**Comparison of Bubble sorting and merge sorting**

**Objective:** Implementing parallelization of Bubble sorting algorithm, odd even sorting and merge sorting.

**Operation of Bubble sort:** It compares each pair of adjacent elements and swaps them as per the order.

**Implementation:** Bubble sorting involves a simple sorting algorithm that sequentially checks the elements in the given array, compares each pair of adjacent items and swaps them as per the order. The array will be kept on sorting until there are no elements needed to be sorted in the given array. It is a comparison sort as per the smaller or larger element in the array. Bubble sort has worst-case and average complexity both О(n2), where n is the number of items being sorted. The only significant advent age that bubble sort has over most other implementations, even quicksort, but not insertion sort, is that the ability to detect that the list is sorted efficiently is built into the algorithm. When the list is already sorted (best-case), the complexity of bubble sort is only O(n).

Bubble sort does not require any amount of additional memory. In parallel representation, the loop must run n-1 times. So, the parallel time complexity will be O(n). If the algorithm has odd numbered steps, then it needs (n/2)-2 processors and even numbered steps need (n/2)-1 processors. So, the total of numbers will be of O(n). In this project, there will be an observation of normal bubble sort and parallelising it regarding time taken and number of steps for implementation. As, there will be a decrease of time of execution in multiple processors is observed vs. the original bubble sort**.**

**Bubble sort serial:**

#include <iostream>

#include <iomanip>

#include <sstream>

#include <stdlib.h>

#include <string.h>

#include <sys/time.h>

#include <math.h>

#define MX\_SZ 320

#define SEED 2397

#define MAX\_VALUE 3000.0

#define MIN\_VALUE -3000.0

#define TIMER\_CLEAR (tv1.tv\_sec = tv1.tv\_usec = tv2.tv\_sec = tv2.tv\_usec = 0)

#define TIMER\_START gettimeofday(&tv1, (struct timezone\*)0)

#define TIMER\_ELAPSED ((long long) (tv3.tv\_usec)+((long long) (tv3.tv\_sec)\*1000000))

#define TIMER\_STOP {gettimeofday(&tv2, (struct timezone\*)0);timersub(&tv2,&tv1,&tv3);}

struct timeval tv1,tv2,tv3;

int numthreads;

void get\_index\_size(int argc,char \*argv[],int \*arr\_size)

{

if(argc!=2)

{

std::cout<<"usage: quick <number of elements>" << std::endl;

exit(1);

}

else

{

if (argc == 2)

\*arr\_size = atoi(argv[1]);

}

if (\*arr\_size<=0)

{

std::cout<<"Error: number of elements must be greater than 0" <<

std::endl;

exit(1);

}

}

void fill\_array(double a[],int num)

{

int i;

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

{

a[i]=drand48()\*MAX\_VALUE;

a[i+1]=drand48()\*MIN\_VALUE;

i++;

}

}

void printArray(double a[], int size)

{

int i;

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

{

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

}

std::cout << "\n";

}

void bubble(double array[],int n)

{

double swap;int c=0,d=0;

for (c = 0 ; c < ( n - 1 ); c++)

{

for (d = 0 ; d < n - c - 1; d++)

{

if (array[d] > array[d+1]) /\* For decreasing order use < \*/

{

swap = array[d];

array[d] = array[d+1];

array[d+1] = swap;

}

}

}

}

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

{

int arr\_size;

get\_index\_size(argc,argv,&arr\_size);

double array[arr\_size];

srand48(SEED);

fill\_array(array,arr\_size);

//std::cout<<"Given array is \n";

//printArray(array, arr\_size);

TIMER\_CLEAR;

TIMER\_START;

bubble(array,arr\_size);

TIMER\_STOP;

//std::cout<<"\nSorted array is \n";

//printArray(array, arr\_size);

std::cout << "time=" << std::setprecision(8)<<TIMER\_ELAPSED/1000000.0<< " seconds" << std::endl;

return 0;

}

**Output:**

Serial

time=0.000124 seconds

time=0.000506 seconds

time=0.002017 seconds

time=0.037341 seconds

time=0.161997 seconds

time=0.682433 seconds

time=2.803556 seconds

time=11.208189 seconds

time=45.035589 seconds

time=181.78824 seconds

Bubble sort parallel:

#include <iostream>

#include <iomanip>

#include <sstream>

#include <stdlib.h>

#include <string.h>

#include <sys/time.h>

#include <math.h>

#include <omp.h>

#define MX\_SZ 320

#define SEED 2397

#define MAX\_VALUE 5000.0

#define MIN\_VALUE -5000.0

#define TIMER\_CLEAR (tv1.tv\_sec = tv1.tv\_usec = tv2.tv\_sec = tv2.tv\_usec = 0)

#define TIMER\_START gettimeofday(&tv1, (struct timezone\*)0)

#define TIMER\_ELAPSED ((long long) (tv3.tv\_usec)+((long long) (tv3.tv\_sec)\*1000000))

#define TIMER\_STOP

{gettimeofday(&tv2, (struct timezone\*)0);timersub(&tv2,&tv1,&tv3);}

struct timeval tv1,tv2,tv3;

int numthreads;

void get\_index\_size(int argc,char \*argv[],int \*numthreads,int \*arr\_size)

{

if(argc!=3)

{

std::cout<<"usage: quick <number of elements>" << std::endl;

exit(1);

}

else

{

if (argc == 3)

\*numthreads= atoi(argv[1]);

\*arr\_size = atoi(argv[2]);

}

if (\*arr\_size<=0)

{

std::cout<<"Error: number of elements must be greater than 0" << std::endl;

exit(1);

}

}

void fill\_array(double array[],int num)

{

int i;

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

{

array[i]=drand48()\*MAX\_VALUE;

array[i+1]=drand48()\*MIN\_VALUE;

i++;

}

}

void merge(double\*,int,int,int);

void merge\_sort(double\* array,int low,int high)

{

int mid;

if(low<high)

{

mid=(low+high)/2;

#pragma omp num\_threads(numthreads) task firstprivate(array,low,mid)

merge\_sort(array,low,mid);

#pragma omp num\_threads(numthreads) task firstprivate(array,mid,high)

merge\_sort(array,mid+1,high);

#pragma omp num\_threads(numthreads) taskwait

merge(array,low,mid,high);

}

}

void merge(double a[],int low,int mid,int high)

{

int h,i,j,k;

double b[high];

h=low;

i=low;

j=mid+1;

while((h<=mid)&&(j<=high))

{

if(a[h]<=a[j])

{

b[i]=a[h];

h++;

}

else

{

b[i]=a[j];

j++;

}

i++;

}

if(h>mid)

{

for(k=j;k<=high;k++)

{

b[i]=a[k];

i++;

}

}

else

{

for(k=h;k<=mid;k++)

{

b[i]=a[k];

i++;

}

}

for(k=low;k<=high;k++)

{

a[k]=b[k];

}

}

void printArray(double A[], int size)

{

int i;

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

{

std::cout << A[i] << " ";

}

std::cout << "\n";

}

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

{

int arr\_size;

get\_index\_size(argc,argv,&numthreads,&arr\_size);

double array[arr\_size];

srand48(SEED);

fill\_array(array,arr\_size);

//std::cout<<"Given array is \n";

//printArray(array, arr\_size);

TIMER\_CLEAR;

TIMER\_START;

merge\_sort(array,0,arr\_size-1);

TIMER\_STOP;

//std::cout<<"\nSorted array is \n";

//printArray(array, arr\_size);

std::cout << "time=" << std::setprecision(8) <<TIMER\_ELAPSED/1000000.0 << " seconds" << std::endl;

return 0;

}

Output:

timing for 1 thread case

time=0.000672 seconds

time=0.001003 seconds

time=0.003043 seconds

time=0.038726 seconds

time=0.155124 seconds

time=0.60727 seconds

time=2.385188 seconds

time=9.429647 seconds

time=38.232846 seconds

time=170.19087 seconds

timing for 2 thread case

time=0.00057 seconds

time=0.001063 seconds

time=0.002428 seconds

time=0.021456 seconds

time=0.082981 seconds

time=0.317654 seconds

time=1.248125 seconds

time=4.851585 seconds

time=19.188217 seconds

time=89.184595 seconds

timing for 4 thread case

time=0.000671 seconds

time=0.001052 seconds

time=0.002141 seconds

time=0.016032 seconds

time=0.052269 seconds

time=0.197784 seconds

time=0.782023 seconds

time=3.009908 seconds

time=11.840464 seconds

time=63.173955 seconds

timing for 8 thread case

time=0.00924 seconds

time=0.020809 seconds

time=0.033701 seconds

time=0.128852 seconds

time=0.286318 seconds

time=0.668369 seconds

time=1.667845 seconds

time=4.878467 seconds

time=16.640869 seconds

time=71.028545 seconds

Merge Sort:

#include <iostream>

#include <iomanip>

#include <sstream>

#include <stdlib.h>

#include <string.h>

#include <sys/time.h>

#include <math.h>

#include <omp.h>

#define MX\_SZ 320

#define SEED 2397

#define MAX\_VALUE 5000.0

#define MIN\_VALUE -5000.0

#define TIMER\_CLEAR (tv1.tv\_sec = tv1.tv\_usec = tv2.tv\_sec = tv2.tv\_usec = 0)

#define TIMER\_START gettimeofday(&tv1, (struct timezone\*)0)

#define TIMER\_ELAPSED ((long long) (tv3.tv\_usec)+((long long) (tv3.tv\_sec)\*1000000))

#define TIMER\_STOP {gettimeofday(&tv2, (struct timezone\*)0);timersub(&tv2,&tv1,&tv3);}

struct timeval tv1,tv2,tv3;

int numthreads;

void get\_index\_size(int argc,char \*argv[],int \*numthreads,int \*arr\_size)

{

if(argc!=3)

{

std::cout<<"usage: quick <number of elements>" << std::endl;

exit(1);

}

else

{

if (argc == 3)

\*numthreads= atoi(argv[1]);

\*arr\_size = atoi(argv[2]);

}

if (\*arr\_size<=0)

{

std::cout<<"Error: number of elements must be greater than 0" << std::endl;

exit(1);

}

}

void fill\_array(double array[],int num)

{

int i;

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

{

array[i]=drand48()\*MAX\_VALUE;

array[i+1]=drand48()\*MIN\_VALUE;

i++;

}

}

void merge(double\*,int,int,int);

void merge\_sort(double\* array,int low,int high)

{

int mid;

if(low<high)

{

mid=(low+high)/2;

#pragma omp num\_threads(numthreads) task firstprivate(array,low,mid)

merge\_sort(array,low,mid);

#pragma omp num\_threads(numthreads) task firstprivate(array,mid,high)

merge\_sort(array,mid+1,high);

#pragma omp num\_threads(numthreads) taskwait

merge(array,low,mid,high);

}

}

void merge(double a[],int low,int mid,int high)

{

int h,i,j,k;

double b[high];

h=low;

i=low;

j=mid+1;

while((h<=mid)&&(j<=high))

{

if(a[h]<=a[j])

{

b[i]=a[h];

h++;

}

else

{

b[i]=a[j];

j++;

}

i++;

}

if(h>mid)

{

for(k=j;k<=high;k++)

{

b[i]=a[k];

i++;

}

}

else

{

for(k=h;k<=mid;k++)

{

b[i]=a[k];

i++;

}

}

for(k=low;k<=high;k++)

{

a[k]=b[k];

}

}

void printArray(double A[], int size)

{

int i;

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

{

std::cout << A[i] << " ";

}

std::cout << "\n";

}

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

{

int arr\_size;

get\_index\_size(argc,argv,&numthreads,&arr\_size);

double array[arr\_size];

srand48(SEED);

fill\_array(array,arr\_size);

//std::cout<<"Given array is \n";

//printArray(array, arr\_size);

TIMER\_CLEAR;

TIMER\_START;

merge\_sort(array,0,arr\_size-1);

TIMER\_STOP;

//std::cout<<"\nSorted array is \n";

//printArray(array, arr\_size);

std::cout << "time=" << std::setprecision(8) <<TIMER\_ELAPSED/1000000.0 << " seconds" << std::endl;

return 0;}

timing for 1 thread case

time=3.3e-05 seconds

time=6.5e-05 seconds

time=0.000137 seconds

time=0.000579 seconds

time=0.00118 seconds

time=0.002646 seconds

time=0.005573 seconds

time=0.011721 seconds

time=0.026974 seconds

time=0.054716 seconds

timing for 2 thread case

time=2.8e-05 seconds

time=5.8e-05 seconds

time=0.000137 seconds

time=0.000531 seconds

time=0.001073 seconds

time=0.002428 seconds

time=0.005023 seconds

time=0.010451 seconds

time=0.022513 seconds

time=0.04944 seconds

timing for 4 thread case

time=2.8e-05 seconds

time=5.8e-05 seconds

time=0.000112 seconds

time=0.000539 seconds

time=0.001068 seconds

time=0.002417 seconds

time=0.005043 seconds

time=0.010431 seconds

time=0.022771 seconds

time=0.048132 seconds

timing for 8 thread case

time=2.8e-05 seconds

time=5.9e-05 seconds

time=0.000115 seconds

time=0.000572 seconds

time=0.001074 seconds

time=0.002449 seconds

time=0.005017 seconds

time=0.010391 seconds

time=0.022879 seconds

time=0.047965 seconds

***Conclusion:***

From the obtained results we can observe merge sort is faster than bubble sort, in all cases. As the serial code is implemented in openmp and executed in jetson there is a perfect implementation. It was easier completing the parallelization.