**PARALLEL AND DISTRIBUTED COMPUTING LAB**

**REPORT**

**NAME:** S Shyam Sundaram

**REG NO:** 19BCE1560

**PROGRAMMING ENVIRONMENT:** MPI

**PROBLEM:** MPI

**DATE:** 1st December, 2021

**HARDWARE CONFIGURATION:**

|  |  |  |  |
| --- | --- | --- | --- |
| CPU NAME | | : | Intel core i5 – 1035G1 @ 1.00 Ghz |
| Number of Sockets: | | : | 1 |
| Cores per Socket | | : | 4 |
| Threads per core | | : | 1 |
| L1 | Cache size | : | 320KB |
| L2 | Cache size | : | 2MB |
| L3 | Cache size (Shared): | | 6MB |
| RAM | | : | 8 GB |

**QUESTION**

Write an MPI program to sort an array.

**CODE**

#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

const int n=10000;

void swap(int \*a,int \*b)

{

int t=\*a;

\*a=\*b;

\*b=t;

}

void fillArray(int a[],int x)

{

srand(time(0));

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

{

a[i]=(rand()%(100-1+1))+1;

}

}

void merge(int a[],int m,int r)

{

int x[r],c=0;

int i=0,j=m;

while(i<m && j<r)

{

if(a[i]<a[j])

x[c++]=a[i++];

else

x[c++]=a[j++];

}

while(i<m)

x[c++]=a[i++];

while(j<r)

x[c++]=a[j++];

for(int k=0;k<r;++k)

a[k]=x[k];

}

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

{

int a[n];

fillArray(a,n);

int id=0,start,siz;

int comm\_size=0;

MPI\_Init(&argc,&argv);

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

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

double t1, t2;

start=n%comm\_size;

siz=n/comm\_size;

if(id==0)

{

printf("Name: S Shyam Sundaram\nReg num: 19BCE1560\n\n");

t1 = MPI\_Wtime();

if(start!=0)

{

int temp;

for(int i=0;i<start-1;++i)

{

for(int j=0;j<start-i-1;++j)

{

if(a[j]>a[j+1])

{

swap(&a[j],&a[j+1]);

}

}

}

}

}

int arr[siz];

MPI\_Scatter(&a[start],siz,MPI\_INT,arr,siz,MPI\_INT,0,MPI\_COMM\_WORLD);

for(int i=0;i<siz-1;++i)

for(int j=0;j<siz-i-1;++j)

{

if(arr[j]>arr[j+1])

{

swap(&arr[j],&arr[j+1]);

}

}

MPI\_Gather(arr,siz,MPI\_INT,&a[start],siz,MPI\_INT,0,MPI\_COMM\_WORLD);

if(id==0)

{

int off=siz;

if(start!=0)

{

off=start+siz;

}

while(off<n)

{

merge(a,off,off+siz);

off=off+siz;

}

t2 = MPI\_Wtime();

printf( "Elapsed time is %f\n", t2 - t1 );

}

MPI\_Finalize();

return 0;

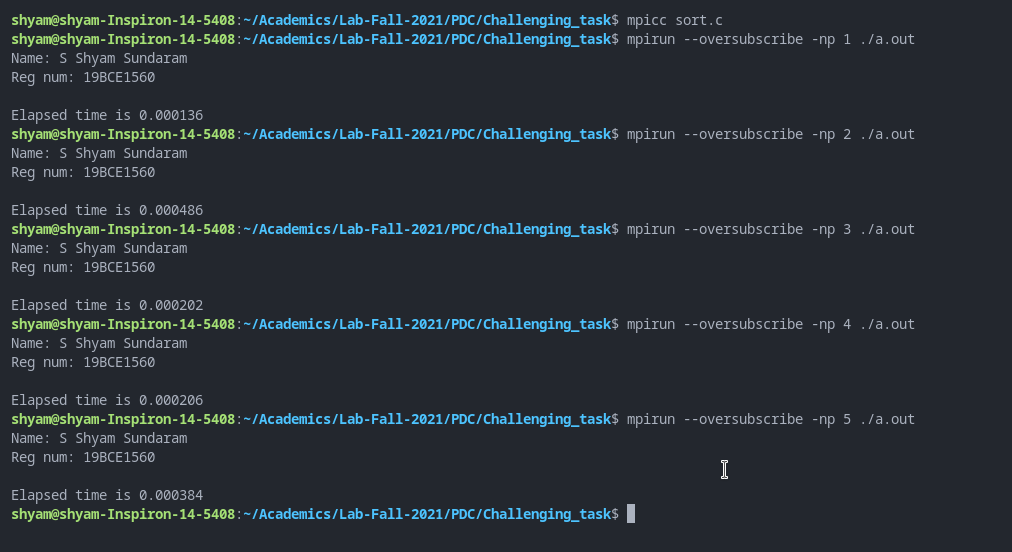
}

**COMMANDS**

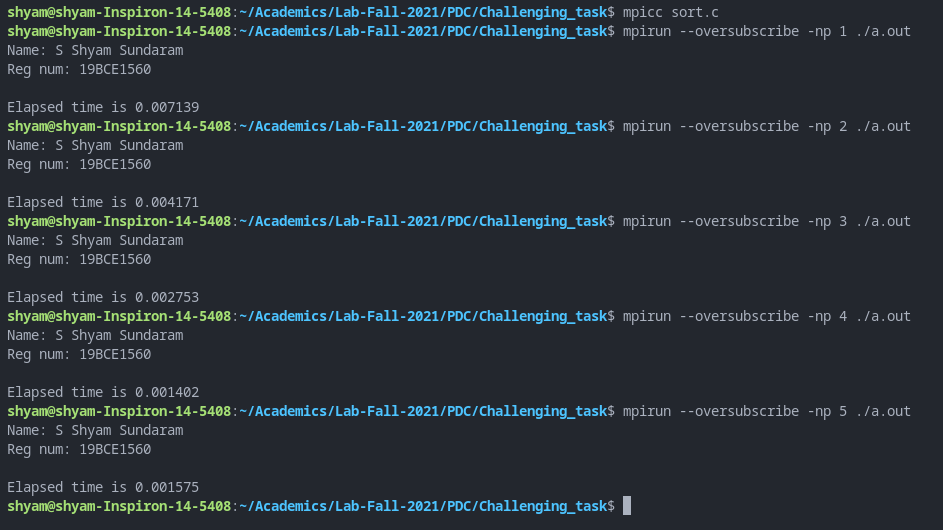
mpicc sort.c

mpirun --oversubscribe -np 4 ./a.out

**OUTPUT**

****

sort.c with 100 elements

****

sort.c with 100 elements

**OBSERVATION**

Each process gets a chunk of the array which then sort them in parallel. Then, the master collects them all and merges the received, sorted partitions in order (like in merge sort).

|  |  |  |
| --- | --- | --- |
| **N** | **NUMBER OF PROCESSES** | **TIME** |
| 100 | 1 | 0.000136 |
| 2 | 0.000486 |
| 3 | 0.000202 |
| 4 | 0.000206 |
| 5 | 0.000384 |
| 1000 | 1 | 0.007139 |
| 2 | 0.004171 |
| 3 | 0.002753 |
| 4 | 0.001402 |
| 5 | 0.001575 |
| 10000 | 1 | 0.943478 |
| 2 | 0.247141 |
| 3 | 0.121381 |
| 4 | 0.125787 |
| 5 | 0.072597 |

**CONCLUSION**

We have sorted an array of elements in parallel using MPI.