PDC PROJECT REPORT

PERFORMANCE EVALUATION OF PARALLEL BUBBLE SORT & SEQUENTIAL BUBBLE SORT

Ahmed Saeed 19K-0159
Muhammad Zahid 19K-0334
Muhammad Zayyan 19K-0182
Section 5C

Department of Computer Science, FAST NATIONAL UNIVERSITY

INTRODUCTION

A sorting algorithm is used to put up all the items in list of order, such as ascending or descending depending on any attribute type we choose Sorting a large data can mainly take a long time, especially if it's a large list. A computer program can be created to do this in a way that sorting such list takes less time and it is much easier.

When it comes to parallel sorting, we are talking about using multiple processors so that performance of the serial form of the similar algorithm could be made better. The performance is evaluated by checking the growth rate and dependency of the threads allocated. In this report we are going to look efficiency and parallel speedup of the parallel bubble sort versus the sequential bubble sort.

OPENMP (Parallelism) is used for implementing the parallel version of bubble sort to conduct our results.

GOAL: TO FIND OUT HOW EFFICIENT CAN PARALLEL BUBBLE SORT CAN PERFORM AGAINST OTHER SERIAL SORTING TECHNIQUES.

SEQUENTIAL BUBBLE SORT

Bubble sort is a sorting algorithm that works by repeatedly stepping through lists that need to be sorted, comparing each pair of adjacent items and swapping them if they are in the wrong order. This passing procedure is repeated until no swaps are required, indicating that the list is sorted. Bubble sort has a worst-case and average complexity of $O(n^2)$, where n is the number of items sorted. Unlike the other sorting algorithms, bubble sort detects whether the sorted list is efficiently built into the algorithm. Bubble sort performance over an already sorted list is O(n).

We do not consider using sequential bubble sort for large data sets.

Bubble sort as above compared array[i] and array[i+1] and swap if want, then compare array[i+1] and array[i+2] until and unless the maximum values moves to the end of the array. Repeat it until the second maximum value is not next to the most maximum values. All of this shall be repeated until no swaps remaining.

Parallel Bubble Sort

Another name to Parallel Bubble sort is Odd Even Transposition bubble sort. This compares the inputs in parallel, that means alternate odds and evens. As implemented the snapshot of part of our code is shown.

Algorithm of Bubble Sort:

```
void BubbleSort_Id(Employee* e2,int left, int right, int tid)
{

    for (int i = left; i < right; ++i)
    {

        for (int j = left + 1; j < right; j += 2)
        {
            if (e2[j].id < e2[j - 1].id)
            {
                Employee temp3 = e2[j];
                e2[j] = e2[j - 1];
                e2[j - 1] = temp3;
            }
        }
        for (int j = left + 2; j < right; j += 2)
        {
            if (e2[j].id < e2[j - 1].id)
            {
                Employee temp4 = e2[j];
                e2[j] = e2[j - 1];
                e2[j - 1] = temp4;
            }
        }
    }
}
```

Algorithm of Merging the Thread level output:

```
void merge_Id(Employee* a, Employee* b, int l, int mid, int r) {
    int i, left_end, count, tmp_pos;
    left_end = mid - 1;
    tmp_pos = l;
    count = r - l + 1;
    while ((l <= left_end) && (mid <= r)) {
        if (a[l].id <= a[mid].id) {
            b[tmp_pos] = a[l];
            tmp_pos++;
            l++;
        }
        else {
            b[tmp_pos] = a[mid];
            tmp_pos++;
            mid++;
        }
    }
    while (l <= left_end) {
        b[tmp_pos] = a[l];
        l++;
        tmp_pos++;
    }
    while (mid <= r) {
        b[tmp_pos] = a[mid];
        mid++;
        tmp_pos++;
    }
    for (i = 0; i < count; i++) {
        a[r] = b[r];
        r--;
    }
}
```

Parallel region of our Code:

```
        Employee* result = new Employee[count];
        double start = omp_get_wtime();
        double t1, t2, t3, t4;
#pragma omp parallel num_threads(nothreads)
        {
            int tid = omp_get_thread_num();
            int l = tid * subarraylength;
            int r = l + subarraylength;
            if (tid < nothreads - 1) {
                ThreadResult t;
                double st = omp_get_wtime();
                t.Setter(tid, e1, l, r, "id");
                double en = omp_get_wtime();
                t1 = en - st;
                BubbleSort_Id(e1, l, r, tid);
                double st1 = omp_get_wtime();
                t.Setter(tid, e1, l, r, "id");
                double en1 = omp_get_wtime();
                t2 = en1 - st1;
            }
            else {
                double st2 = omp_get_wtime();
                ThreadResult t;
                t.Setter(tid, e1, l, r, "id");
                double en2 = omp_get_wtime();
                t3 = en2 - st2;
                BubbleSort_Id(e1, l, count, tid);
                double st3 = omp_get_wtime();
                t.Setter(tid, e1, l, r, "id");
                double en3 = omp_get_wtime();
                t4 = en3 - st3;
            }
#pragma omp barrier
        }
        for (int i = 1; i < nothreads - 1; i++) {
            int mid = i * subarraylength;
            merge_Id(e1, result, 0, mid, mid + subarraylength - 1);
        }
        merge_Id(e1, result, 0, (nothreads - 1) * subarraylength, count - 1);
        double end = omp_get_wtime();
        fstream nf;
```

If we have number of processes that are lower than n/2 where n is the data size, then every processor will carry out (n/2)/p comparisons, and complexity will be $O(n^2/2p)$, where p is the number of processors. If the system has a more number of processors than n/2 then complexity of this algorithm is O(1) because all the iterations are performed in parallel.

In the best form case, the processor will sort n/p of data using bubble sort. There are two loops, the outer loop will take n iterations and inner loop is dependent on the odd and even state which will take n/2 steps, so that time is divided with the number of processors.

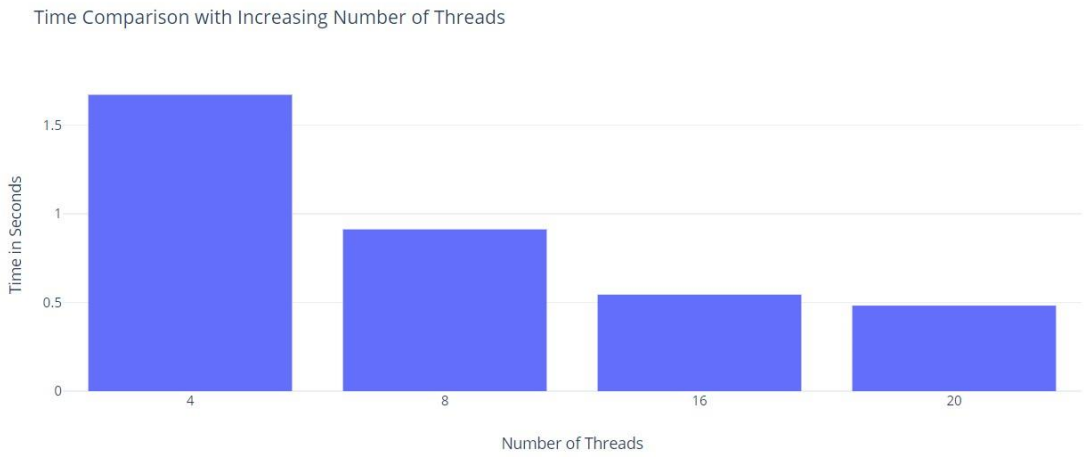
RESULTS

We have found results in form of time taken and parallel efficiency. OPENMP library is used for implementation of the parallel bubble sort according to different number of processors

Run Time Evaluation

A fixed data size of 2000 is chosen, the behavior is summarized as follows.

- As the number of processors increases the run time is reduced due to better parallelism and load distribution among the threads. We have analyzed this from 4 to 8 to 16 to 20 processors.



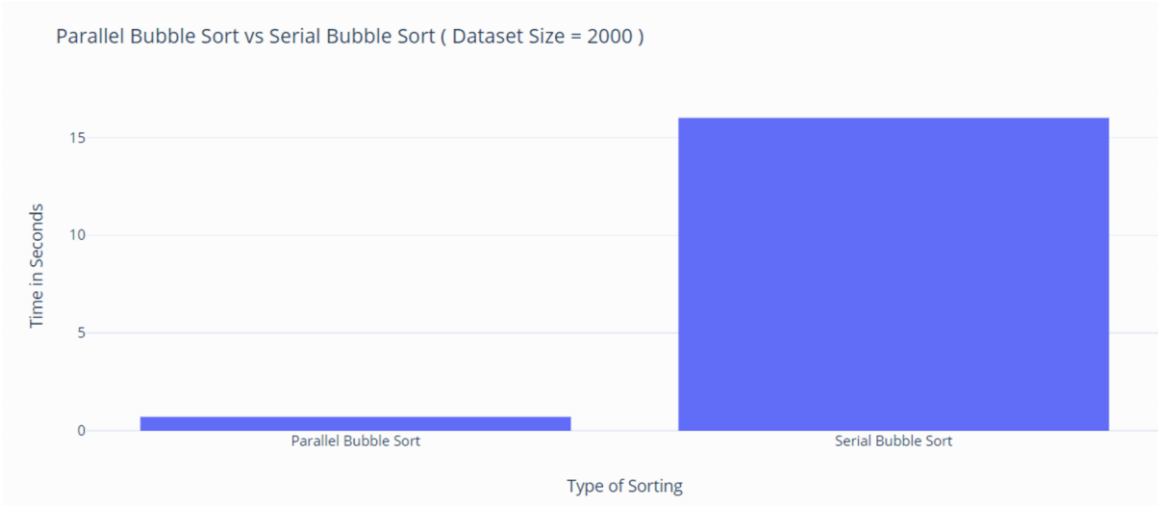
4.5 Difference Between Serial and Parallel.

Here we will compare the time results of Serial Bubble Sort with Parallel Bubble sort.

The Difference between Serial and Parallel time on different dataset size

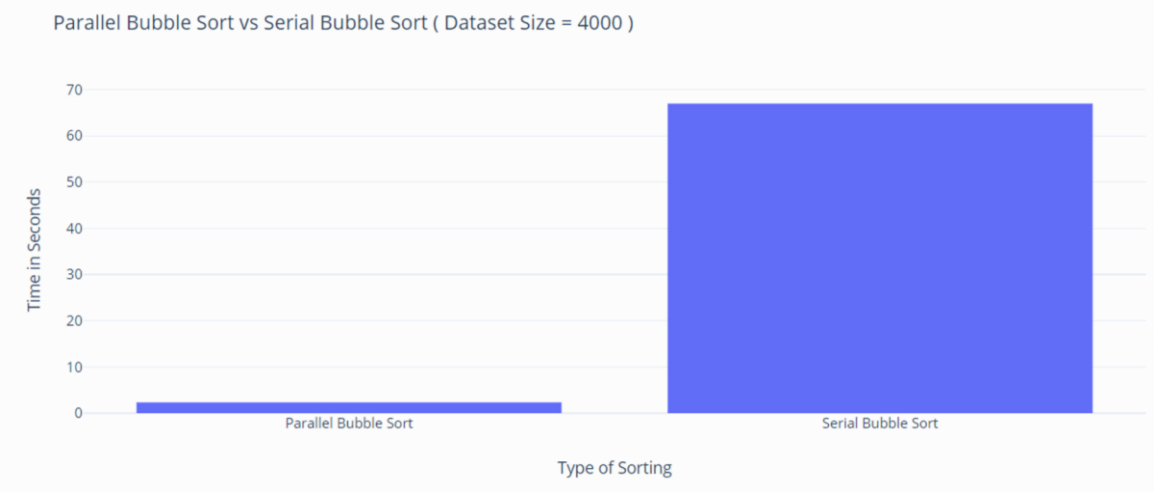
Number of Threads = 16

Dataset Size	Serial	Parallel
2000	17 seconds	0.71 seconds



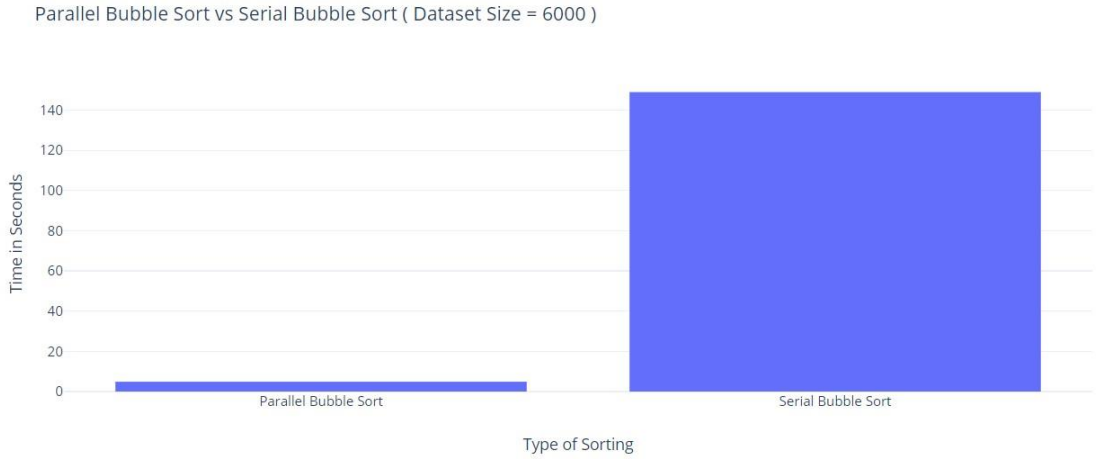
Number of Threads = 16

Dataset Size	Serial	Parallel
4000	67 seconds	2.3 seconds



Number of Threads = 16

Dataset Size	Serial	*Parallel*
6000	149 seconds	4.9 seconds

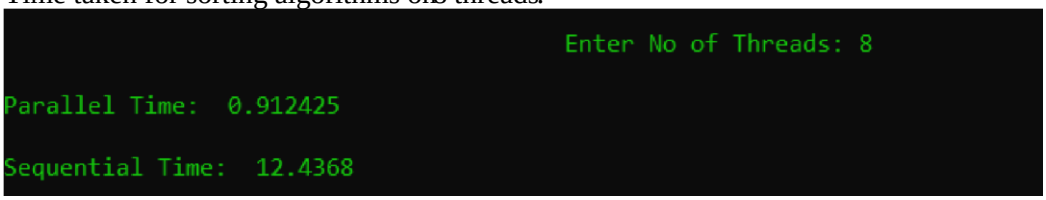


What did we analyze? We see no matter what size of the data is, Parallel bubble sorting Algorithm works efficiently and shows that it is time efficient as compared to Serial

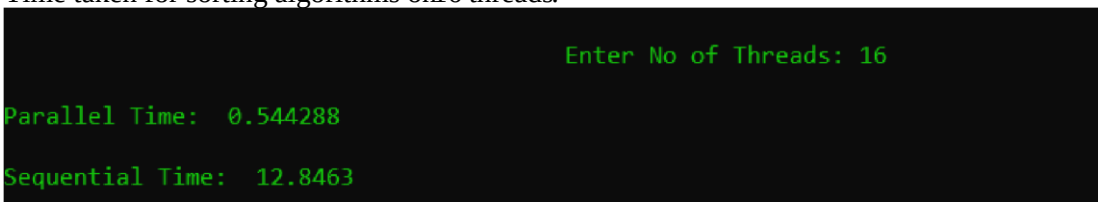
Time taken for sorting algorithms on 4 threads:

```
                                Enter No of Threads: 4

Parallel Time:  1.67111

Sequential Time:  12.7895
```

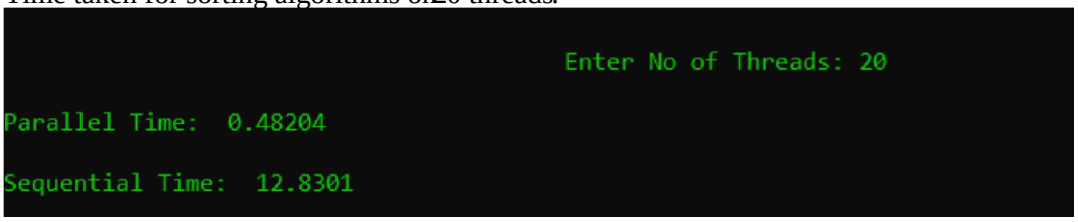
Time taken for sorting algorithms on 8 threads:



Time taken for sorting algorithms on 16 threads:



Time taken for sorting algorithms on 20 threads:



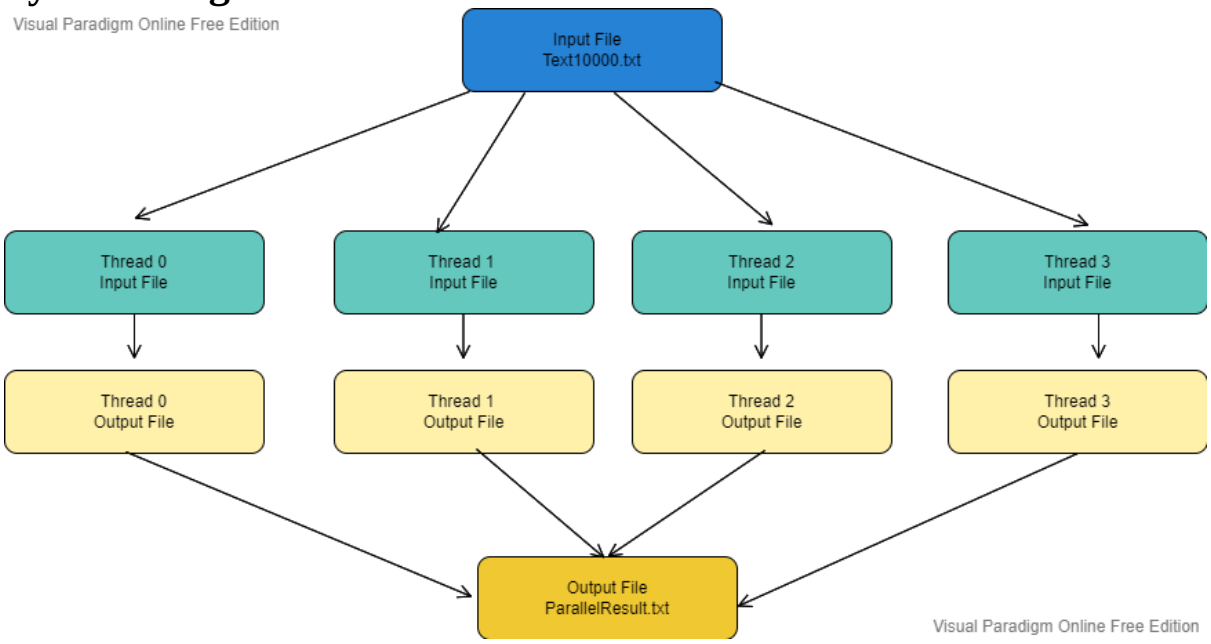
What we see here? We see that on a dataset of any size with threads from 4 to 20, Parallel Bubble Sort out performs Serial bubble sort.

Source Description

The source data we use are from www.kaggle.com this is the data of an Employee which contains the seven attributes of an Employee which include his ID, Name, Department, gender, Location, Salary and Tenure. This data has multiple datatypes which include integer, Strings and Float.

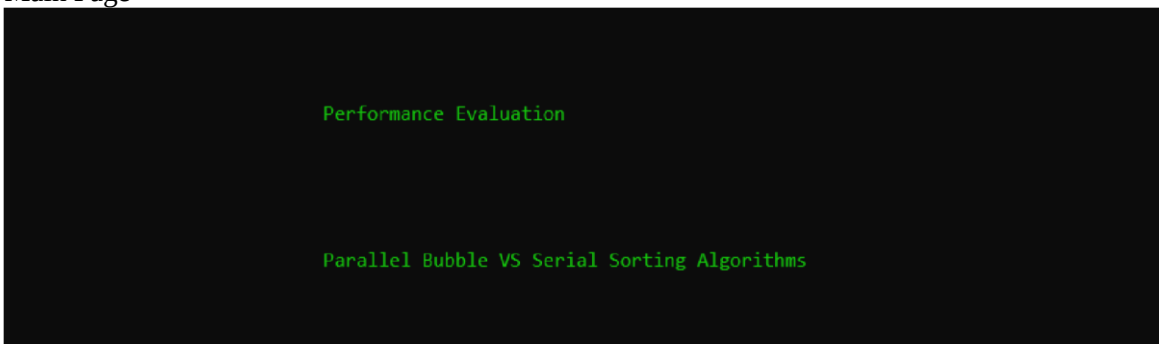
8712	Jada	Female	Engineering	568700	Bengaluru	15.6		
8711	Moses	Male	Engineering	508600	Mumbai	15.6		
8710	Willow	Female	Engineering	416000	Mumbai	15.7		
8709	Hadassah		Male	Engineering	416000	Mumbai	15.7	
8708	Amery	Female	Support	1107300	Mumbai	15.8		
8707	Travis	Male	Accounting	1041200	Mumbai	15.8		
8706	Leila	Female	Marketing	373600	Bengaluru	15.8		
8705	Yvonne	Male	Support	1107300	Mumbai	15.8		
8704	Zoe	Female	Support	1107300	Mumbai	15.8		
8703	Uta	Male	Accounting	1041200	Mumbai	15.8		
8702	Eagan	Female	Marketing	373600	Bengaluru	15.8		
8701	Akeem	Male	Support	1107300	Mumbai	15.8		
8700	Lawrence		Female	Support	432000	Mumbai	15.9	
8699	Josiah	Male	Engineering	908000	Mumbai	15.9		
8698	Freya	Female	Humanresource	763200	Bengaluru	15.9		
8697	Aurelia	Male	Support	432000	Mumbai	15.9		
8696	August	Female	Support	432000	Mumbai	15.9		
8695	Abigail	Male	Engineering	908000	Mumbai	15.9		
8694	Jin	Female	Humanresource	763200	Bengaluru	15.9		
8693	Derek	Male	Support	432000	Mumbai	15.9		
8692	Zenia	Female	Accounting	529600	Bengaluru	16.2		
8691	August	Male	Accounting	529600	Bengaluru	16.2		
8690	Sean	Female	ProjectManagement	812200	Bengaluru	16.4		
8689	Rhonda	Male	ProjectManagement	812200	Bengaluru	16.4		
8688	Teagan	Female	ProjectManagement	812200	Bengaluru	16.4		
8687	Ethan	Male	ProjectManagement	812200	Bengaluru	16.4		
8686	Rogan	Female	ProjectManagement	812200	Bengaluru	16.4		
8685	Hedwig	Male	Support	693400	Bengaluru	16.6		
8684	Paloma	Female	Humanresource	503100	Delhi	16.6		
8683	Dillon	Male	Support	693400	Bengaluru	16.6		
8682	Georgia	Female	Support	693400	Bengaluru	16.6		
8681	Drake	Male	Humanresource	503100	Delhi	16.6		
8680	Len	Female	Support	693400	Bengaluru	16.6		
8679	Christen		Male	Business	785000	Delhi	16.7	
8678	Mari	Female	Business	1178500	Delhi	16.7		
8677	Octavius		Male	Legal	898400	Mumbai	16.7	
8676	Cally	Female	Business	785000	Delhi	16.7		
8675	Frances	Male	Business	1178500	Delhi	16.7		
8674	Magee	Female	Legal	898400	Mumbai	16.7		
8673	Jessamine		Male	Services	697300	Delhi	16.8	
8672	Jasmine	Female	Services	697300	Delhi	16.8		
8671	Alexander		Male	Services	697300	Delhi	16.8	
8670	Vanna	Female	Services	697300	Delhi	16.8		
8669	Upton	Male	Services	699100	Delhi	16.9		
8668	Xenos	Female	Humanresource	312000	Mumbai	16.9		
8667	Sigourney		Male	Services	699100	Delhi	16.9	
8666	Claire	Female	Services	699100	Delhi	16.9		
8665	Maggie	Male	Humanresource	312000	Mumbai	16.9		
8664	Rogan	Female	Services	699100	Delhi	16.9		
8663	Tucker	Male	Accounting	996300	Mumbai	17		
8662	Xena	Female	Humanresource	580300	Delhi	17		
8661	Cruz	Male	Accounting	996300	Mumbai	17		
8660	Curran	Female	Humanresource	580300	Delhi	17		
8659	Kimberly		Male	Business	665100	Delhi	17.2	
8658	Keiko	Female	Business	665100	Delhi	17.2		
8657	Brenden	Male	Humanresource	532400	Mumbai	17.4		
8656	Thor	Female	Humanresource	532400	Mumbai	17.4		
8655	Ila	Male	Training	939600	Delhi	17.6		
8654	Ryan	Female	ProjectManagement	682000	Bengaluru	17.6		
8653	Fatima	Male	Training	939600	Delhi	17.6		
8652	Myles	Female	ProjectManagement	682000	Bengaluru	17.6		
8651	Shelly	Male	Training	939600	Delhi	17.6		
8650	Blaze	Female	Training	939600	Delhi	17.6		
8649	Plato	Male	ProjectManagement	682000	Bengaluru	17.6		
8648	Rafael	Female	Training	939600	Delhi	17.6		

System Diagram

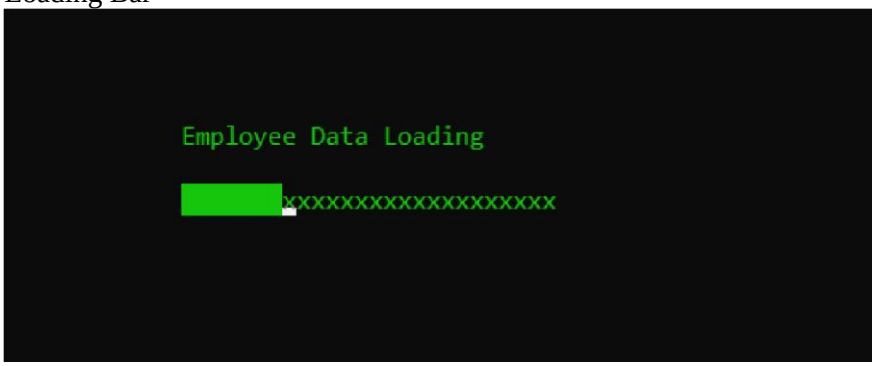


Output Snapshots

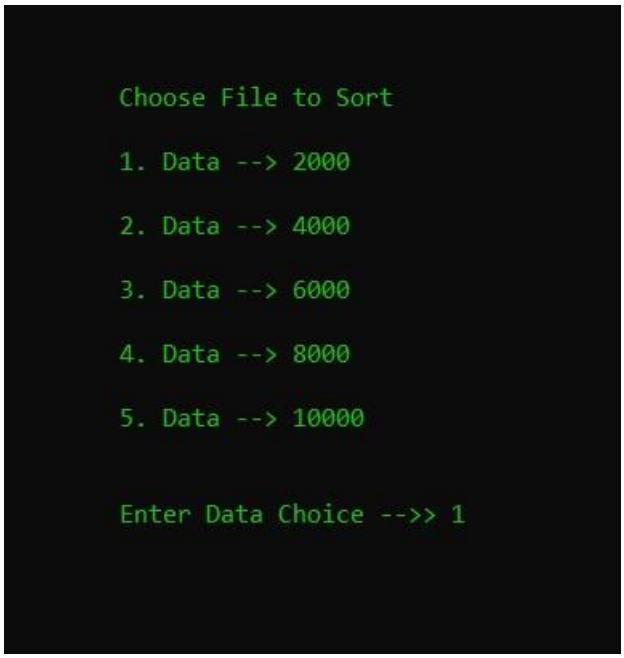
Main Page



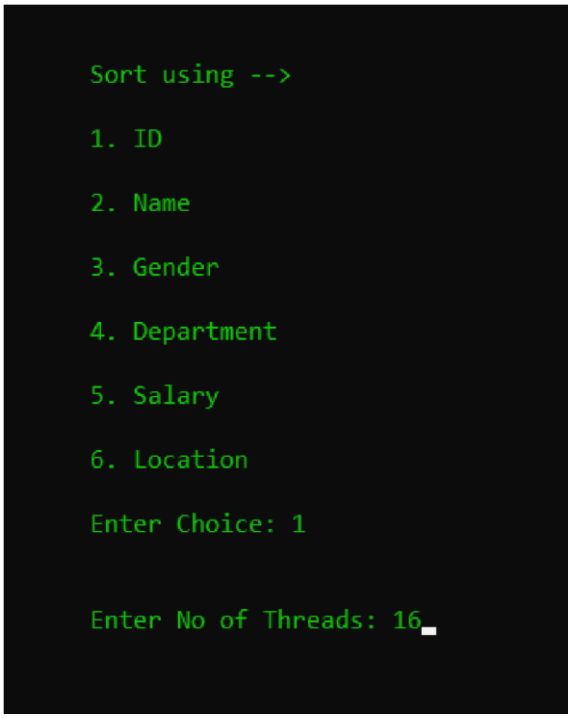
Loading Bar



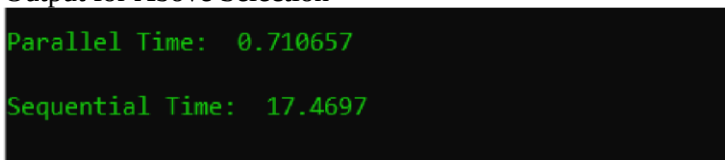
Dataset Size Choice Menu



Attribute Choice Menu



Output for Above Selection



Time taken for sorting algorithms on Dataset of Size = 4000

```
Parallel Time:  2.33908

Sequential Time:  67.863
```

Time taken for sorting algorithms on Dataset of Size = 6000

```
Parallel Time:  4.89658

Sequential Time:  149.469
```

Time taken for sorting algorithms on 4 threads:

```
                                Enter No of Threads: 4

Parallel Time:  1.67111

Sequential Time:  12.7895
```

Time taken for sorting algorithms on 8 threads:

```
                                Enter No of Threads: 8

Parallel Time:  0.912425

Sequential Time:  12.4368
```

Time taken for sorting algorithms on 16 threads:

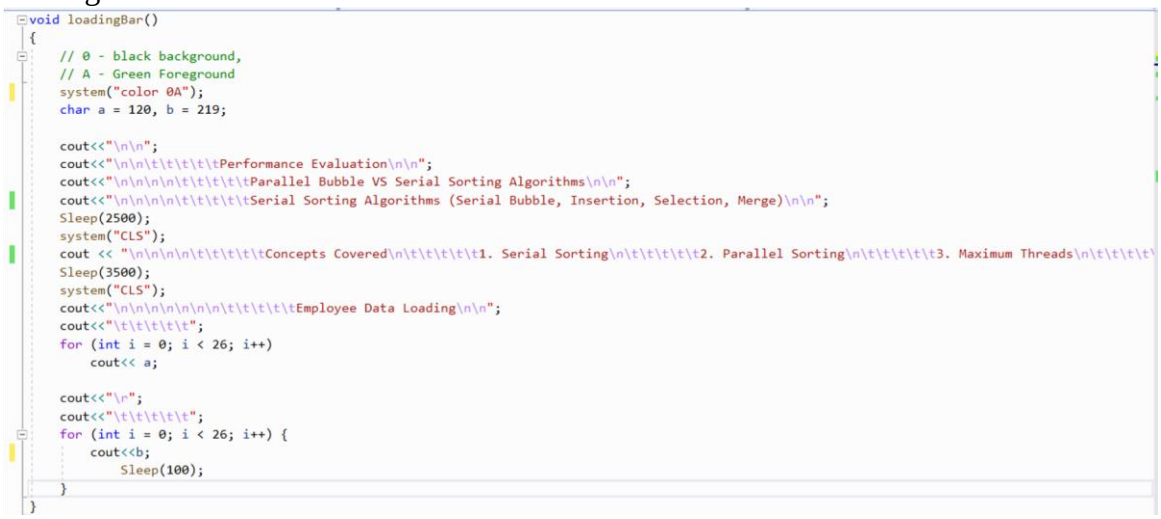
```
                                    Enter No of Threads: 16

Parallel Time:  0.544288

Sequential Time:  12.8463
```

Time taken for sorting algorithms on 20 threads:

```
                                    Enter No of Threads: 20

Parallel Time:  0.48204

Sequential Time:  12.8301
```

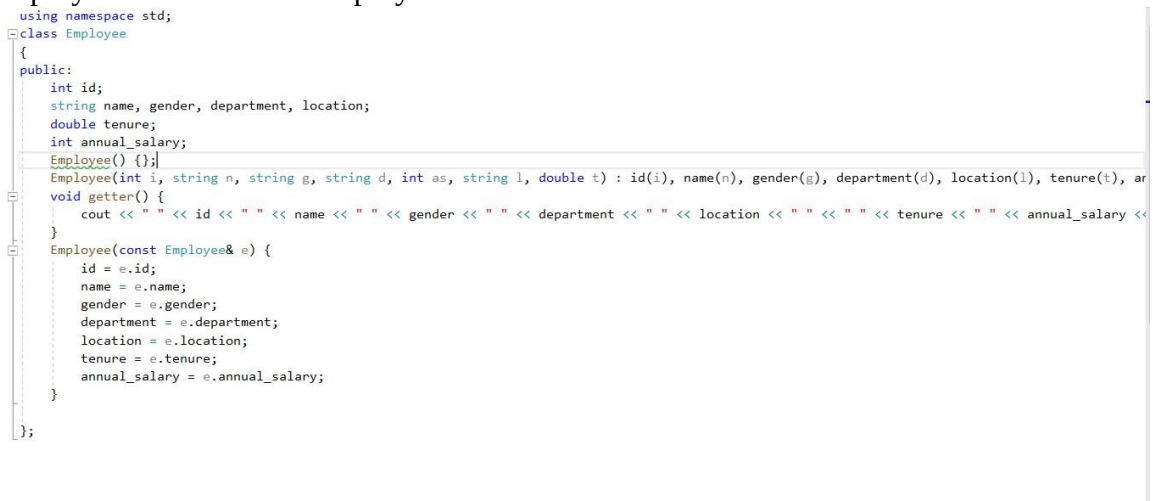
Code Snapshots

Loading Bar



```
void loadingBar()
{
    // 0 - black background,
    // A - Green Foreground
    system("color 0A");
    char a = 120, b = 219;

    cout<<"\n\n";
    cout<<"\n\n\t\t\t\t\tPerformance Evaluation\n\n";
    cout<<"\n\n\n\n\t\t\t\t\tParallel Bubble VS Serial Sorting Algorithms\n\n";
    cout<<"\n\n\n\n\t\t\t\t\tSerial Sorting Algorithms (Serial Bubble, Insertion, Selection, Merge)\n\n";
    Sleep(2500);
    system("CLS");
    cout << "\n\n\n\n\t\t\t\t\tConcepts Covered\n\t\t\t\t\t1. Serial Sorting\n\t\t\t\t\t2. Parallel Sorting\n\t\t\t\t\t3. Maximum Threads\n\t\t\t\t\t
    Sleep(3500);
    system("CLS");
    cout<<"\n\n\n\n\n\n\n\t\t\t\t\tEmployee Data Loading\n\n";
    cout<<"\t\t\t\t\t";
    for (int i = 0; i < 26; i++)
        cout<< a;

    cout<<"\r";
    cout<<"\t\t\t\t\t";
    for (int i = 0; i < 26; i++) {
        cout<<b;
            Sleep(100);
    }
}
```

Employee Class to Store Employee Data



```
using namespace std;
class Employee
{
public:
    int id;
    string name, gender, department, location;
    double tenure;
    int annual_salary;
    Employee() {};
    Employee(int i, string n, string g, string d, int as, string l, double t) : id(i), name(n), gender(g), department(d), location(l), tenure(t), ar
    void getter() {
        cout << " " << id << " " << name << " " << gender << " " << department << " " << location << " " << " " << tenure << " " << annual_salary <<
    }
    Employee(const Employee& e) {
        id = e.id;
        name = e.name;
        gender = e.gender;
        department = e.department;
        location = e.location;
        tenure = e.tenure;
        annual_salary = e.annual_salary;
    }

};
```

Parallel Sorting

Parallel Bubble Sort performed using OPENMP, Number of threads as user input. Data is read from file and it is distributed among threads. Each thread sorts its own data. Merge functions merges all data from each thread into one file.

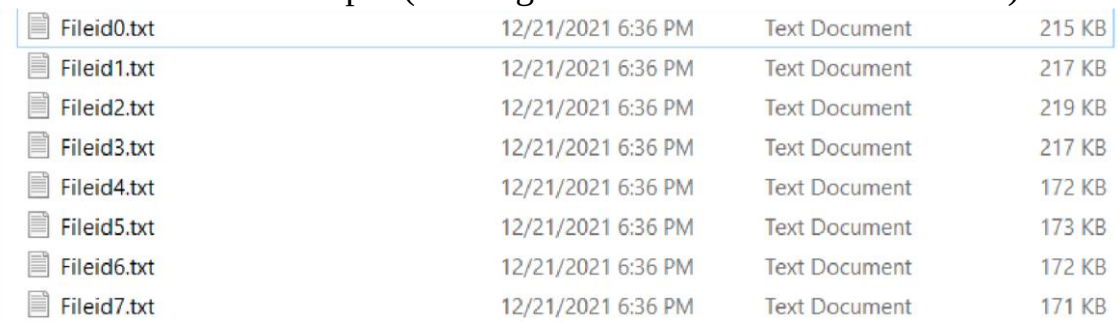
```
        /// <summary>
        /// Parallel Sorting Phase Start
        /// </summary>
        /// <returns></returns>
        cout << "\n\n\t\t\t\t\tEnter No of Threads: ";
        cin >> nothreads;
        int subarraylength = count / nothreads;
        //omp_set_num_threads(nothreads);
        Employee* result = new Employee[count];
        double start = omp_get_wtime();
        double t1, t2, t3, t4;
#pragma omp parallel num_threads(nothreads)
        {
            int tid = omp_get_thread_num();
            int l = tid * subarraylength;
            int r = l + subarraylength;
            if (tid < nothreads - 1) {
                ThreadResult t;
                double st = omp_get_wtime();
                t.Setter(tid, e1, l, r, "id");
                double en = omp_get_wtime();
                t1 = en - st;
                BubbleSort_Id(e1, l, r, tid);
                double st1 = omp_get_wtime();
                t.Setter(tid, e1, l, r, "id");
                double en1 = omp_get_wtime();
                t2 = en1 - st1;
            }
            else {
                double st2 = omp_get_wtime();
                ThreadResult t;
                t.Setter(tid, e1, l, r, "id");
```

```
(Global Scope)                                                    main()
            t.Setter(tid, e1, l, r, "id");
            double en2 = omp_get_wtime();
            t3 = en2 - st2;
            BubbleSort_Id(e1, l, count, tid);
            double st3 = omp_get_wtime();
            t.Setter(tid, e1, l, r, "id");
            double en3 = omp_get_wtime();
            t4 = en3 - st3;
        }
#pragma omp barrier
    }
    for (int i = 1; i < nothreads - 1; i++) {
        int mid = i * subarraylength;
        merge_Id(e1, result, 0, mid, mid + subarraylength - 1);
    }
    merge_Id(e1, result, 0, (nothreads - 1) * subarraylength, count - 1);
    double end = omp_get_wtime();
    fstream pf;
    pf.open("ParallelResultOfID.txt", ios::out);
    for (int z = 0; z < count; z++) {
        pf << e1[z].id << " " << e1[z].name << " " << e1[z].annual_salary << " " << e1[z].gender << " " << e1[z].location << " " << e1[z].depart
    }
    pf.close();
    cout << "\nParallel Time:  " << end - start<< endl;
    /// <summary>
    /// Parallel Sorting Phase End
    /// </summary>
```

```
void BubbleSort_Id(Employee* e2,int left, int right, int tid)
{

    for (int i = left; i < right; ++i)
    {

        for (int j = left + 1; j < right; j += 2)
        {
            if (e2[j].id < e2[j - 1].id)
            {
                Employee temp3 = e2[j];
                e2[j] = e2[j - 1];
                e2[j - 1] = temp3;
            }
        }
        for (int j = left + 2; j < right; j += 2)
        {
            if (e2[j].id < e2[j - 1].id)
            {
                Employee temp4 = e2[j];
                e2[j] = e2[j - 1];
                e2[j - 1] = temp4;
            }
        }
    }
}
```

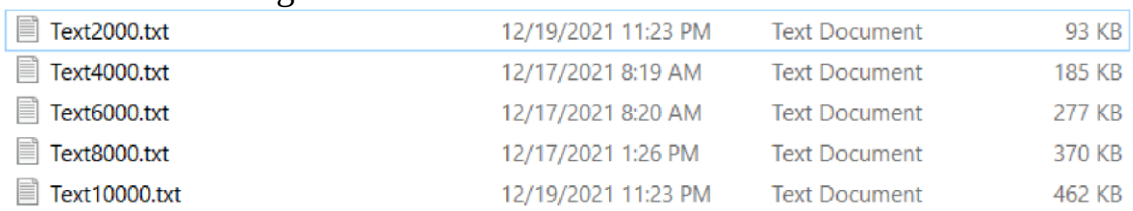
```
void merge_Id(Employee* a, Employee* b, int l, int mid, int r) {
    int i, left_end, count, tmp_pos;
    left_end = mid - 1;
    tmp_pos = l;
    count = r - l + 1;
    while ((l <= left_end) && (mid <= r)) {
        if (a[l].id <= a[mid].id) {
            b[tmp_pos] = a[l];
            tmp_pos++;
            l++;
        }
        else {
            b[tmp_pos] = a[mid];
            tmp_pos++;
            mid++;
        }
    }
    while (l <= left_end) {
        b[tmp_pos] = a[l];
        l++;
        tmp_pos++;
    }
    while (mid <= r) {
        b[tmp_pos] = a[mid];
        mid++;
        tmp_pos++;
    }
    for (i = 0; i < count; i++) {
        a[r] = b[r];
        r--;
    }
}
```

Local Thread Level Output (Sorting that each Processor Performed)



Fileid0.txt	12/21/2021 6:36 PM	Text Document	215 KB
Fileid1.txt	12/21/2021 6:36 PM	Text Document	217 KB
Fileid2.txt	12/21/2021 6:36 PM	Text Document	219 KB
Fileid3.txt	12/21/2021 6:36 PM	Text Document	217 KB
Fileid4.txt	12/21/2021 6:36 PM	Text Document	172 KB
Fileid5.txt	12/21/2021 6:36 PM	Text Document	173 KB
Fileid6.txt	12/21/2021 6:36 PM	Text Document	172 KB
Fileid7.txt	12/21/2021 6:36 PM	Text Document	171 KB

Data Files according to their size



Text2000.txt	12/19/2021 11:23 PM	Text Document	93 KB
Text4000.txt	12/17/2021 8:19 AM	Text Document	185 KB
Text6000.txt	12/17/2021 8:20 AM	Text Document	277 KB
Text8000.txt	12/17/2021 1:26 PM	Text Document	370 KB
Text10000.txt	12/19/2021 11:23 PM	Text Document	462 KB

Unsorted Data ID Descending Order IN Single Thread (Worst Case Scenario)

9998	Ahmed	Female	Accounting	763000	Delhi	1.5	
9997	Jaime	Male	Humanresource	878500	Delhi	2	
9996	Rama	Female	Humanresource	356700	Delhi	2.5	
9995	Marsden	Male	Legal	486300	Delhi	3.4	
9994	Rhoda	Female	Support	611000	Delhi	3.4	
9993	Brian	Male	Support	611000	Delhi	3.4	
9992	Jane	Female	Training	596700	Delhi	3.4	
9991	Preston	Male	Training	741100	Delhi	3.4	
9990	Dillon	Female	Accounting	1064600	Bengaluru		3.5
9989	Jenette	Male	Accounting	1132800	Mumbai	3.5	
9988	Len	Female	Business	1014200	Bengaluru		3.5
9987	Briar	Male	Humanresource	679600	Mumbai	3.5	
9986	Cleo	Female	Humanresource	1175200	Delhi	3.5	
9985	Lael	Male	Legal	367400	Delhi	3.5	
9984	Heidi	Female	Accounting	1064600	Bengaluru		3.5
9983	Calista	Male	Accounting	1132800	Mumbai	3.5	
9982	Liberty	Female	Business	1014200	Bengaluru		3.5
9981	Darius	Male	Humanresource	679600	Mumbai	3.5	
9980	Yolanda	Female	Humanresource	1175200	Delhi	3.5	
9979	Ivor	Male	Legal	367400	Delhi	3.5	
9978	Carla	Female	Services	619200	Delhi	3.6	
9977	Chester	Male	Support	422400	Mumbai	3.6	
9976	Samson	Female	Support	800600	Delhi	3.6	
9975	Ima	Male	Training	1083400	Delhi	3.6	
9974	Stuart	Female	Research	986400	Delhi	3.6	
1	Yvette	Male	CEO	10000000	Mumbai	1	
9973	Amal	Male	Training	1083400	Delhi	3.6	
9972	Kiona	Female	Research	986400	Delhi	3.6	

SORTED DATA OUTPUT ACCORDING TO ID IN SINGLE THREAD

```
Sorted Data
1
9500 Alea 913100 Female Delhi Training 6.2
9501 Inez 595600 Male Delhi Training 6.2
9502 Hiram 508000 Female Mumbai Support 6.2
9503 Martena 855300 Male Delhi Services 6.2
9504 Armand 694600 Female Mumbai Engineering 6.2
9505 Daryl 783800 Male Mumbai Engineering 6.2
9506 Sylvia 1052900 Female Delhi Business 6.2
9507 Flavia 984000 Male Bengaluru Accounting 6.2
9508 Aquila 587400 Female Delhi Training 6.2
9509 Benjamin 913100 Male Delhi Training 6.2
9510 Salvador 595600 Female Delhi Training 6.2
9511 Ori 587400 Male Delhi Training 6.2
9512 Trevor 913100 Female Delhi Training 6.2
9513 Zorita 595600 Male Delhi Training 6.2
9514 Darryl 587400 Female Delhi Training 6.2
9515 Philip 913100 Male Delhi Training 6.2
9516 Sage 595600 Female Delhi Training 6.2
9517 Summer 508000 Male Mumbai Support 6.2
9518 Clarke 855300 Female Delhi Services 6.2
9519 Illana 899600 Male Bengaluru ProjectManagement 6.1
9520 Cade 921900 Female Mumbai Support 6.1
9521 Dane 365500 Male Delhi Accounting 6.1
9522 Grady 899600 Female Bengaluru ProjectManagement 6.1
9523 Alec 921900 Male Mumbai Support 6.1
9524 Hedda 899600 Female Bengaluru ProjectManagement 6.1
9525 Hollee 921900 Male Mumbai Support 6.1
9526 Mara 365500 Female Delhi Accounting 6.1
9527 Darryl 899600 Male Bengaluru ProjectManagement 6.1
9528 Joel 899600 Female Bengaluru ProjectManagement 6.1
9529 Sierra 921900 Male Mumbai Support 6.1
9530 Graiden 589400 Female Delhi Support 6
9531 Xandra 589400 Male Delhi Support 6
9532 Blythe 684300 Female Mumbai Humanresource 6
9533 Mohammad 589400 Male Delhi Support 6
9534 Audra 589400 Female Delhi Support 6
9535 Demetria 589400 Male Delhi Support 6
9536 Beverly 684300 Female Mumbai Humanresource 6
9537 Clayton 589400 Male Delhi Support 6
9538 Bert 581300 Female Delhi Services 5.9
9539 Chaney 1158400 Male Bengaluru Legal 5.9
9540 Galena 485300 Female Delhi Humanresource 5.9
9541 Cole 1144300 Male Bengaluru Engineering 5.9
9542 Zenia 525900 Female Bengaluru Engineering 5.9
9543 Clementine 674300 Male Delhi Accounting 5.9
```

SORTED DATA FINAL OUTPUT ACCORDING TO ID

```
1 Yvette 10000000 Male Mumbai CEO 1
6000 Omar 1165200 Female Bengaluru Business 6.7
6001 Lisandra 718200 Male Delhi Accounting 6.7
6002 Yoshio 940700 Female Mumbai Accounting 6.7
6003 Anika 354400 Male Mumbai Accounting 6.7
6004 Lamar 1084500 Female Mumbai Support 6.7
6005 Ezekiel 1036000 Male Bengaluru Training 6.6
6006 Venus 732400 Female Delhi Support 6.6
6007 Michelle 428200 Male Delhi Services 6.6
6008 Reagan 1036000 Female Bengaluru Training 6.6
6009 Aspen 732400 Male Delhi Support 6.6
6010 Hedda 299700 Female Delhi Legal 6.6
6011 Lila 1114800 Male Delhi Legal 6.6
6012 Aiko 795900 Female Mumbai Humanresource 6.6
6013 Germane 346500 Male Mumbai Humanresource 6.6
6014 Yetta 682200 Female Bengaluru Engineering 6.6
6015 Lucas 715900 Male Bengaluru Business 6.6
6016 Matthew 1036000 Female Bengaluru Training 6.6
6017 Raven 732400 Male Delhi Support 6.6
6018 Graiden 1036000 Female Bengaluru Training 6.6
6019 Aiko 732400 Male Delhi Support 6.6
6020 Beck 428200 Female Delhi Services 6.6
6021 Galvin 1036000 Male Bengaluru Training 6.6
6022 Rashad 732400 Female Delhi Support 6.6
6023 Ronan 299700 Male Delhi Legal 6.6
6024 Ciara 1114800 Female Delhi Legal 6.6
6025 Sylvester 795900 Male Mumbai Humanresource 6.6
6026 Wing 346500 Female Mumbai Humanresource 6.6
6027 Kirestin 682200 Male Bengaluru Engineering 6.6
6028 Hakeem 715900 Female Bengaluru Business 6.6
6029 Cailin 1036000 Male Bengaluru Training 6.6
6030 Rae 1036000 Female Bengaluru Training 6.6
6031 Alika 732400 Male Delhi Support 6.6
6032 Yoko 428200 Female Delhi Services 6.6
6033 Conan 711800 Male Mumbai Humanresource 6.5
6034 Madonna 433300 Female Mumbai Engineering 6.5
6035 Nissim 711800 Male Mumbai Humanresource 6.5
6036 Gisela 433300 Female Mumbai Engineering 6.5
6037 Shaine 997500 Male Delhi Legal 6.4
6038 Abbot 997500 Female Delhi Legal 6.4
6039 Alice 844700 Male Bengaluru Humanresource 6.3
6040 Rajah 436000 Female Mumbai Engineering 6.3
6041 Bradley 702700 Male Mumbai Business 6.3
6042 Rinah 1171500 Female Mumbai Accounting 6.3
6043 Ulla 844700 Male Bengaluru Humanresource 6.3
6044 Lacota 436000 Female Mumbai Engineering 6.3
6045 Otto 702700 Male Mumbai Business 6.3
6046 Basil 1171500 Female Mumbai Accounting 6.3
6047 Chase 587400 Male Delhi Training 6.2
6048 Thaddeus 913100 Female Delhi Training 6.2
6049 Ria 595600 Male Delhi Training 6.2
6050 Jaden 508000 Female Mumbai Support 6.2
6051 Chanda 855300 Male Delhi Services 6.2
6052 Vera 694600 Female Mumbai Engineering 6.2
6053 Griffin 783800 Male Mumbai Engineering 6.2
6054 MacKenzie 1052900 Female Delhi Business 6.2
6055 Cullen 984000 Male Bengaluru Accounting 6.2
6056 Bruce 587400 Female Delhi Training 6.2
6057 Cynthia 913100 Male Delhi Training 6.2
6058 Cadman 595600 Female Delhi Training 6.2
6059 Micah 587400 Male Delhi Training 6.2
6060 Erasmus 913100 Female Delhi Training 6.2
6061 Vernon 595600 Male Delhi Training 6.2
6062 Vivien 508000 Female Mumbai Support 6.2
6063 Alea 855300 Male Delhi Services 6.2
6064 Wendy 587400 Female Delhi Training 6.2
6065 Paul 913100 Male Delhi Training 6.2
6066 Amy 595600 Female Delhi Training 6.2
6067 Vielka 508000 Male Mumbai Support 6.2
6068 Melyssa 855300 Female Delhi Services 6.2
6069 Emily 694600 Male Mumbai Engineering 6.2
6070 Quinlan 783800 Female Mumbai Engineering 6.2
```

CONCLUSION

After thoroughly analyzing in all condition, the performance of parallel bubble sort algorithm in means of the execution time and efficiency is shown. We implemented this method using OPENMP library. This was tested on different number of thread/processors and varying dataset of Employee (any data set can be used, we used Employee data set). For now, results show that parallel bubble sort out performs in terms of execution time as compared to other serial sorting algorithms (serial bubble, serial insertion sort, serial selection sort) except Serial merge sort. According to this, we can say that we can use this parallel algorithm and merge sort for sorting large data sets instead of choosing other serial algorithms.

More the number of threads/processors less the time Parallel algorithm takes, while it still doesn't take lesser time than the merge sort but is still in a far better position as compared to Insertion, Selection and Serial bubble for sorting of large datasets.

This project was created after immense hard work of 3 of us as we had to learn Parallelism using OPENMP, Local thread level sorting and merging of data of each thread in order to create an algorithm efficient enough to work better than its previous version.

Thank you!

Ahmed Saeed 19K-0159

Muhammad Zayyan 19K-0182

Muhammad Zahid 19K-0334