```
%%writefile Gauri_merge.cu
#include<iostream>
#include<stdlib.h>
#include<omp.h>
#include<chrono>//for calculating time
#include <bits/stdc++.h>
using namespace std::chrono;
using namespace std;
void mergesort(int a[],int i,int j);
void merge(int a[],int i1,int j1,int i2,int j2);
void mergesort(int a[],int i,int j)
{
int mid;
if(i<j)
{
mid=(i+j)/2;
#pragma omp parallel sections
{
#pragma omp section
{
mergesort(a,i,mid);
}
#pragma omp section
{
mergesort(a,mid+1,j);
}
}
merge(a,i,mid,mid+1,j);
}
}
void merge(int a[],int i1,int j1,int i2,int j2)
{
int temp[1000];
int i,j,k;
i=i1;
j=i2;
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];
}
}
int main()
{
int *a,n,i;
cout<<"\n enter total no of elements=>";
cin>>n;
a= new int[n];
cout<<"\n enter elements=>";
for(i=0;i<n;i++)
{
cin>>a[i];
}
// Sequential algorithm
auto start = high_resolution_clock::now();
mergesort(a, 0, n-1);
auto stop = high_resolution_clock::now();
auto seq_time = duration_cast<microseconds>(stop - start);
cout << "\nSequential Time: " << seq_time.count() << endl;
// Parallel algorithm
auto start_time = high_resolution_clock::now();
#pragma omp parallel
{
#pragma omp single
{
mergesort(a, 0, n-1);
}
}
auto end_time = high_resolution_clock::now();
auto par_time = duration_cast<microseconds>(end_time - start_time);
cout << "\nParallel Time: " << par_time.count()<< endl;
cout<<"\n sorted array is=>";
for(i=0;i<n;i++)
{
cout<<"\n"<<a[i];
}
return 0;
}
```

```
Overwriting Gauri_merge.cu
```

```
!nvcc Gauri_merge.cu -o Gauri_merge
```

```
!./Gauri_merge
```

```
 enter total no of elements=>5

 enter elements=>12
74
51
18
7

Sequential Time: 0

Parallel Time: 0

 sorted array is=>
7
12
18
51
74
```

Start coding or generate with AI.