Title: Write a Parallel program (using OpenMP) for Bubble Sort(Even-Odd Sort).

1)OpenMP:

OpenMP (Open Multi-Processing) is an [application programming interface](https://en.wikipedia.org/wiki/Application_programming_interface) (API) that supports multi-platform [shared memory](https://en.wikipedia.org/wiki/Shared_memory_architecture) [multiprocessing](https://en.wikipedia.org/wiki/Multiprocessing) programming in [C](https://en.wikipedia.org/wiki/C_(programming_language)), [C++](https://en.wikipedia.org/wiki/C++), and [Fortran](https://en.wikipedia.org/wiki/Fortran), on most platforms, [instruction set architectures](https://en.wikipedia.org/wiki/Instruction_set_architecture) and [operating systems](https://en.wikipedia.org/wiki/Operating_system), including [Solaris](https://en.wikipedia.org/wiki/Solaris_(operating_system)), [AIX](https://en.wikipedia.org/wiki/IBM_AIX), [HP-UX](https://en.wikipedia.org/wiki/HP-UX), [Linux](https://en.wikipedia.org/wiki/Linux), [macOS](https://en.wikipedia.org/wiki/MacOS), and [Windows](https://en.wikipedia.org/wiki/Microsoft_Windows). It consists of a set of [compiler directives](https://en.wikipedia.org/wiki/Compiler_directive), [library routines](https://en.wikipedia.org/wiki/Library_(computing)), and [environment variables](https://en.wikipedia.org/wiki/Environment_variable) that influence run-time behavior.

OpenMP is managed by the [nonprofit](https://en.wikipedia.org/wiki/Nonprofit_organization) technology [consortium](https://en.wikipedia.org/wiki/Consortium) OpenMP Architecture Review Board (or OpenMP ARB), jointly defined by a group of major computer hardware and software vendors, including [AMD](https://en.wikipedia.org/wiki/AMD), [IBM](https://en.wikipedia.org/wiki/IBM), [Intel](https://en.wikipedia.org/wiki/Intel), [Cray](https://en.wikipedia.org/wiki/Cray), [HP](https://en.wikipedia.org/wiki/Hewlett-Packard), [Fujitsu](https://en.wikipedia.org/wiki/Fujitsu), [Nvidia](https://en.wikipedia.org/wiki/Nvidia), [NEC](https://en.wikipedia.org/wiki/NEC), [Red Hat](https://en.wikipedia.org/wiki/Red_Hat), [Texas Instruments](https://en.wikipedia.org/wiki/Texas_Instruments), [Oracle Corporation](https://en.wikipedia.org/wiki/Oracle_Corporation), and more.

OpenMP uses a [portable](https://en.wikipedia.org/wiki/Software_portability), scalable model that gives [programmers](https://en.wikipedia.org/wiki/Programmer) a simple and flexible interface for developing parallel applications for platforms ranging from the standard [desktop computer](https://en.wikipedia.org/wiki/Desktop_computer) to the [supercomputer](https://en.wikipedia.org/wiki/Supercomputer).

Comprised of three primary API components:

* Compiler Directives
* Runtime Library Routines
* Environment Variables

**Goals of OpenMP:**

* **Standardization:**
  + Provide a standard among a variety of shared memory architectures/platforms
  + Jointly defined and endorsed by a group of major computer hardware and software vendors
* **Lean and Mean:**
  + Establish a simple and limited set of directives for programming shared memory machines.
  + Significant parallelism can be implemented by using just 3 or 4 directives.
  + This goal is becoming less meaningful with each new release, apparently.
* **Ease of Use:**
  + Provide capability to incrementally parallelize a serial program, unlike message-passing libraries which typically require an all or nothing approach
  + Provide the capability to implement both coarse-grain and fine-grain parallelism
* **Portability:**
  + The API is specified for C/C++ and Fortran
  + Public forum for API and membership
  + Most major platforms have been implemented including Unix/Linux platforms and Windows

**OpenMP Programming Model :**

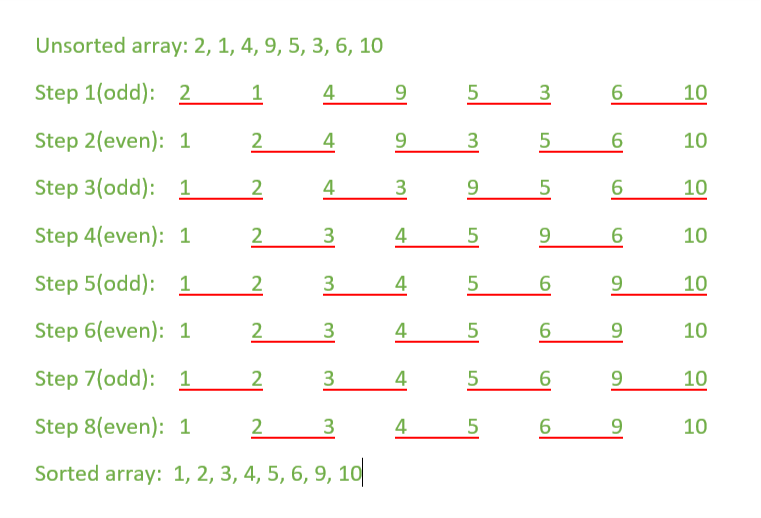
**Shared Memory Model:**

* OpenMP is designed for multi-processor/core, shared memory machines. The underlying architecture can be shared memory UMA or NUMA.
* Because OpenMP is designed for shared memory parallel programming, it largely limited to **single node** parallelism. Typically, the number of processing elements (cores) on a node determine how much parallelism can be implemented.

# **2)Odd Even Transposition Sort**

[Odd-Even Transposition Sort](https://www.geeksforgeeks.org/odd-even-sort-brick-sort/) is a [parallel sorting algorithm](https://www.geeksforgeeks.org/serial-sort-vs-parallel-sort-java/). It is based on the [Bubble Sort technique](https://www.geeksforgeeks.org/bubble-sort/), which compares every 2 consecutive numbers in the array and swap them if first is greater than the second to get an ascending order array. It consists of 2 phases – the odd phase and even phase:

* **Odd phase:** Every odd indexed element is compared with the next even indexed element(considering 1-based indexing).
* **Even phase:** Every even indexed element is compared with the next odd indexed element.



# **3)Bubble Sort**

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

**Example:**  
**First Pass:**  
( **5** **1** 4 2 8 ) –> ( **1** **5** 4 2 8 ), Here, algorithm compares the first two elements, and swaps since 5 > 1.  
( 1 **5** **4** 2 8 ) –>  ( 1 **4** **5** 2 8 ), Swap since 5 > 4  
( 1 4 **5** **2** 8 ) –>  ( 1 4 **2** **5** 8 ), Swap since 5 > 2  
( 1 4 2 **5** **8** ) –> ( 1 4 2 **5** **8** ), Now, since these elements are already in order (8 > 5), algorithm does not swap them.

**Second Pass:**  
( **1** **4** 2 5 8 ) –> ( **1** **4** 2 5 8 )  
( 1 **4** **2** 5 8 ) –> ( 1 **2** **4** 5 8 ), Swap since 4 > 2  
( 1 2 **4** **5** 8 ) –> ( 1 2 **4** **5** 8 )  
( 1 2 4 **5** **8** ) –>  ( 1 2 4 **5** **8** )  
Now, the array is already sorted, but our algorithm does not know if it is completed. The algorithm needs one **whole** pass without **any** swap to know it is sorted.

**Third Pass:**  
( **1** **2** 4 5 8 ) –> ( **1** **2** 4 5 8 )  
( 1 **2** **4** 5 8 ) –> ( 1 **2** **4** 5 8 )  
( 1 2 **4** **5** 8 ) –> ( 1 2 **4** **5** 8 )  
( 1 2 4 **5** **8** ) –> ( 1 2 4 **5** **8** )

**Code:**

#include <iostream>

#include <omp.h>

using namespace std;

#define SIZE 10000

int main()

{

int a[SIZE];

#pragma omp parallel for

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

{

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

}

double start\_t = omp\_get\_wtime();

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

{

if(i%2 == 0)

{

#pragma omp parallel for

for(int j = 0; j < SIZE; j+=2)

{

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

{

int temp = a[j];

a[j] = a[j+1];

a[j+1] = temp;

}

}

}

else

{

#pragma omp parallel for

for(int j = 1; j < SIZE; j+=2)

{

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

{

int temp = a[j];

a[j] = a[j+1];

a[j+1] = temp;

}

}

}

}

double end\_t = omp\_get\_wtime();

double elapse\_t = end\_t - start\_t;

cout<<"\nsorted 5 elements by parallel algo: ";

for (int i = 0; i < 5; i++)

{

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

}

cout<<"\nparallel time elapsed: "<<elapse\_t;

#pragma omp parallel for

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

{

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

}

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

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

{

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

{

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

{

int temp = a[j];

a[j] = a[j+1];

a[j+1] = temp;

}

}

}

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

elapse\_t = end\_t - start\_t;

/\* cout<<"\nsorted 5 elements by serial algo: ";

for (int i = 0; i < 5; i++)

{

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

}\*/

cout<<"\nsequential time elapsed: "<<elapse\_t<<"\n";

return 0;

}

**OUTPUT:**

