**Practical 2 : Merge Sort**

#include<iostream>

#include<stdlib.h>

#include<omp.h>

#include<cstdlib>

using namespace std;

const int threshold = 1000; // Set your desired threshold here

void mergesort\_parallel(int a[],int i,int j);

void merge\_parallel(int a[], int i1, int j1, int i2, int j2);

void merge\_seq(int a[],int i1,int j1,int i2,int j2);

void mergesort\_seq(int a[],int i,int j);

void mergesort\_parallel(int a[], int i, int j)

{

int mid;

if (i < j)

{

mid = (i + j) / 2;

if ((j - i + 1) <= threshold)

{

mergesort\_seq(a, i, mid);

mergesort\_seq(a, mid + 1, j);

}

else

{

#pragma omp parallel sections

{

#pragma omp section

{

mergesort\_parallel(a, i, mid);

}

#pragma omp section

{

mergesort\_parallel(a, mid + 1, j);

}

}

}

merge\_parallel(a, i, mid, mid + 1, j);

}

}

void merge\_parallel(int a[], int i1, int j1, int i2, int j2)

{

int size = (j1 - i1 + 1) + (j2 - i2 + 1);

int\* temp = new int[size];

int i=i1;

int j=i2;

int k=0;

while(i<=j1 && j<=j2)

{

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

{

temp[k++]=a[i++];

}

else

{

temp[k++]=a[j++];

}

}

while(i<=j1)

{

temp[k++]=a[i++];

}

while(j<=j2)

{

temp[k++]=a[j++];

}

for(i=i1,j=0;i<=j2;i++,j++)

{

a[i]=temp[j];

}

delete[] temp; // Free the dynamically allocated memory

}

void mergesort\_seq(int a[], int i, int j)

{

int mid;

if (i < j)

{

mid = (i + j) / 2;

mergesort\_seq(a, i, mid);

mergesort\_seq(a, mid + 1, j);

merge\_seq(a, i, mid, mid + 1, j);

}

}

void merge\_seq(int a[],int i1,int j1,int i2,int j2)

{

int size = (j1 - i1 + 1) + (j2 - i2 + 1);

int\* temp = new int[size];

int i=i1;

int j=i2;

int k=0;

while(i<=j1 && j<=j2)

{

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

{

temp[k++]=a[i++];

}

else

{

temp[k++]=a[j++];

}

}

while(i<=j1)

{

temp[k++]=a[i++];

}

while(j<=j2)

{

temp[k++]=a[j++];

}

for(i=i1,j=0;i<=j2;i++,j++)

{

a[i]=temp[j];

}

delete[] temp; // Free the dynamically allocated memory

}

int main()

{

cout<<"This is Atharva Pingale's Code";

cout<<"\nPractical 2 : Merge Sort\n";

int \*a,\*b;

long int n,i;

double start\_time, end\_time, seq\_time, parallel\_time;

cout<<"\nEnter total no of elements=>";

cin>>n;

a = new int[n];

b = new int[n];

for(i=0;i<n;i++)

{

int random\_value = (rand()%(9999999 - 9999 + 1)) + 9999;

a[i] = random\_value;

b[i] = random\_value;

}

// Sequential algorithm

start\_time = omp\_get\_wtime();

mergesort\_seq(a, 0, n-1);

end\_time = omp\_get\_wtime();

seq\_time = end\_time - start\_time;

cout<<"\nFirst 20 elements of sorted array ( Sequential algorithm ) : \n";

for(i=0;i<20;i++)

{

cout<<a[i]<<", ";

}

// Parallel algorithm

start\_time = omp\_get\_wtime();

mergesort\_parallel(b, 0, n-1);

end\_time = omp\_get\_wtime();

parallel\_time = end\_time - start\_time;

cout<<"\nFirst 20 elements of sorted array ( Parallel algorithm ) : \n";

for(i=0;i<20;i++)

{

cout<<b[i]<<", ";

}

cout << "\n\nSequential Time: " << seq\_time;

cout << "\n\nParallel Time: " << parallel\_time;

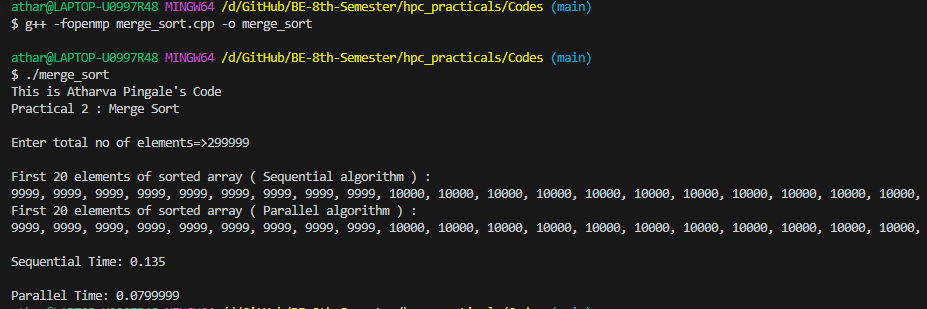
delete[] a;

delete[] b;

return 0;

}

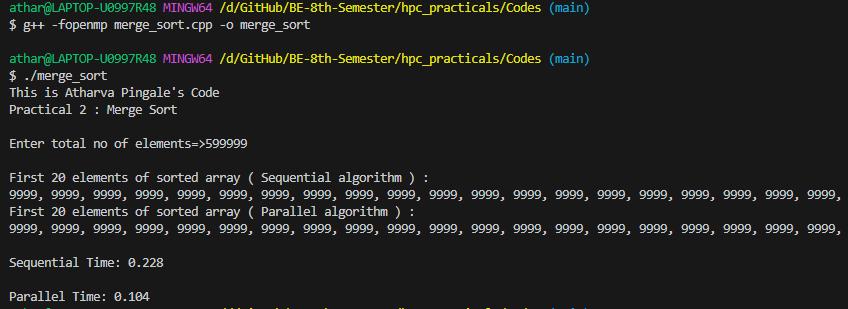
**Output :**



Total number of elements : 299999

Time taken by Sequential Algo : 0.135 seconds

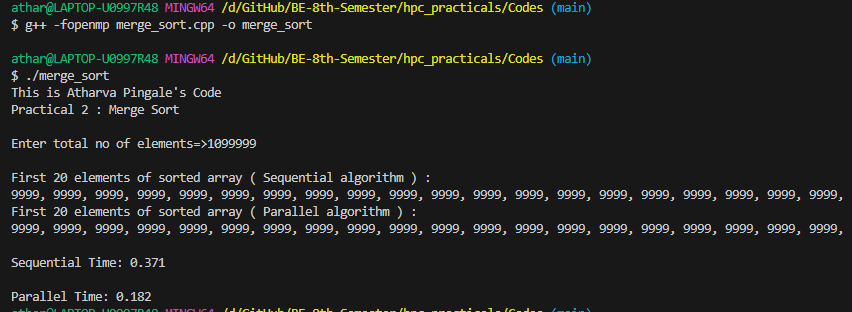
Time taken by Parallel Algo : 0.0799999



Total number of elements : 599999

Time taken by Sequential Algo : 0.228 seconds

Time taken by Parallel Algo : 0.104 seconds



Total number of elements : 1099999

Time taken by Sequential Algo : 0.371 seconds

Time taken by Parallel Algo : 0.182 seconds