel_time,seq_time;
            FILE *fptr;
            printf("Enter size of array: ");
            scanf("%d",&n);
            float *a=(float *) malloc (n*sizeof(float));
            float *b=(float *) malloc (n*sizeof(float));
            float *sum=(float *) malloc (n*sizeof(float));
            fptr = fopen("/home/likewise-open/VITUNIVERSITY/16bce0965/lab2_1_A.txt", "w");
   if(fptr == NULL)
   {
      printf("Error!");
      exit(1);
   }
   for (i = 0; i < n; ++i)
   {
            m = rand()%1000;
        fprintf(fptr,"%f ", m);
   }

   fclose(fptr);
   printf("\nArray A Generated\n");

  fptr = fopen("/home/likewise-open/VITUNIVERSITY/16bce0965/lab2_1_B.txt", "w");
   if(fptr == NULL)
   {
      printf("Error!");
      exit(1);
   }
   for (i = 0; i < n; ++i)
   {
      m = rand()%1000;
      fprintf(fptr,"%f ", m);
   }
   fclose(fptr);
   printf("\nArray B Generated\n\n");
   // Reading arrays from files
  fptr = fopen("/home/likewise-open/VITUNIVERSITY/16bce0965/lab2_1_A.txt", "r");
  if(fptr == NULL)
  {
    printf("Error!");
    exit(1);
  }
  for (i = 0; i < n; ++i)
  {
    fscanf(fptr,"%f",&m);
     a[i]=m;
 }
 fclose(fptr);
```

```
  fptr = fopen("/home/likewise-open/VITUNIVERSITY/16bce0965/lab2_1_B.txt", "r");
  if(fptr == NULL)
  {
    printf("Error!");
    exit(1);
  }

  for (i = 0; i < n; ++i)
  {
    fscanf(fptr,"%f",&m);
    b[i]=m;
 }
 fclose(fptr);
seq_start = omp_get_wtime();
 for (i=0; i<n; i++)
  {
    sum[i] = a[i] + b[i];
  }
  seq_end = omp_get_wtime();
seq_time = seq_end - seq_start;
 float temp=10.0;
 printf("temp initialised = %f\n",temp );
 parallel_start = omp_get_wtime();
 #pragma omp parallel for lastprivate(temp)
 for (i=0; i<n; i++)
  {
    printf("\nThread %d: \n",omp_get_thread_num());
    printf("temp value = %f\n",temp );
    temp = a[i] + b[i];
    printf("temp updated = %f\n",temp );
    sum[i] = temp;
    printf("sum[%d]= %f\n",i,sum[i]);
  }
parallel_end = omp_get_wtime();
parallel_time = parallel_end - parallel_start;
printf("\ntemp outside = %f\n\n",temp );
printf("Parallel Time :%lf",parallel_time);
printf("\nSequential Time :%lf\n",seq_time);
  // Storing Output in file
    fptr = fopen("/home/likewise-open/VITUNIVERSITY/16bce0965/lab2_1_sum.txt", "w");

    for (i = 0; i < n; ++i)
    {
      fprintf(fptr,"%f ",sum[i]);
   }
   fclose(fptr);
           return (0);
 }
```

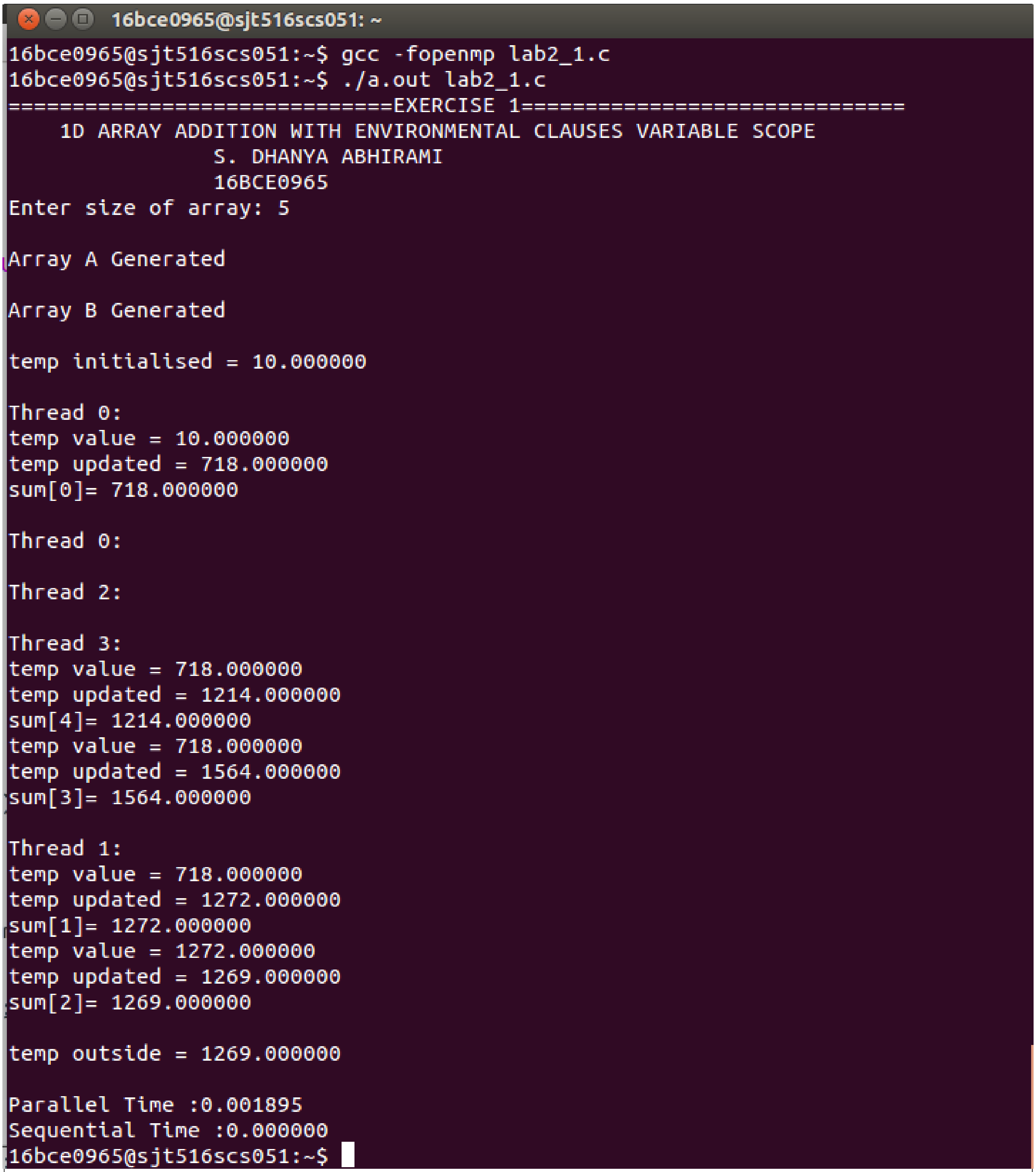


```
16bce0965@sjt516scs051:~$ gcc -fopenmp lab2_1.c
16bce0965@sjt516scs051:~$ ./a.out lab2_1.c
==============================EXERCISE 1==============================
    1D ARRAY ADDITION WITH ENVIRONMENTAL CLAUSES VARIABLE SCOPE
                S. DHANYA ABHIRAMI
                16BCE0965
Enter size of array: 5

Array A Generated

Array B Generated

temp initialised = 10.000000

Thread 0:
temp value = 0.000000
temp updated = 718.000000
sum[0]= 718.000000

Thread 0:
temp value = 718.000000
temp updated = 1272.000000
sum[1]= 1272.000000

Thread 2:
temp value = 0.000000
temp updated = 1564.000000

Thread 3:
temp value = 0.000000

Thread 1:
temp value = 0.000000
temp updated = 1269.000000
sum[2]= 1269.000000
sum[3]= 1564.000000
temp updated = 1214.000000
sum[4]= 1214.000000

temp outside = 1214.000000

Parallel Time :0.003340
Sequential Time :0.000000
16bce0965@sjt516scs051:~$ 
```

Shared

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include<time.h>
int main ()
{
	printf("==================================EXERCISE 
1==============================\n");
   printf("    1D ARRAY ADDITION WITH ENVIRONMENTAL CLAUSES VARIABLE 
SCOPE\n\t\tS. DHANYA ABHIRAMI\n\t\t16BCE0965\n");
	int i,n;
	float m;
	double parallel_start, parallel_end,seq_start,seq_end,parallel_time,seq_time;
	FILE *fptr;
```

```
        printf("Enter size of array: ");
        scanf("%d",&n);
        float *a=(float *) malloc (n*sizeof(float));
        float *b=(float *) malloc (n*sizeof(float));
        float *sum=(float *) malloc (n*sizeof(float));
        fptr = fopen("/home/likewise-open/VITUNIVERSITY/16bce0965/lab2_1_A.txt", "w");
  if(fptr == NULL)
  {
     printf("Error!");
     exit(1);
  }
  for (i = 0; i < n; ++i)
  {
        m = rand()%1000;
      fprintf(fptr,"%f ", m);
  }

  fclose(fptr);
  printf("\nArray A Generated\n");

 fptr = fopen("/home/likewise-open/VITUNIVERSITY/16bce0965/lab2_1_B.txt", "w");
  if(fptr == NULL)
  {
     printf("Error!");
     exit(1);
  }
  for (i = 0; i < n; ++i)
  {
     m = rand()%1000;
     fprintf(fptr,"%f ", m);
  }
  fclose(fptr);
  printf("\nArray B Generated\n\n");
  // Reading arrays from files
 fptr = fopen("/home/likewise-open/VITUNIVERSITY/16bce0965/lab2_1_A.txt", "r");
 if(fptr == NULL)
 {
   printf("Error!");
   exit(1);
 }
 for (i = 0; i < n; ++i)
 {
   fscanf(fptr,"%f",&m);
    a[i]=m;
}
fclose(fptr);

 fptr = fopen("/home/likewise-open/VITUNIVERSITY/16bce0965/lab2_1_B.txt", "r");
 if(fptr == NULL)
 {
   printf("Error!");
   exit(1);
  }

  for (i = 0; i < n; ++i)
  {
    fscanf(fptr,"%f",&m);
    b[i]=m;
 }
 fclose(fptr);
seq_start = omp_get_wtime();
 for (i=0; i<n; i++)
  {
    sum[i] = a[i] + b[i];
  }
  seq_end = omp_get_wtime();
seq_time = seq_end - seq_start;
 float temp=10.0;
 printf("temp initialised = %f\n",temp );
 parallel_start = omp_get_wtime();
 #pragma omp parallel for shared(temp)
 for (i=0; i<n; i++)
  {
    printf("\nThread %d: \n",omp_get_thread_num());
    printf("temp value = %f\n",temp );
    temp = a[i] + b[i];
    printf("temp updated = %f\n",temp );
    sum[i] = temp;
    printf("sum[%d]= %f\n",i,sum[i]);
  }
parallel_end = omp_get_wtime();
parallel_time = parallel_end - parallel_start;
printf("\ntemp outside = %f\n\n",temp );
printf("Parallel Time :%lf",parallel_time);
printf("\nSequential Time :%lf\n",seq_time);
  // Storing Output in file
    fptr = fopen("/home/likewise-open/VITUNIVERSITY/16bce0965/lab2_1_sum.txt", "w");

    for (i = 0; i < n; ++i)
    {
      fprintf(fptr,"%f ",sum[i]);
   }
   fclose(fptr);
            return (0);
 }
```



```
16bce0965@sjt516scs051:~$ gcc -fopenmp lab2_1.c
16bce0965@sjt516scs051:~$ ./a.out lab2_1.c
==============================EXERCISE 1==============================
    1D ARRAY ADDITION WITH ENVIRONMENTAL CLAUSES VARIABLE SCOPE
                S. DHANYA ABHIRAMI
                16BCE0965
Enter size of array: 5

Array A Generated

Array B Generated

temp initialised = 10.000000

Thread 0:
temp value = 10.000000
temp updated = 718.000000
sum[0]= 718.000000

Thread 0:

Thread 2:

Thread 3:
temp value = 718.000000
temp updated = 1214.000000
sum[4]= 1214.000000
temp value = 718.000000
temp updated = 1564.000000
sum[3]= 1564.000000

Thread 1:
temp value = 718.000000
temp updated = 1272.000000
sum[1]= 1272.000000
temp value = 1272.000000
temp updated = 1269.000000
sum[2]= 1269.000000

temp outside = 1269.000000

Parallel Time :0.001895
Sequential Time :0.000000
16bce0965@sjt516scs051:~$
```

Results

Array Length :

	Time
Private	0.002069
First Private	0.004452
Last Private	0.003340
Shared	0.001895
Sequential	0.000000

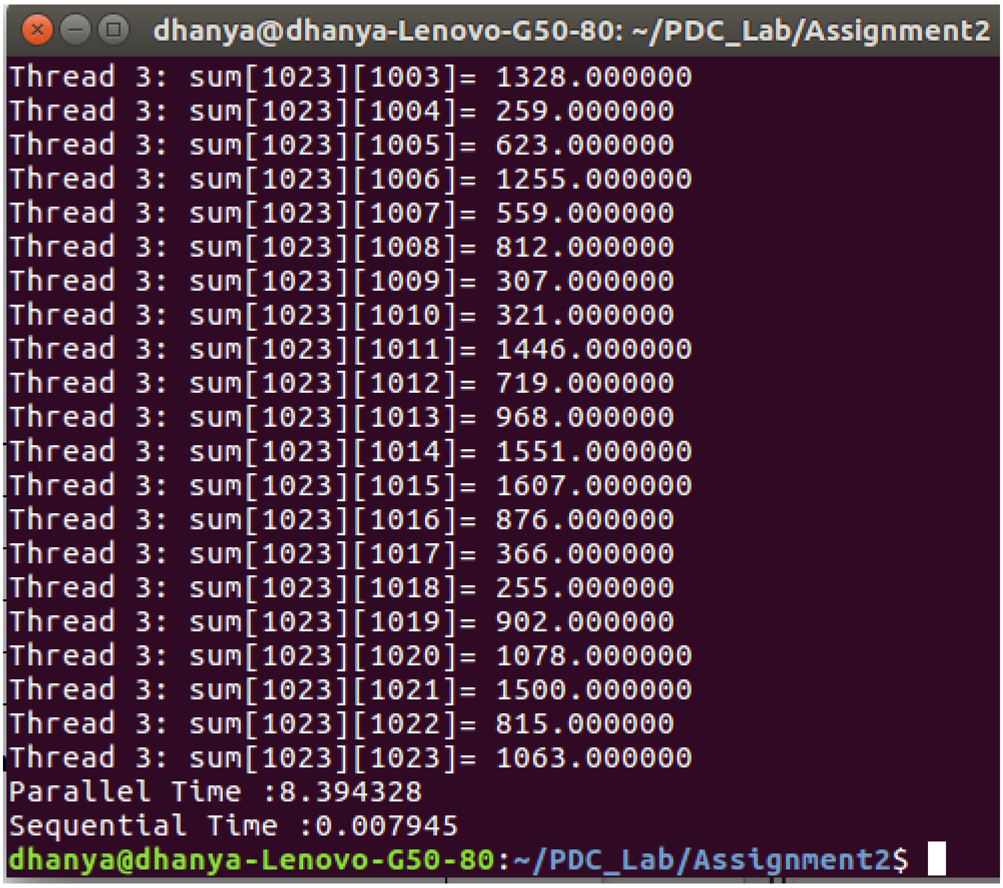
2. Write a parallel program using OpenMP for matrix addition and subtractions of above

1024 x 1024 size.
Use files concept for input and output.
Test for various scheduling clauses.
Compute serial program execution time and parallel program execution time.
Static

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include<time.h>
#define CHUNKSIZE 4

int main (int argc, char *argv[])
{
double parallel_start, parallel_end,seq_start,seq_end,parallel_time,seq_time;
  srand(time(NULL));
  printf("==========EXERCISE 2==========\n");
    printf("    MATRIX ADDITION AND SUBTRACTION\n\t\tS. DHANYA
ABHIRAMI\n\t\t16BCE0965\n");
// Generating Matrices and saving to file
  int i,j,r,c;
  FILE *fptr;
printf("Enter the number of rows: ");
  scanf("%d",&r);
  printf("\nEnter the number of columns: ");
  scanf("%d",&c);
  float **a = (float **) malloc (r*sizeof(float *));
  float **b = (float **) malloc (r*sizeof(float *));
  float **sum = (float **) malloc (r*sizeof(float *));
  float **diff = (float **) malloc (r*sizeof(float *));
  float n;
  fptr = fopen("/home/dhanya/PDC_Lab/Assignment2/lab2_2_A.txt", "w");
   if(fptr == NULL)
   {
      printf("Error!");
      exit(1);
   }
   for (i = 0; i < r; ++i)
   {
      for (j = 0; j < c; ++j)
      {
         n = rand()%1000;
         fprintf(fptr,"%f ", n);
      }
      fprintf(fptr,"\n");
   }

   fclose(fptr);
   printf("\nMatrix A Generated\n");

  fptr = fopen("/home/dhanya/PDC_Lab/Assignment2/lab2_2_B.txt", "w");
   if(fptr == NULL)
   {
    printf("Error!");
    exit(1);
  }
  for (i = 0; i < r; ++i)
  {
    for (j = 0; j < c; ++j)
    {
      n = rand()%1000;
      fprintf(fptr,"%f ", n);
    }
    fprintf(fptr,"\n");
  }
  fclose(fptr);
  printf("\nMatrix B Generated\n\n");

// Reading matrices from files
 fptr = fopen("/home/dhanya/PDC_Lab/Assignment2/lab2_2_A.txt", "r");
 if(fptr == NULL)
 {
  printf("Error!");
  exit(1);
 }

 for (i = 0; i < r; ++i)
 {
  a[i]=(float *) malloc (c*sizeof(float));
  for (j = 0; j < c; ++j)
   {
   fscanf(fptr,"%f",&n);
   a[i][j]=n;
  }
 }
 fclose(fptr);

 fptr = fopen("/home/dhanya/PDC_Lab/Assignment2/lab2_2_B.txt", "r");
 if(fptr == NULL)
 {
  printf("Error!");
  exit(1);
 }

 for (i = 0; i < r; ++i)
 {
  b[i]=(float *) malloc (c*sizeof(float));
  for (j = 0; j < c; ++j)
   {
   fscanf(fptr,"%f",&n);
   b[i][j]=n;
  }
 }
 fclose(fptr);
for (i=0; i<r; i++)
```

```
  {
sum[i]=(float *) malloc (c*sizeof(float));
diff[i]=(float *) malloc (c*sizeof(float));
}
// Computing Sum and Difference
seq_start = omp_get_wtime();
for (i=0; i<r; i++)
  {
    for(j=0;j<c;j++)
      {sum[i][j] = a[i][j] + b[i][j];
        diff[i][j] = a[i][j] - b[i][j];}
  }
seq_end = omp_get_wtime();
seq_time = seq_end - seq_start;

int nthreads, tid,chunk;
chunk = CHUNKSIZE;
parallel_start = omp_get_wtime();
#pragma omp parallel shared(a,b,nthreads,chunk) private(i,j,tid)
 {
  tid = omp_get_thread_num();
  if (tid == 0)
  {
    nthreads = omp_get_num_threads();
    printf("Number of threads = %d\n", nthreads);
  }
  printf("Thread %d starting...\n",tid);

  #pragma omp for schedule(static,chunk)
  for (i=0; i<r; i++)
  {
    for(j=0;j<c;j++)
      {sum[i][j] = a[i][j] + b[i][j];
        diff[i][j] = a[i][j] - b[i][j];
        printf("Thread %d: sum[%d][%d]= %f\n",tid,i,j,sum[i][j]);}
      }
    }
parallel_end = omp_get_wtime();
parallel_time = parallel_end - parallel_start;

// Storing Output in file
    fptr = fopen("/home/dhanya/PDC_Lab/Assignment2/lab2_2_sum.txt", "w");

    for (i = 0; i < r; ++i)
    {
      for (j = 0; j < c; ++j)
      {
       fprintf(fptr,"%f ",sum[i][j]);
     }
     fprintf(fptr, "\n" );
   }
   fclose(fptr);
```

```
// Storing Output in file
    fptr = fopen("/home/dhanya/PDC_Lab/Assignment2/lab2_2_diff.txt", "w");

    for (i = 0; i < r; ++i)
    {
      for (j = 0; j < c; ++j)
      {
       fprintf(fptr,"%f ",diff[i][j]);
     }
     fprintf(fptr, "\n" );
   }
   fclose(fptr);
printf("Parallel Time :%lf",parallel_time);
printf("\nSequential Time :%lf\n",seq_time);
   return (0);
 }
```



```
Thread 3: sum[1023][1003]= 1328.000000
Thread 3: sum[1023][1004]= 259.000000
Thread 3: sum[1023][1005]= 623.000000
Thread 3: sum[1023][1006]= 1255.000000
Thread 3: sum[1023][1007]= 559.000000
Thread 3: sum[1023][1008]= 812.000000
Thread 3: sum[1023][1009]= 307.000000
Thread 3: sum[1023][1010]= 321.000000
Thread 3: sum[1023][1011]= 1446.000000
Thread 3: sum[1023][1012]= 719.000000
Thread 3: sum[1023][1013]= 968.000000
Thread 3: sum[1023][1014]= 1551.000000
Thread 3: sum[1023][1015]= 1607.000000
Thread 3: sum[1023][1016]= 876.000000
Thread 3: sum[1023][1017]= 366.000000
Thread 3: sum[1023][1018]= 255.000000
Thread 3: sum[1023][1019]= 902.000000
Thread 3: sum[1023][1020]= 1078.000000
Thread 3: sum[1023][1021]= 1500.000000
Thread 3: sum[1023][1022]= 815.000000
Thread 3: sum[1023][1023]= 1063.000000
Parallel Time :8.394328
Sequential Time :0.007945
dhanya@dhanya-Lenovo-G50-80:~/PDC_Lab/Assignment2$
```

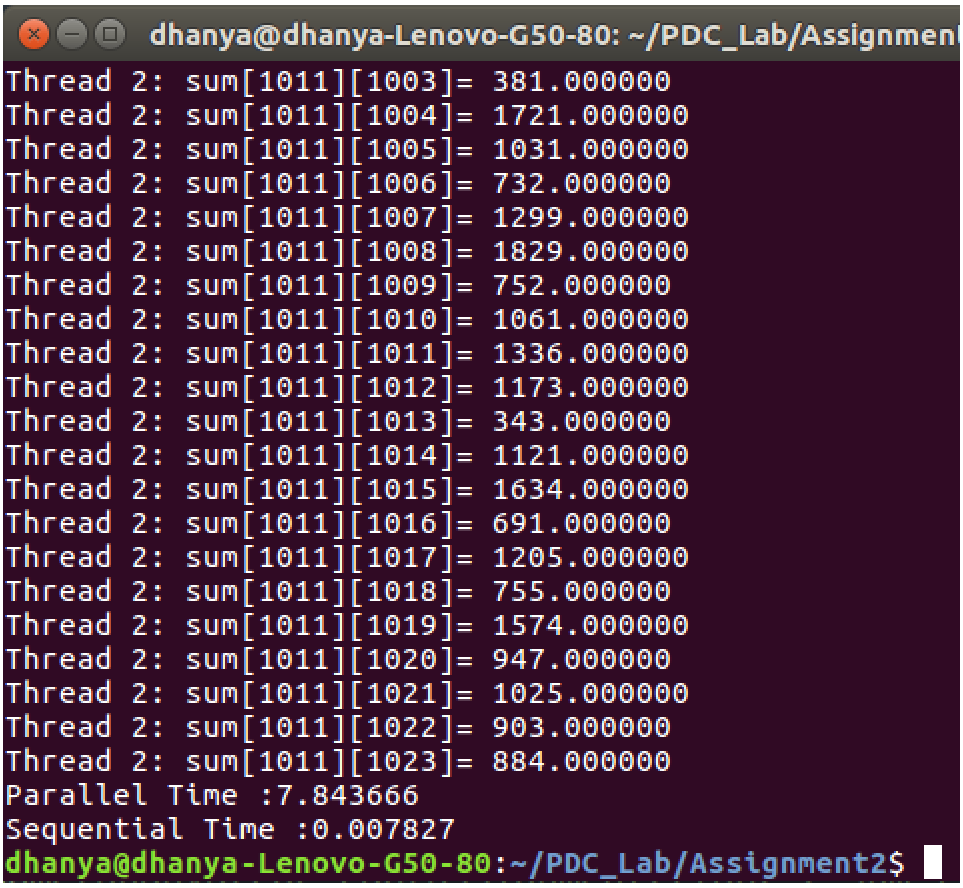
Dynamic

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include<time.h>
#define CHUNKSIZE 4

int main (int argc, char *argv[])
{
double parallel_start, parallel_end,seq_start,seq_end,parallel_time,seq_time;
  srand(time(NULL));
  printf("==========EXERCISE 2==========\n");
    printf("    MATRIX ADDITION AND SUBTRACTION\n\t\tS. DHANYA
ABHIRAMI\n\t\t16BCE0965\n");
// Generating Matrices and saving to file
  int i,j,r,c;
  FILE *fptr;
printf("Enter the number of rows: ");
  scanf("%d",&r);
  printf("\nEnter the number of columns: ");
  scanf("%d",&c);
  float **a = (float **) malloc (r*sizeof(float *));
  float **b = (float **) malloc (r*sizeof(float *));
  float **sum = (float **) malloc (r*sizeof(float *));
  float **diff = (float **) malloc (r*sizeof(float *));
  float n;
  fptr = fopen("/home/dhanya/PDC_Lab/Assignment2/lab2_2_A.txt", "w");
   if(fptr == NULL)
   {
      printf("Error!");
      exit(1);
   }
   for (i = 0; i < r; ++i)
   {
      for (j = 0; j < c; ++j)
      {
         n = rand()%1000;
         fprintf(fptr,"%f ", n);
      }
      fprintf(fptr,"\n");
   }

   fclose(fptr);
   printf("\nMatrix A Generated\n");

  fptr = fopen("/home/dhanya/PDC_Lab/Assignment2/lab2_2_B.txt", "w");
   if(fptr == NULL)
   {
      printf("Error!");
      exit(1);
   }
   for (i = 0; i < r; ++i)
   {
      for (j = 0; j < c; ++j)
      {
         n = rand()%1000;
         fprintf(fptr,"%f ", n);
      }
      fprintf(fptr,"\n");
   }
   fclose(fptr);
   printf("\nMatrix B Generated\n\n");
```

```
// Reading matrices from files
  fptr = fopen("/home/dhanya/PDC_Lab/Assignment2/lab2_2_A.txt", "r");
  if(fptr == NULL)
  {
    printf("Error!");
    exit(1);
  }

  for (i = 0; i < r; ++i)
  {
    a[i]=(float *) malloc (c*sizeof(float));
    for (j = 0; j < c; ++j)
    {
     fscanf(fptr,"%f",&n);
     a[i][j]=n;
   }
 }
 fclose(fptr);

  fptr = fopen("/home/dhanya/PDC_Lab/Assignment2/lab2_2_B.txt", "r");
  if(fptr == NULL)
  {
    printf("Error!");
    exit(1);
  }

  for (i = 0; i < r; ++i)
  {
    b[i]=(float *) malloc (c*sizeof(float));
    for (j = 0; j < c; ++j)
    {
     fscanf(fptr,"%f",&n);
     b[i][j]=n;
   }
 }
 fclose(fptr);
for (i=0; i<r; i++)
  {
sum[i]=(float *) malloc (c*sizeof(float));
diff[i]=(float *) malloc (c*sizeof(float));
}
// Computing Sum and Difference
seq_start = omp_get_wtime();
for (i=0; i<r; i++)
  {
    for(j=0;j<c;j++)
      {sum[i][j] = a[i][j] + b[i][j];
        diff[i][j] = a[i][j] - b[i][j];}
  }
seq_end = omp_get_wtime();
seq_time = seq_end - seq_start;
```

```
int nthreads, tid,chunk;
chunk = CHUNKSIZE;
parallel_start = omp_get_wtime();
#pragma omp parallel shared(a,b,nthreads,chunk) private(i,j,tid)
 {
  tid = omp_get_thread_num();
  if (tid == 0)
  {
    nthreads = omp_get_num_threads();
    printf("Number of threads = %d\n", nthreads);
  }
  printf("Thread %d starting...\n",tid);

  #pragma omp for schedule(dynamic,chunk)
  for (i=0; i<r; i++)
  {
    for(j=0;j<c;j++)
      {sum[i][j] = a[i][j] + b[i][j];
        diff[i][j] = a[i][j] - b[i][j];
        printf("Thread %d: sum[%d][%d]= %f\n",tid,i,j,sum[i][j]);}
      }
    }
parallel_end = omp_get_wtime();
parallel_time = parallel_end - parallel_start;

// Storing Output in file
    fptr = fopen("/home/dhanya/PDC_Lab/Assignment2/lab2_2_sum.txt", "w");

    for (i = 0; i < r; ++i)
    {
      for (j = 0; j < c; ++j)
      {
       fprintf(fptr,"%f ",sum[i][j]);
     }
     fprintf(fptr, "\n" );
   }
   fclose(fptr);
// Storing Output in file
    fptr = fopen("/home/dhanya/PDC_Lab/Assignment2/lab2_2_diff.txt", "w");

    for (i = 0; i < r; ++i)
    {
      for (j = 0; j < c; ++j)
      {
       fprintf(fptr,"%f ",diff[i][j]);
     }
     fprintf(fptr, "\n" );
   }
   fclose(fptr);
printf("Parallel Time :%lf",parallel_time);
printf("\nSequential Time :%lf\n",seq_time);
   return (0);
```

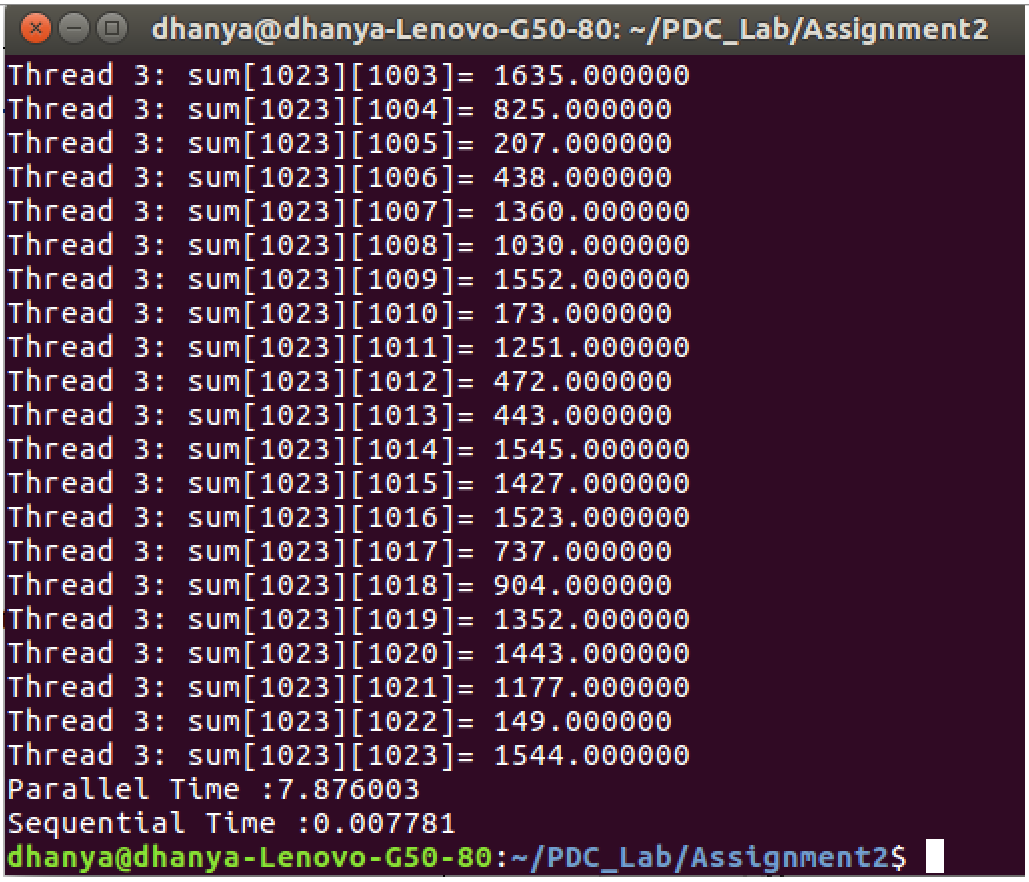
```
}
```



```
Thread 2: sum[1011][1003]= 381.000000
Thread 2: sum[1011][1004]= 1721.000000
Thread 2: sum[1011][1005]= 1031.000000
Thread 2: sum[1011][1006]= 732.000000
Thread 2: sum[1011][1007]= 1299.000000
Thread 2: sum[1011][1008]= 1829.000000
Thread 2: sum[1011][1009]= 752.000000
Thread 2: sum[1011][1010]= 1061.000000
Thread 2: sum[1011][1011]= 1336.000000
Thread 2: sum[1011][1012]= 1173.000000
Thread 2: sum[1011][1013]= 343.000000
Thread 2: sum[1011][1014]= 1121.000000
Thread 2: sum[1011][1015]= 1634.000000
Thread 2: sum[1011][1016]= 691.000000
Thread 2: sum[1011][1017]= 1205.000000
Thread 2: sum[1011][1018]= 755.000000
Thread 2: sum[1011][1019]= 1574.000000
Thread 2: sum[1011][1020]= 947.000000
Thread 2: sum[1011][1021]= 1025.000000
Thread 2: sum[1011][1022]= 903.000000
Thread 2: sum[1011][1023]= 884.000000
Parallel Time :7.843666
Sequential Time :0.007827
dhanya@dhanya-Lenovo-G50-80:~/PDC_Lab/Assignment2$
```

Guided

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include<time.h>
#define CHUNKSIZE 4

int main (int argc, char *argv[])
{
double parallel_start, parallel_end,seq_start,seq_end,parallel_time,seq_time;
  srand(time(NULL));
  printf("==========EXERCISE 2==========\n");
    printf("    MATRIX ADDITION AND SUBTRACTION\n\t\tS. DHANYA
ABHIRAMI\n\t\t16BCE0965\n");
// Generating Matrices and saving to file
  int i,j,r,c;
  FILE *fptr;
printf("Enter the number of rows: ");
  scanf("%d",&r);
  printf("\nEnter the number of columns: ");
  scanf("%d",&c);
  float **a = (float **) malloc (r*sizeof(float *));
  float **b = (float **) malloc (r*sizeof(float *));
  float **sum = (float **) malloc (r*sizeof(float *));
  float **diff = (float **) malloc (r*sizeof(float *));
  float n;
  fptr = fopen("/home/dhanya/PDC_Lab/Assignment2/lab2_2_A.txt", "w");
  if(fptr == NULL)
  {
     printf("Error!");
     exit(1);
  }
  for (i = 0; i < r; ++i)
  {
     for (j = 0; j < c; ++j)
     {
        n = rand()%1000;
        fprintf(fptr,"%f ", n);
     }
     fprintf(fptr,"\n");
  }

  fclose(fptr);
  printf("\nMatrix A Generated\n");

 fptr = fopen("/home/dhanya/PDC_Lab/Assignment2/lab2_2_B.txt", "w");
  if(fptr == NULL)
  {
     printf("Error!");
     exit(1);
  }
  for (i = 0; i < r; ++i)
  {
     for (j = 0; j < c; ++j)
     {
        n = rand()%1000;
        fprintf(fptr,"%f ", n);
     }
     fprintf(fptr,"\n");
  }
  fclose(fptr);
  printf("\nMatrix B Generated\n\n");

// Reading matrices from files
 fptr = fopen("/home/dhanya/PDC_Lab/Assignment2/lab2_2_A.txt", "r");
 if(fptr == NULL)
 {
   printf("Error!");
   exit(1);
 }

 for (i = 0; i < r; ++i)
 {
   a[i]=(float *) malloc (c*sizeof(float));
   for (j = 0; j < c; ++j)
    {
     fscanf(fptr,"%f",&n);
     a[i][j]=n;
  }
 }
 fclose(fptr);

  fptr = fopen("/home/dhanya/PDC_Lab/Assignment2/lab2_2_B.txt", "r");
  if(fptr == NULL)
  {
    printf("Error!");
    exit(1);
  }

  for (i = 0; i < r; ++i)
  {
    b[i]=(float *) malloc (c*sizeof(float));
    for (j = 0; j < c; ++j)
    {
     fscanf(fptr,"%f",&n);
     b[i][j]=n;
   }
 }
 fclose(fptr);
for (i=0; i<r; i++)
  {
sum[i]=(float *) malloc (c*sizeof(float));
diff[i]=(float *) malloc (c*sizeof(float));
}
// Computing Sum and Difference
seq_start = omp_get_wtime();
for (i=0; i<r; i++)
  {
    for(j=0;j<c;j++)
      {sum[i][j] = a[i][j] + b[i][j];
        diff[i][j] = a[i][j] - b[i][j];}
  }
seq_end = omp_get_wtime();
seq_time = seq_end - seq_start;

int nthreads, tid,chunk;
chunk = CHUNKSIZE;
parallel_start = omp_get_wtime();
#pragma omp parallel shared(a,b,nthreads,chunk) private(i,j,tid)
 {
  tid = omp_get_thread_num();
  if (tid == 0)
  {
    nthreads = omp_get_num_threads();
    printf("Number of threads = %d\n", nthreads);
  }
  printf("Thread %d starting...\n",tid);

  #pragma omp for schedule(guided,chunk)
  for (i=0; i<r; i++)
  {
    for(j=0;j<c;j++)
      {sum[i][j] = a[i][j] + b[i][j];
        diff[i][j] = a[i][j] - b[i][j];
        printf("Thread %d: sum[%d][%d]= %f\n",tid,i,j,sum[i][j]);}
      }
    }
parallel_end = omp_get_wtime();
parallel_time = parallel_end - parallel_start;

// Storing Output in file
    fptr = fopen("/home/dhanya/PDC_Lab/Assignment2/lab2_2_sum.txt", "w");

    for (i = 0; i < r; ++i)
    {
      for (j = 0; j < c; ++j)
      {
       fprintf(fptr,"%f ",sum[i][j]);
     }
     fprintf(fptr, "\n" );
   }
   fclose(fptr);
// Storing Output in file
    fptr = fopen("/home/dhanya/PDC_Lab/Assignment2/lab2_2_diff.txt", "w");

    for (i = 0; i < r; ++i)
    {
      for (j = 0; j < c; ++j)
      {
       fprintf(fptr,"%f ",diff[i][j]);
     }
     fprintf(fptr, "\n" );
   }
   fclose(fptr);
printf("Parallel Time :%lf",parallel_time);
printf("\nSequential Time :%lf\n",seq_time);
   return (0);
 }
```



```
Thread 3: sum[1023][1003]= 1635.000000
Thread 3: sum[1023][1004]= 825.000000
Thread 3: sum[1023][1005]= 207.000000
Thread 3: sum[1023][1006]= 438.000000
Thread 3: sum[1023][1007]= 1360.000000
Thread 3: sum[1023][1008]= 1030.000000
Thread 3: sum[1023][1009]= 1552.000000
Thread 3: sum[1023][1010]= 173.000000
Thread 3: sum[1023][1011]= 1251.000000
Thread 3: sum[1023][1012]= 472.000000
Thread 3: sum[1023][1013]= 443.000000
Thread 3: sum[1023][1014]= 1545.000000
Thread 3: sum[1023][1015]= 1427.000000
Thread 3: sum[1023][1016]= 1523.000000
Thread 3: sum[1023][1017]= 737.000000
Thread 3: sum[1023][1018]= 904.000000
Thread 3: sum[1023][1019]= 1352.000000
Thread 3: sum[1023][1020]= 1443.000000
Thread 3: sum[1023][1021]= 1177.000000
Thread 3: sum[1023][1022]= 149.000000
Thread 3: sum[1023][1023]= 1544.000000
Parallel Time :7.876003
Sequential Time :0.007781
dhanya@dhanya-Lenovo-G50-80:~/PDC_Lab/Assignment2$
```

Results

Matrix Dimensions : 1024*1024

	Time
Sequential	0.007945
Static	8.394328
Dynamic	7.843666
Guided	7.876003

3. Write a OpenMP program using sections for Quick Sort and Merge sort algorithms. Use files concept for input and output.

Quicksort

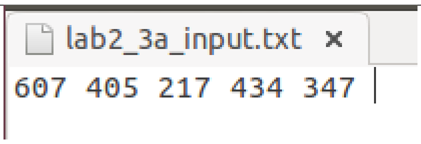
Code

```
#include<stdlib.h>
#include<stdio.h>
#include<omp.h>
#include<time.h>
int partition (int *arr, int low, int high)
{
   int temp;
   int pivot = arr[high];
   int i = (low - 1);
   int j;
   for (j = low; j <= high- 1; j++)
   {
      if (arr[j] <= pivot)
      {
         i++;
                temp = arr[i];
                arr[i ] = arr[j];
                arr[j] = temp;
      }
   }
   temp = arr[i + 1];
   arr[i + 1] = arr[high];
   arr[high] = temp;
   return (i + 1);
}

void quickSort(int *arr, int low, int high)
{
   if (low < high)
   {
        int i;
        int pi = partition(arr, low, high);
        int *arr_copy = (int*) malloc(sizeof(arr));
        for(i=0;i<sizeof(arr)/sizeof(int);i++)
        arr_copy[i]=arr[i];
   double seq_start = omp_get_wtime();
        quickSort(arr_copy, low, pi - 1);
        quickSort(arr_copy, pi + 1, high);
        printf("Sequential Time : %lf\n",omp_get_wtime()-seq_start);
        double parallel_start = omp_get_wtime();
        #pragma omp parallel
        {
                #pragma omp sections
                {
                        #pragma omp section
                        {printf("Thread ID: %d\n",omp_get_thread_num());
                quickSort(arr, low, pi - 1); }
                        #pragma omp section
                        {printf("Thread ID: %d\n",omp_get_thread_num());
                quickSort(arr, pi + 1, high); }
                }
        }
        printf("Parallel Time: %lf\n",omp_get_wtime()-parallel_start);
   }
}


int main()
{
   srand(time(NULL));
printf("=========EXERCISE 3a=========\n");
   printf("    QUICKSORT\n\t\tS. DHANYA ABHIRAMI\n\t\t16BCE0965\n");
   int i,n,m;
   printf("Enter array size: ");
```

```
    scanf("%d",&n);
    int *arr = (int*) malloc(sizeof(int)*n);
    FILE *fptr;
    fptr = fopen("/home/likewise-open/VITUNIVERSITY/16bce0965/lab2_3a_input.txt", "w");
    if(fptr == NULL)
    {
    printf("Error!");
    exit(1);
    }
    for (i = 0; i < n; ++i)
    {
    m = rand()%1000;
    fprintf(fptr,"%d ", m);
    }
    fclose(fptr);
   printf("\nArray Generated\n");
   fptr = fopen("/home/likewise-open/VITUNIVERSITY/16bce0965/lab2_3a_input.txt", "r");
  if(fptr == NULL)
  {
    printf("Error!");
    exit(1);
  }
  for (i = 0; i < n; ++i)
  {
    fscanf(fptr,"%d",&m);
     arr[i]=m;
 }
 fclose(fptr);
    quickSort(arr, 0, n-1);

    printf("Sorted array: ");
    for (i=0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
    fptr = fopen("/home/likewise-open/VITUNIVERSITY/16bce0965/lab2_3a_output.txt", "w");
    if(fptr == NULL)
    {
    printf("Error!");
    exit(1);
    }
    for (i = 0; i < n; ++i)
    {
    fprintf(fptr,"%d ", arr[i]);
    }
    fclose(fptr);
    return 0;
}
```

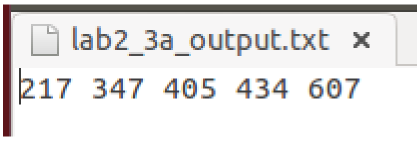
Output

Unsorted

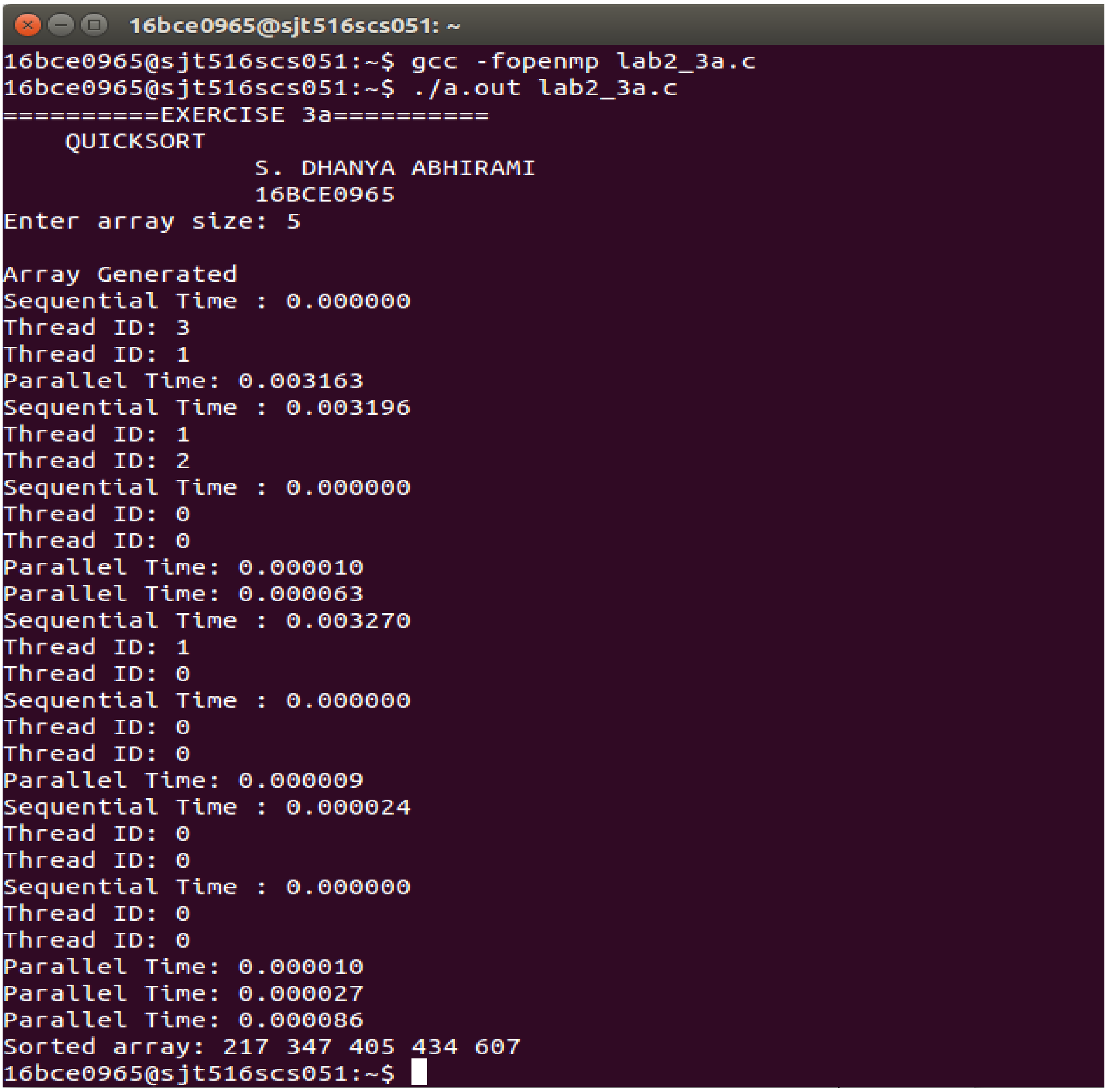


```
607 405 217 434 347
```

Sorted



```
217 347 405 434 607
```



```
16bce0965@sjt516scs051:~$ gcc -fopenmp lab2_3a.c
16bce0965@sjt516scs051:~$ ./a.out lab2_3a.c
==========EXERCISE 3a===========
    QUICKSORT
                S. DHANYA ABHIRAMI
                16BCE0965
Enter array size: 5

Array Generated
Sequential Time : 0.000000
Thread ID: 3
Thread ID: 1
Parallel Time: 0.003163
Sequential Time : 0.003196
Thread ID: 1
Thread ID: 2
Sequential Time : 0.000000
Thread ID: 0
Thread ID: 0
Parallel Time: 0.000010
Parallel Time: 0.000063
Sequential Time : 0.003270
Thread ID: 1
Thread ID: 0
Sequential Time : 0.000000
Thread ID: 0
Thread ID: 0
Parallel Time: 0.000009
Sequential Time : 0.000024
Thread ID: 0
Thread ID: 0
Sequential Time : 0.000000
Thread ID: 0
Thread ID: 0
Parallel Time: 0.000010
Parallel Time: 0.000027
Parallel Time: 0.000086
Sorted array: 217 347 405 434 607
16bce0965@sjt516scs051:~$
```

Merge Sort
Code

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

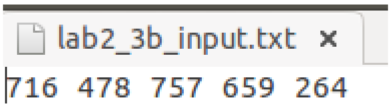
```
#include<time.h>

void merge(int *arr, int lower, int mid, int upper)
{
    int i, j, k;
    int n1 = mid - lower + 1;
    int n2 =  upper - mid;

    int *L = (int*) malloc(sizeof(int)*n1);
    int *R = (int*) malloc(sizeof(int)*n2);

    for (i = 0; i < n1; i++)
        L[i] = arr[lower + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[mid + 1+ j];

    i = 0;
    j = 0;
    k = lower;
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2)
    {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(int *arr, int lower, int upper)
{
    if (lower < upper)
```

```
   {
         int i;
         int mid = lower+(upper-lower)/2;
         int *arr_copy = (int*) malloc(sizeof(arr));
         for(i=0;i<sizeof(arr)/sizeof(int);i++)
         arr_copy[i]=arr[i];
   double seq_start = omp_get_wtime();
         mergeSort(arr_copy, lower, mid);
         mergeSort(arr_copy, mid+1, upper);
         printf("Sequential Time : %lf\n",omp_get_wtime()-seq_start);
         double parallel_start = omp_get_wtime();
         #pragma omp parallel
         {
                  #pragma omp sections
                  {
                           #pragma omp section
                           {printf("Thread ID: %d\n",omp_get_thread_num());
                           mergeSort(arr, lower, mid);
                           }
                           #pragma omp section
                           {printf("Thread ID: %d\n",omp_get_thread_num());
                           mergeSort(arr, mid+1, upper);
                           }
                  }
         }
         printf("Parallel Time: %lf\n",omp_get_wtime()-parallel_start);
      merge(arr, lower, mid, upper);
   }
}


int main()
{
   srand(time(NULL));
   printf("==========EXERCISE 3b==========\n");
   printf("    MERGESORT\n\t\tS. DHANYA ABHIRAMI\n\t\t16BCE0965\n");
   int i,n,m;
   printf("Enter array size: ");
   scanf("%d",&n);
   int *arr = (int*) malloc(sizeof(int)*n);
   FILE *fptr;
   fptr = fopen("/home/likewise-open/VITUNIVERSITY/16bce0965/lab2_3b_input.txt", "w");
   if(fptr == NULL)
   {
   printf("Error!");
   exit(1);
   }
   for (i = 0; i < n; ++i)
   {
   m = rand()%1000;arr[i]=m;
   fprintf(fptr,"%d ", m);
   }
```

```
    fclose(fptr);
    fptr = fopen("/home/likewise-open/VITUNIVERSITY/16bce0965/lab2_3b_input.txt", "r");
    if(fptr == NULL)
    {
    printf("Error!");
    exit(1);
    }
    for (i = 0; i < n; ++i)
    {
    fscanf(fptr,"%d ", &m);
    arr[i]=m;
    }
    fclose(fptr);
    mergeSort(arr, 0, n - 1);

    printf("Sorted array: ");
    for (i=0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
    fptr = fopen("/home/likewise-open/VITUNIVERSITY/16bce0965/lab2_3b_output.txt", "w");
    if(fptr == NULL)
    {
    printf("Error!");
    exit(1);
    }
    for (i = 0; i < n; ++i)
    {
    fprintf(fptr,"%d ", arr[i]);
    }
    fclose(fptr);
    return 0;
}
```

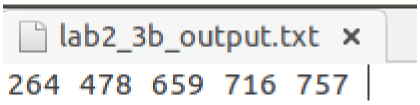
Output

Unsorted

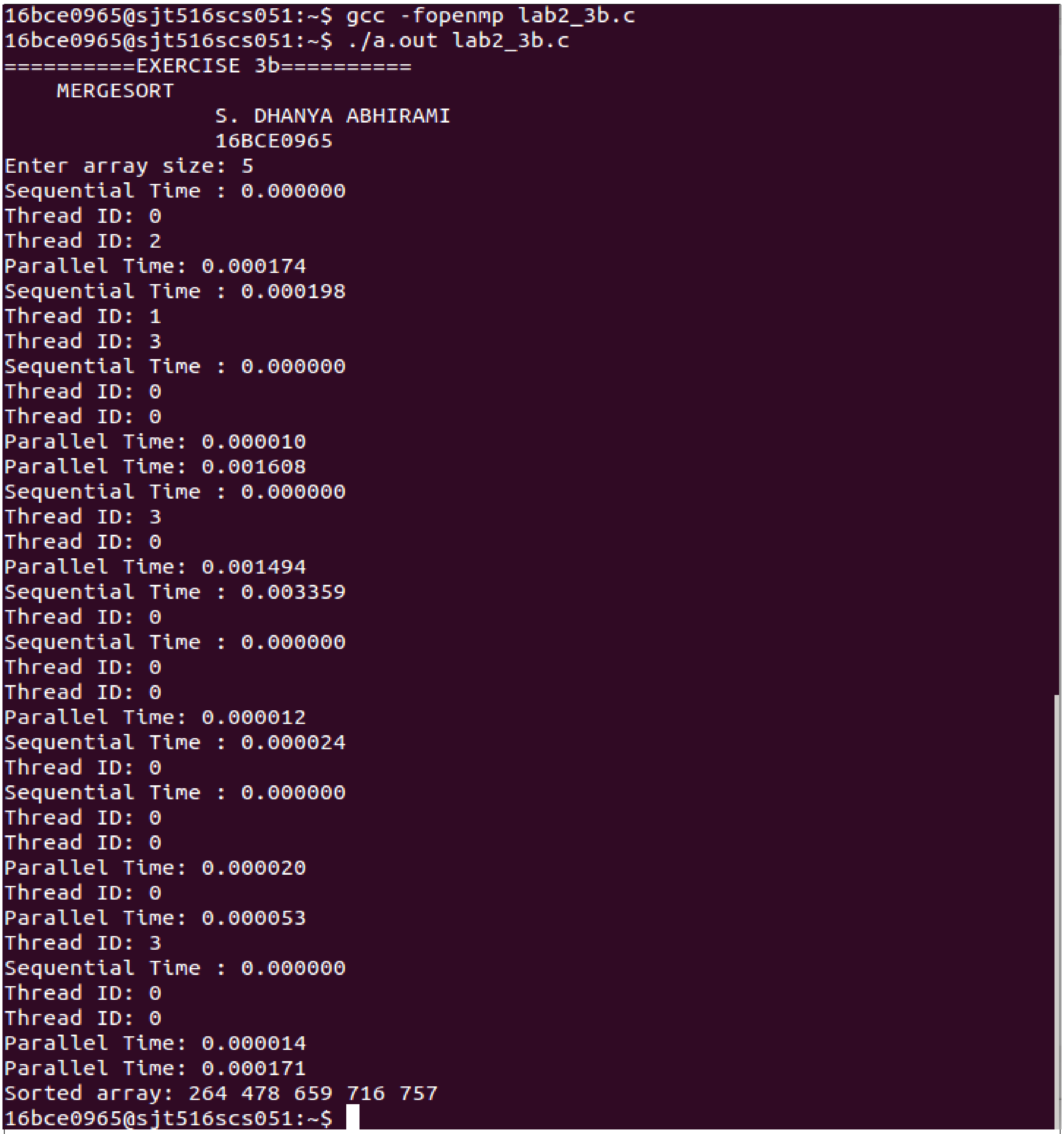


```
716 478 757 659 264
```

Sorted



```
264 478 659 716 757
```



```
16bce0965@sjt516scs051:~$ gcc -fopenmp lab2_3b.c
16bce0965@sjt516scs051:~$ ./a.out lab2_3b.c
==========EXERCISE 3b==========
    MERGESORT
                S. DHANYA ABHIRAMI
                16BCE0965
Enter array size: 5
Sequential Time : 0.000000
Thread ID: 0
Thread ID: 2
Parallel Time: 0.000174
Sequential Time : 0.000198
Thread ID: 1
Thread ID: 3
Sequential Time : 0.000000
Thread ID: 0
Thread ID: 0
Parallel Time: 0.000010
Parallel Time: 0.001608
Sequential Time : 0.000000
Thread ID: 3
Thread ID: 0
Parallel Time: 0.001494
Sequential Time : 0.003359
Thread ID: 0
Sequential Time : 0.000000
Thread ID: 0
Thread ID: 0
Parallel Time: 0.000012
Sequential Time : 0.000024
Thread ID: 0
Sequential Time : 0.000000
Thread ID: 0
Thread ID: 0
Parallel Time: 0.000020
Thread ID: 0
Parallel Time: 0.000053
Thread ID: 3
Sequential Time : 0.000000
Thread ID: 0
Thread ID: 0
Parallel Time: 0.000014
Parallel Time: 0.000171
Sorted array: 264 478 659 716 757
16bce0965@sjt516scs051:~$
```

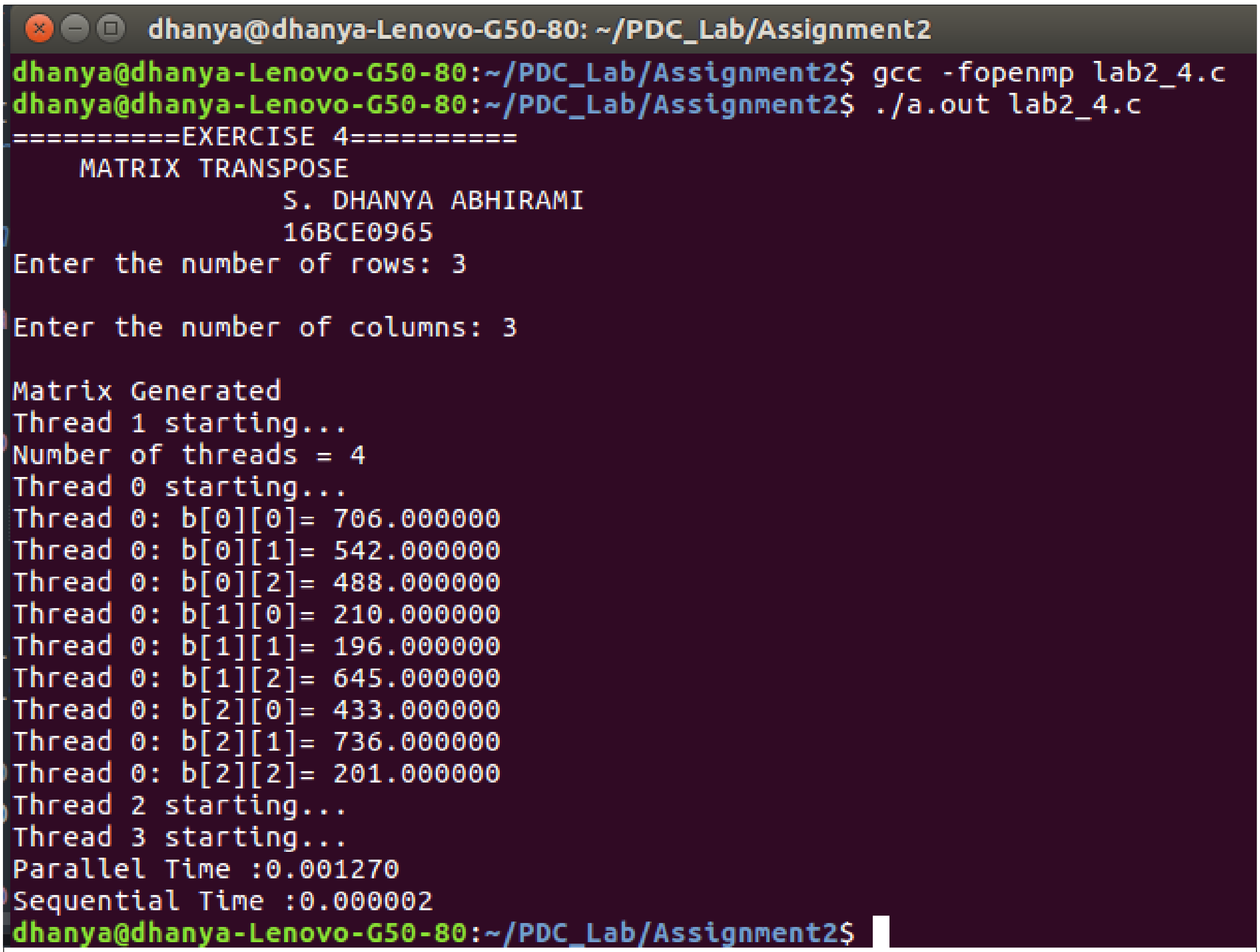
4. To perform the transpose of the matrix using parallel for loop constructs with different scheduling clause and compare parallel and serial execution time.

Static

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include<time.h>
#define CHUNKSIZE 4

int main (int argc, char *argv[])
{
double parallel_start, parallel_end,seq_start,seq_end,parallel_time,seq_time;
srand(time(NULL));
  printf("==========EXERCISE 4==========\n");
```

```
    printf("    MATRIX TRANSPOSE\n\t\tS. DHANYA ABHIRAMI\n\t\t16BCE0965\n");
// Generating Matrix and saving to file
  int i,j,r,c;
  FILE *fptr;
printf("Enter the number of rows: ");
  scanf("%d",&r);
  printf("\nEnter the number of columns: ");
  scanf("%d",&c);
  float **a = (float **) malloc (r*sizeof(float *));
  float **b = (float **) malloc (c*sizeof(float *));
  float n;
  fptr = fopen("/home/likewise-open/VITUNIVERSITY/16bce0965/lab2_4_input.txt", "w");
   if(fptr == NULL)
   {
      printf("Error!");
      exit(1);
   }
   for (i = 0; i < r; ++i)
   {
      for (j = 0; j < c; ++j)
      {
         n = rand()%1000;
         fprintf(fptr,"%f ", n);
      }
      fprintf(fptr,"\n");
   }

   fclose(fptr);
   printf("\nMatrix Generated\n");

// Reading matrix from file
  fptr = fopen("/home/likewise-open/VITUNIVERSITY/16bce0965/lab2_4_input.txt", "r");
  if(fptr == NULL)
  {
    printf("Error!");
    exit(1);
  }

  for (i = 0; i < r; ++i)
  {
    a[i]=(float *) malloc (c*sizeof(float));
    for (j = 0; j < c; ++j)
    {
     fscanf(fptr,"%f",&n);
     a[i][j]=n;
    }
 }

 fclose(fptr);

// Computing Transpose
seq_start = omp_get_wtime();
```

```
for (i=0; i<c; i++)
  {
    b[i]=(float *) malloc (r*sizeof(float));
    for(j=0;j<r;j++)
      {b[i][j] = a[j][i];}
  }
seq_end = omp_get_wtime();
seq_time = seq_end - seq_start;

int nthreads, tid,chunk;
chunk = CHUNKSIZE;
parallel_start = omp_get_wtime();
#pragma omp parallel shared(a,b,nthreads,chunk) private(i,j,tid)
 {
  tid = omp_get_thread_num();
  if (tid == 0)
  {
    nthreads = omp_get_num_threads();
    printf("Number of threads = %d\n", nthreads);
  }
  printf("Thread %d starting...\n",tid);

  #pragma omp for schedule(static,chunk)
  for (i=0; i<c; i++)
  {
    for(j=0;j<r;j++)
      {b[i][j] = a[j][i];
        printf("Thread %d: b[%d][%d]= %f\n",tid,i,j,b[i][j]);}
      }
    }
parallel_end = omp_get_wtime();
parallel_time = parallel_end - parallel_start;

// Storing Output in file
    fptr = fopen("/home/likewise-open/VITUNIVERSITY/16bce0965/lab2_4_output.txt", "w");

    for (i = 0; i < c; ++i)
    {
      for (j = 0; j < r; ++j)
      {
       fprintf(fptr,"%f ",b[i][j]);
     }
     fprintf(fptr, "\n" );
   }
   fclose(fptr);
  printf("Parallel Time :%lf",parallel_time);
printf("\nSequential Time :%lf\n",seq_time);
   return (0);
 }
```



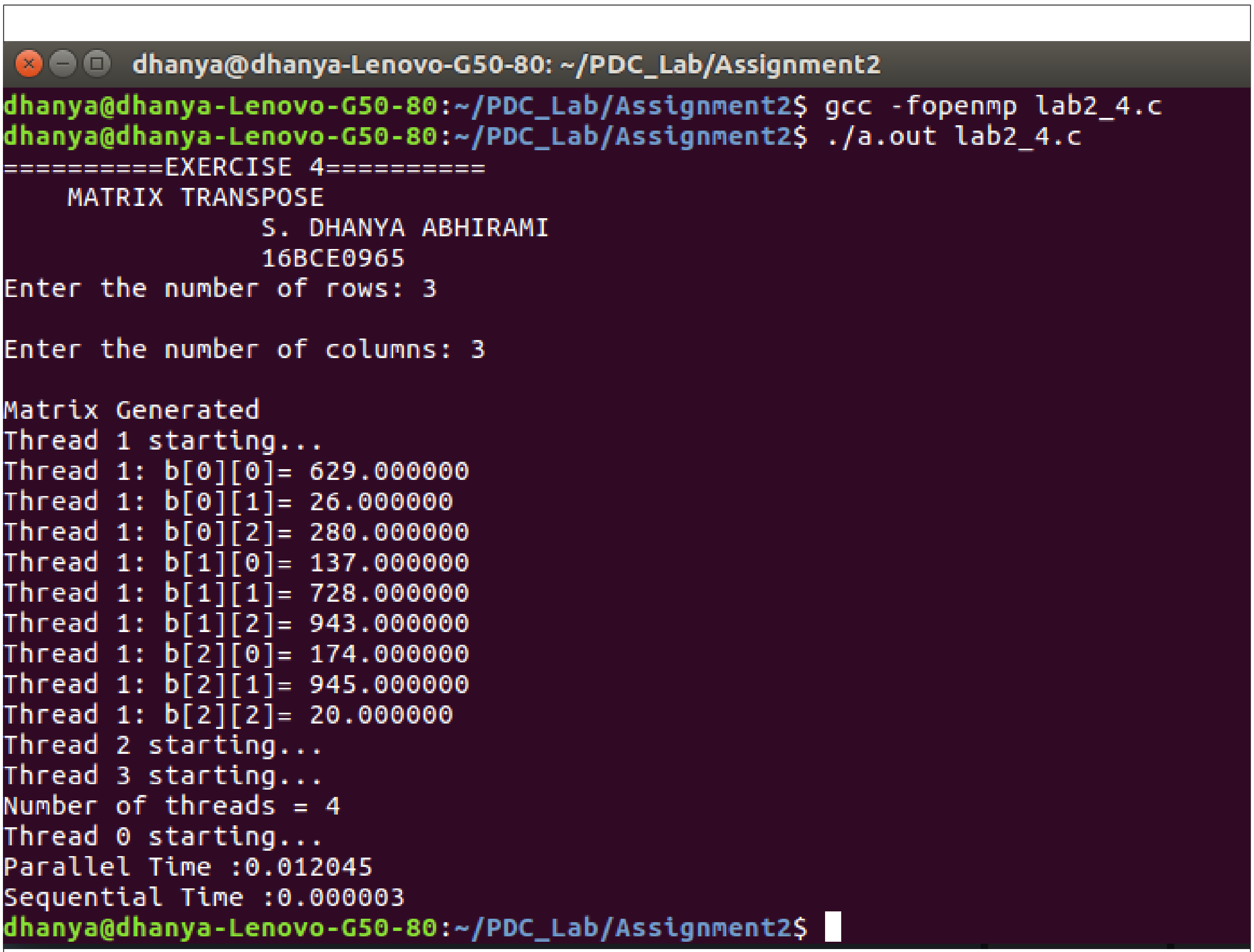
```
dhanya@dhanya-Lenovo-G50-80:~/PDC_Lab/Assignment2$ gcc -fopenmp lab2_4.c
dhanya@dhanya-Lenovo-G50-80:~/PDC_Lab/Assignment2$ ./a.out lab2_4.c
==========EXERCISE 4==========
    MATRIX TRANSPOSE
                S. DHANYA ABHIRAMI
                16BCE0965
Enter the number of rows: 3

Enter the number of columns: 3

Matrix Generated
Thread 1 starting...
Number of threads = 4
Thread 0 starting...
Thread 0: b[0][0]= 706.000000
Thread 0: b[0][1]= 542.000000
Thread 0: b[0][2]= 488.000000
Thread 0: b[1][0]= 210.000000
Thread 0: b[1][1]= 196.000000
Thread 0: b[1][2]= 645.000000
Thread 0: b[2][0]= 433.000000
Thread 0: b[2][1]= 736.000000
Thread 0: b[2][2]= 201.000000
Thread 2 starting...
Thread 3 starting...
Parallel Time :0.001270
Sequential Time :0.000002
dhanya@dhanya-Lenovo-G50-80:~/PDC_Lab/Assignment2$
```

Dynamic

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include<time.h>
#define CHUNKSIZE 4

int main (int argc, char *argv[])
{
double parallel_start, parallel_end,seq_start,seq_end,parallel_time,seq_time;
srand(time(NULL));
  printf("==========EXERCISE 4==========\n");
    printf("    MATRIX TRANSPOSE\n\t\tS. DHANYA ABHIRAMI\n\t\t16BCE0965\n");
// Generating Matrix and saving to file
  int i,j,r,c;
  FILE *fptr;
printf("Enter the number of rows: ");
  scanf("%d",&r);
  printf("\nEnter the number of columns: ");
  scanf("%d",&c);
  float **a = (float **) malloc (r*sizeof(float *));
  float **b = (float **) malloc (c*sizeof(float *));
  float n;
  fptr = fopen("/home/likewise-open/VITUNIVERSITY/16bce0965/lab2_4_input.txt", "w");
```

```
   if(fptr == NULL)
   {
      printf("Error!");
      exit(1);
   }
   for (i = 0; i < r; ++i)
   {
      for (j = 0; j < c; ++j)
      {
         n = rand()%1000;
         fprintf(fptr,"%f ", n);
      }
      fprintf(fptr,"\n");
   }

   fclose(fptr);
   printf("\nMatrix Generated\n");

// Reading matrix from file
  fptr = fopen("/home/likewise-open/VITUNIVERSITY/16bce0965/lab2_4_input.txt", "r");
  if(fptr == NULL)
  {
    printf("Error!");
    exit(1);
  }

  for (i = 0; i < r; ++i)
  {
    a[i]=(float *) malloc (c*sizeof(float));
    for (j = 0; j < c; ++j)
    {
     fscanf(fptr,"%f",&n);
     a[i][j]=n;
   }
 }

 fclose(fptr);

// Computing Transpose
seq_start = omp_get_wtime();
for (i=0; i<c; i++)
  {
    b[i]=(float *) malloc (r*sizeof(float));
    for(j=0;j<r;j++)
      {b[i][j] = a[j][i];}
  }
seq_end = omp_get_wtime();
seq_time = seq_end - seq_start;

int nthreads, tid,chunk;
chunk = CHUNKSIZE;
parallel_start = omp_get_wtime();
```

```
#pragma omp parallel shared(a,b,nthreads,chunk) private(i,j,tid)
 {
  tid = omp_get_thread_num();
  if (tid == 0)
  {
    nthreads = omp_get_num_threads();
    printf("Number of threads = %d\n", nthreads);
  }
  printf("Thread %d starting...\n",tid);

  #pragma omp for schedule(dynamic,chunk)
  for (i=0; i<c; i++)
  {
    for(j=0;j<r;j++)
      {b[i][j] = a[j][i];
        printf("Thread %d: b[%d][%d]= %f\n",tid,i,j,b[i][j]);}
      }
    }
parallel_end = omp_get_wtime();
parallel_time = parallel_end - parallel_start;

// Storing Output in file
    fptr = fopen("/home/likewise-open/VITUNIVERSITY/16bce0965/lab2_4_output.txt", "w");

    for (i = 0; i < c; ++i)
    {
      for (j = 0; j < r; ++j)
      {
       fprintf(fptr,"%f ",b[i][j]);
     }
     fprintf(fptr, "\n" );
   }
   fclose(fptr);
  printf("Parallel Time :%lf",parallel_time);
printf("\nSequential Time :%lf\n",seq_time);
   return (0);
 }
```

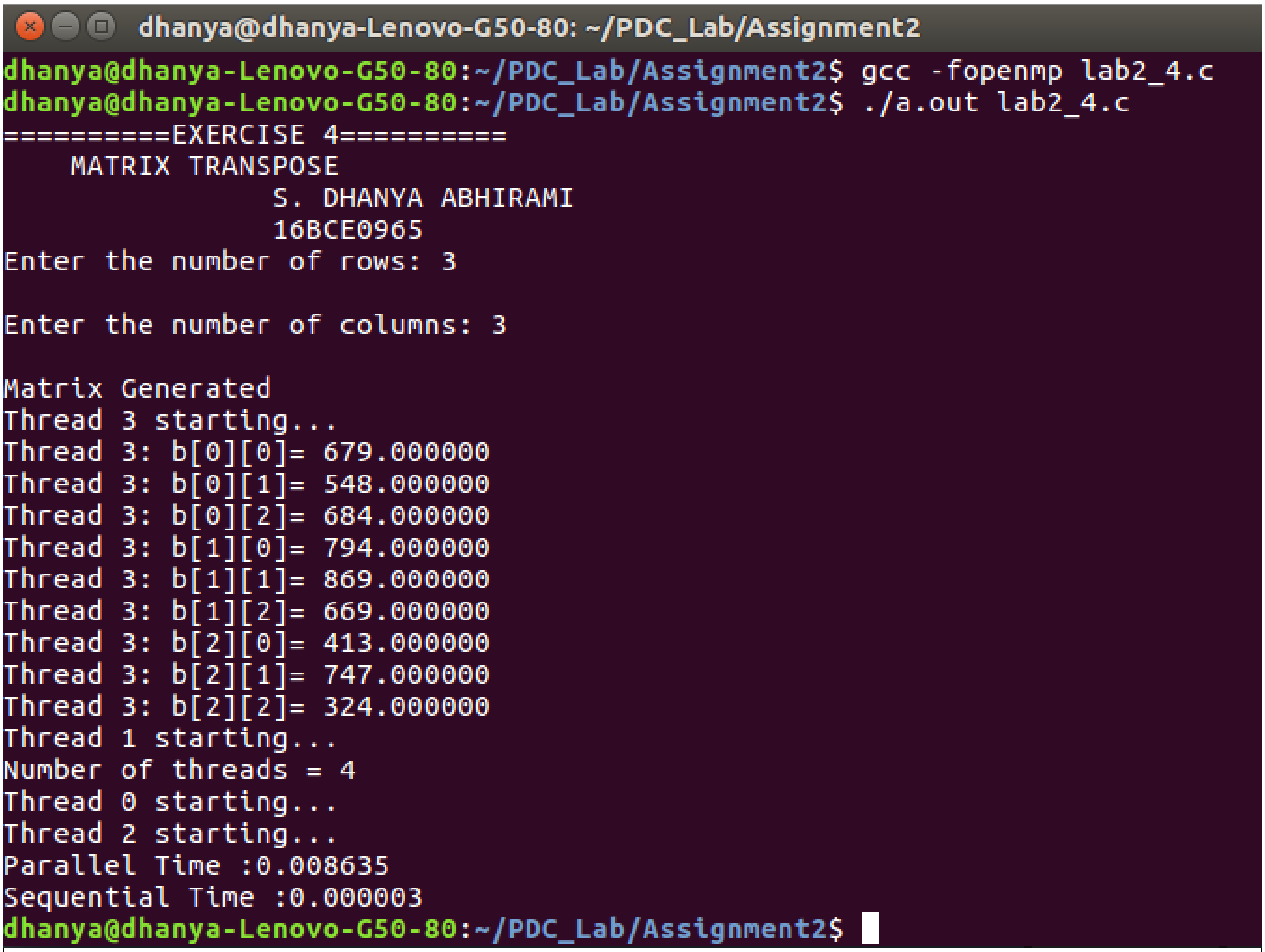


```
dhanya@dhanya-Lenovo-G50-80: ~/PDC_Lab/Assignment2
dhanya@dhanya-Lenovo-G50-80:~/PDC_Lab/Assignment2$ gcc -fopenmp lab2_4.c
dhanya@dhanya-Lenovo-G50-80:~/PDC_Lab/Assignment2$ ./a.out lab2_4.c
==========EXERCISE 4==========
    MATRIX TRANSPOSE
                S. DHANYA ABHIRAMI
                16BCE0965
Enter the number of rows: 3

Enter the number of columns: 3

Matrix Generated
Thread 1 starting...
Thread 1: b[0][0]= 629.000000
Thread 1: b[0][1]= 26.000000
Thread 1: b[0][2]= 280.000000
Thread 1: b[1][0]= 137.000000
Thread 1: b[1][1]= 728.000000
Thread 1: b[1][2]= 943.000000
Thread 1: b[2][0]= 174.000000
Thread 1: b[2][1]= 945.000000
Thread 1: b[2][2]= 20.000000
Thread 2 starting...
Thread 3 starting...
Number of threads = 4
Thread 0 starting...
Parallel Time :0.012045
Sequential Time :0.000003
dhanya@dhanya-Lenovo-G50-80:~/PDC_Lab/Assignment2$ 
```

Guided

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include<time.h>
#define CHUNKSIZE 4

int main (int argc, char *argv[])
{
double parallel_start, parallel_end,seq_start,seq_end,parallel_time,seq_time;
srand(time(NULL));
  printf("==========EXERCISE 4==========\n");
    printf("    MATRIX TRANSPOSE\n\t\tS. DHANYA ABHIRAMI\n\t\t16BCE0965\n");
// Generating Matrix and saving to file
  int i,j,r,c;
  FILE *fptr;
printf("Enter the number of rows: ");
  scanf("%d",&r);
  printf("\nEnter the number of columns: ");
  scanf("%d",&c);
  float **a = (float **) malloc (r*sizeof(float *));
  float **b = (float **) malloc (c*sizeof(float *));
  float n;
  fptr = fopen("/home/likewise-open/VITUNIVERSITY/16bce0965/lab2_4_input.txt", "w");
  if(fptr == NULL)
  {
    printf("Error!");
    exit(1);
  }
  for (i = 0; i < r; ++i)
  {
    for (j = 0; j < c; ++j)
    {
      n = rand()%1000;
      fprintf(fptr,"%f ", n);
    }
    fprintf(fptr,"\n");
  }

  fclose(fptr);
  printf("\nMatrix Generated\n");

// Reading matrix from file
  fptr = fopen("/home/likewise-open/VITUNIVERSITY/16bce0965/lab2_4_input.txt", "r");
  if(fptr == NULL)
  {
   printf("Error!");
   exit(1);
  }

  for (i = 0; i < r; ++i)
  {
   a[i]=(float *) malloc (c*sizeof(float));
   for (j = 0; j < c; ++j)
   {
    fscanf(fptr,"%f",&n);
    a[i][j]=n;
   }
 }

 fclose(fptr);

// Computing Transpose
seq_start = omp_get_wtime();
for (i=0; i<c; i++)
  {
   b[i]=(float *) malloc (r*sizeof(float));
   for(j=0;j<r;j++)
     {b[i][j] = a[j][i];}
  }
seq_end = omp_get_wtime();
seq_time = seq_end - seq_start;

int nthreads, tid,chunk;
chunk = CHUNKSIZE;
parallel_start = omp_get_wtime();
```

```
#pragma omp parallel shared(a,b,nthreads,chunk) private(i,j,tid)
 {
  tid = omp_get_thread_num();
  if (tid == 0)
  {
    nthreads = omp_get_num_threads();
    printf("Number of threads = %d\n", nthreads);
  }
  printf("Thread %d starting...\n",tid);

  #pragma omp for schedule(guided,chunk)
  for (i=0; i<c; i++)
  {
    for(j=0;j<r;j++)
      {b[i][j] = a[j][i];
       printf("Thread %d: b[%d][%d]= %f\n",tid,i,j,b[i][j]);}
      }
    }
parallel_end = omp_get_wtime();
parallel_time = parallel_end - parallel_start;

// Storing Output in file
    fptr = fopen("/home/likewise-open/VITUNIVERSITY/16bce0965/lab2_4_output.txt", "w");

    for (i = 0; i < c; ++i)
    {
      for (j = 0; j < r; ++j)
      {
       fprintf(fptr,"%f ",b[i][j]);
     }
     fprintf(fptr, "\n" );
   }
   fclose(fptr);
  printf("Parallel Time :%lf",parallel_time);
printf("\nSequential Time :%lf\n",seq_time);
   return (0);
 }
```



```
dhanya@dhanya-Lenovo-G50-80:~/PDC_Lab/Assignment2$ gcc -fopenmp lab2_4.c
dhanya@dhanya-Lenovo-G50-80:~/PDC_Lab/Assignment2$ ./a.out lab2_4.c
==========EXERCISE 4==========
    MATRIX TRANSPOSE
                S. DHANYA ABHIRAMI
                16BCE0965
Enter the number of rows: 3

Enter the number of columns: 3

Matrix Generated
Thread 3 starting...
Thread 3: b[0][0]= 679.000000
Thread 3: b[0][1]= 548.000000
Thread 3: b[0][2]= 684.000000
Thread 3: b[1][0]= 794.000000
Thread 3: b[1][1]= 869.000000
Thread 3: b[1][2]= 669.000000
Thread 3: b[2][0]= 413.000000
Thread 3: b[2][1]= 747.000000
Thread 3: b[2][2]= 324.000000
Thread 1 starting...
Number of threads = 4
Thread 0 starting...
Thread 2 starting...
Parallel Time :0.008635
Sequential Time :0.000003
dhanya@dhanya-Lenovo-G50-80:~/PDC_Lab/Assignment2$
```

Results

Matrix Dimensions :

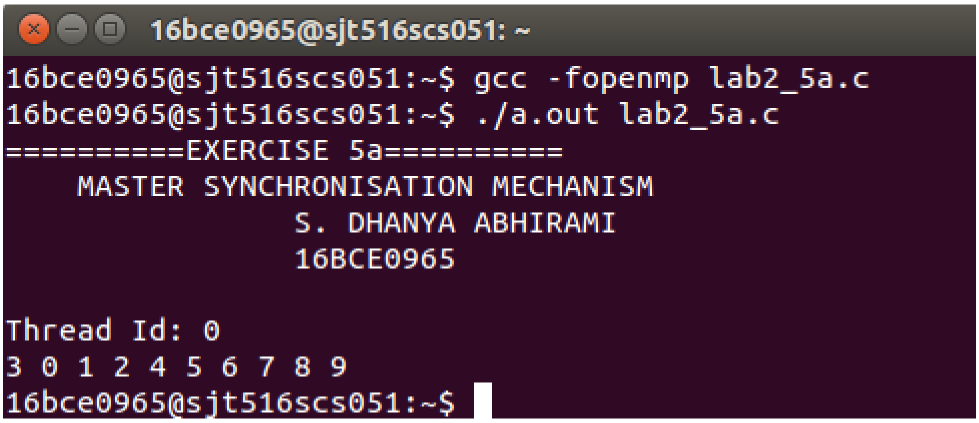
	Time
Sequential	0.000003
Static	0.001270
Dynamic	0.012045
Guided	0.008635

5. Demonstrate an example by using Single, Master, Barrier and critical constructs of OpenMP.

Master

```
#include <omp.h>
#include <stdio.h>
int main ()
{
int i,tid;
printf("==========EXERCISE 5a==========\n");
printf("    MASTER SYNCHRONISATION MECHANISM\n\t\tS. DHANYA ABHIRAMI\n\t\t16BCE0965\n");
#pragma omp parallel
{
        #pragma omp master
        {tid = omp_get_thread_num();
```

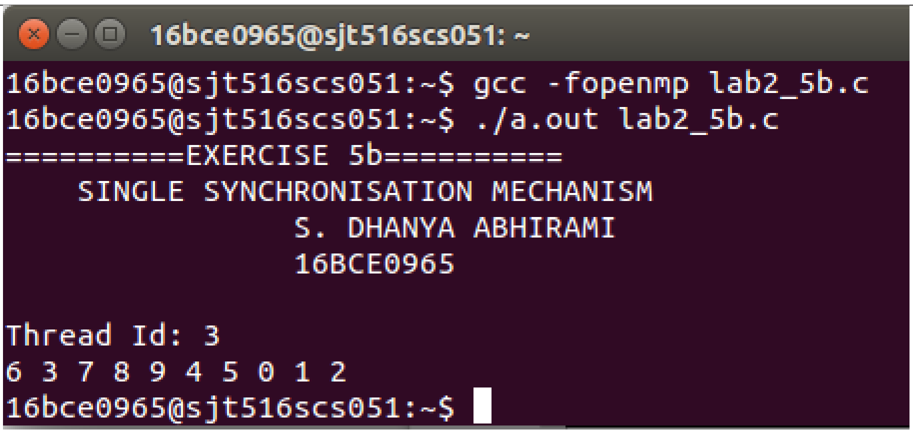
```
	printf("\nThread Id: %d\n",tid);}
	#pragma omp for
	for(i=0;i<10;i++)
	printf("%d ",i);
}
printf("\n");
return (0);
}
```



```
16bce0965@sjt516scs051:~$ gcc -fopenmp lab2_5a.c
16bce0965@sjt516scs051:~$ ./a.out lab2_5a.c
==========EXERCISE 5a==========
    MASTER SYNCHRONISATION MECHANISM
                S. DHANYA ABHIRAMI
                16BCE0965

Thread Id: 0
3 0 1 2 4 5 6 7 8 9
16bce0965@sjt516scs051:~$ 
```

Single

```
#include <omp.h>
#include <stdio.h>
int main ()
{
int i,tid;
printf("==========EXERCISE 5b==========\n");
printf("    SINGLE SYNCHRONISATION MECHANISM\n\t\tS. DHANYA ABHIRAMI\n\t\t16BCE0965\n");
#pragma omp parallel
{
	#pragma omp single
	{tid = omp_get_thread_num();
	printf("\nThread Id: %d\n",tid);}
	#pragma omp for
	for(i=0;i<10;i++)
	printf("%d ",i);
}
printf("\n");
return (0);
}
```

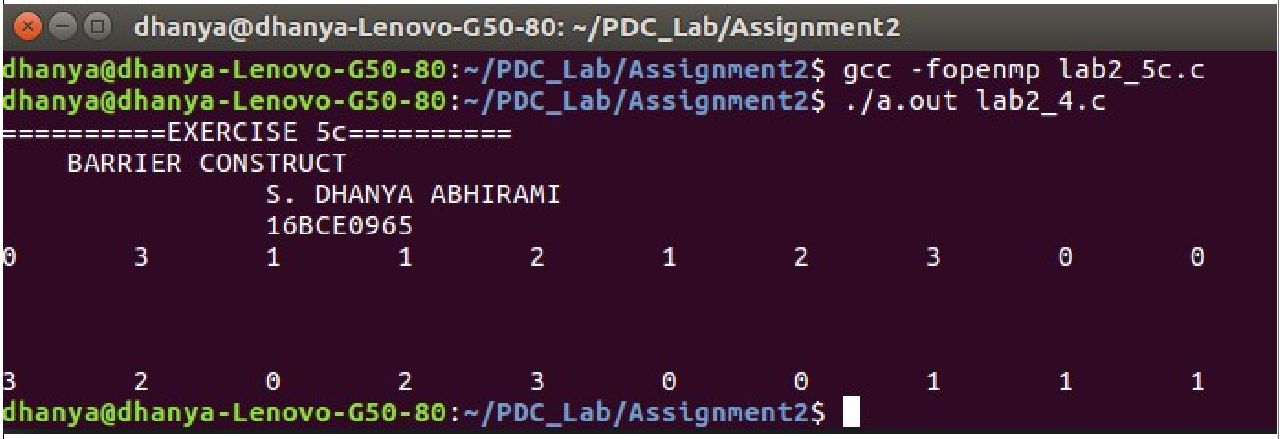


```
16bce0965@sjt516scs051:~$ gcc -fopenmp lab2_5b.c
16bce0965@sjt516scs051:~$ ./a.out lab2_5b.c
==========EXERCISE 5b==========
    SINGLE SYNCHRONISATION MECHANISM
                S. DHANYA ABHIRAMI
                16BCE0965

Thread Id: 3
6 3 7 8 9 4 5 0 1 2
16bce0965@sjt516scs051:~$
```

Barrier

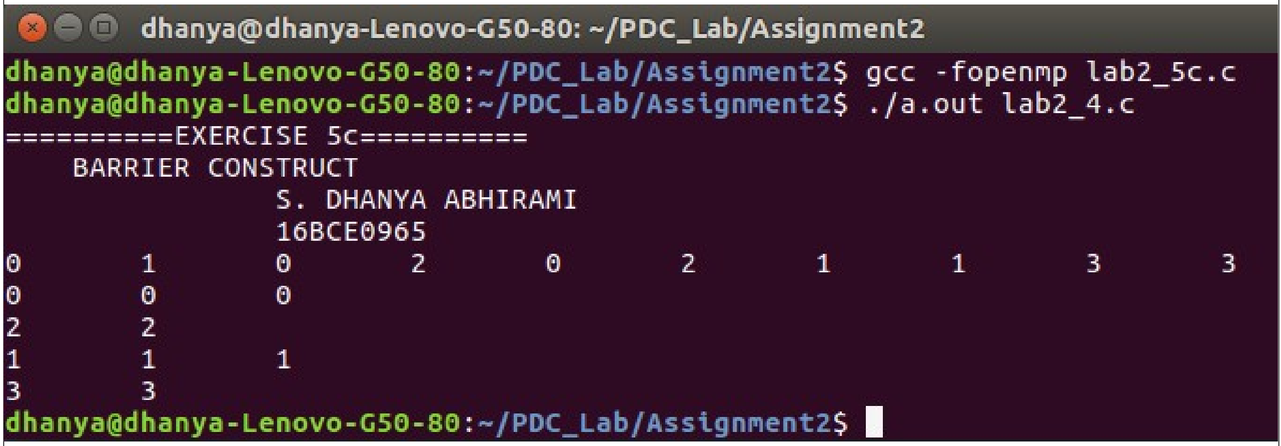
```
#include <omp.h>
#include <stdio.h>

int main() {
    printf("==========EXERCISE 5c==========\n");
    printf("    BARRIER CONSTRUCT\n\t\tS. DHANYA ABHIRAMI\n\t\t16BCE0965\n");
  int n=10,i;
  #pragma omp parallel private(i) shared(n)
  {
    #pragma omp for
    for (i=0; i<n; i++)
    printf("%d\t",omp_get_thread_num() );
    printf("\n");
    #pragma omp barrier
    #pragma omp for
    for (i=0; i<n; i++)
    printf("%d\t",omp_get_thread_num() );
  }
  printf("\n");

 return (0);
}
```

With Barrier Construct



```
dhanya@dhanya-Lenovo-G50-80:~/PDC_Lab/Assignment2$ gcc -fopenmp lab2_5c.c
dhanya@dhanya-Lenovo-G50-80:~/PDC_Lab/Assignment2$ ./a.out lab2_4.c
==========EXERCISE 5c==========
    BARRIER CONSTRUCT
                S. DHANYA ABHIRAMI
                16BCE0965
0       3       1       1       2       1       2       3       0       0



3       2       0       2       3       0       0       1       1       1
dhanya@dhanya-Lenovo-G50-80:~/PDC_Lab/Assignment2$
```

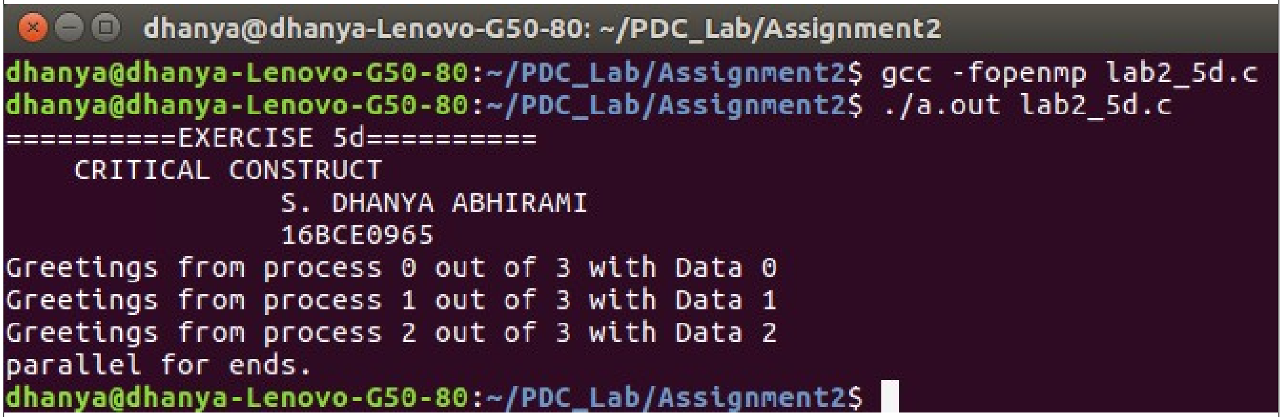
Without Barrier Construct



```
dhanya@dhanya-Lenovo-G50-80:~/PDC_Lab/Assignment2$ gcc -fopenmp lab2_5c.c
dhanya@dhanya-Lenovo-G50-80:~/PDC_Lab/Assignment2$ ./a.out lab2_4.c
==========EXERCISE 5c==========
    BARRIER CONSTRUCT
                S. DHANYA ABHIRAMI
                16BCE0965
0       1       0       2       0       2       1       1       3       3
0       0       0
2       2
1       1       1
3       3
dhanya@dhanya-Lenovo-G50-80:~/PDC_Lab/Assignment2$ 
```

Critical

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

int main() {
     printf("==========EXERCISE 5d==========\n");
   printf("    CRITICAL CONSTRUCT\n\t\tS. DHANYA ABHIRAMI\n\t\t16BCE0965\n");
  int data;
  #pragma omp parallel num_threads(3)
  {
    int id = omp_get_thread_num();
    int total = omp_get_num_threads();
    #pragma omp critical
    { // make sure only 1 thread exectutes the critical section at a time.
        data = id; // threads may interleaving the modification
        printf("Greetings from process %d out of %d with Data %d\n", id, total, data);
    }
  }
  printf("parallel for ends.\n");
  return 0;
}
```



```
dhanya@dhanya-Lenovo-G50-80:~/PDC_Lab/Assignment2$ gcc -fopenmp lab2_5d.c
dhanya@dhanya-Lenovo-G50-80:~/PDC_Lab/Assignment2$ ./a.out lab2_5d.c
==========EXERCISE 5d==========
    CRITICAL CONSTRUCT
                S. DHANYA ABHIRAMI
                16BCE0965
Greetings from process 0 out of 3 with Data 0
Greetings from process 1 out of 3 with Data 1
Greetings from process 2 out of 3 with Data 2
parallel for ends.
dhanya@dhanya-Lenovo-G50-80:~/PDC_Lab/Assignment2$ 
```