PADC

Lab – 4

K. Manoj Kumar

2022BCS0184

1. Write an OpenMP program with C++ that solves the system of linear equations $\boldsymbol{Ax} = \boldsymbol{b}$ using Gaussian elimination with row pivoting, followed by backward substitution. The following components are to be shown. (a) Write the serial version program to solves the system of linear equations, $\boldsymbol{Ax} = \boldsymbol{b}$. Calculate the execution time by using the OpenMP library function. Test Case: N=Number of Equations, M=Number of unknowns N=3, M=3 $x-y+z=8$, $2x+3y-z=-2$, $3x-2y-9z=9$ Solution: (x, y, z) = (4, -3, 1) (b) Write the parallel version program to estimate the same. Test the result with (a). It includes number of threads involved and the result calculated by which thread number. Calculate the execution time by using the OpenMP library function. (c) Identify the line of statement which leads the Race condition. Race condition occurs when the multiple threads accessing a shared variable. If it exists how will you handle this problem? Use appropriate OpenMP directives/clauses and find the solution. Test the result with value obtained in (a) and (b). Calculate the execution time by using the OpenMP library function.

Program:

```
#include <bits/stdc++.h>

#include <omp.h>


using namespace std;


vector<vector<double>> pivot(vector<vector<double>>& q, int r1, int r2, int
a) {

    double r = q[r2][a];

    #pragma omp parallel for

    for(int i = 0; i < q[r2].size(); i++) {
        q[r2][i] /= r;
    }

    double p = q[r1][a];
    #pragma omp parallel for
    for(int i = 0; i < q[r2].size(); i++) {
        q[r1][i] -= q[r2][i] * p;
    }

    return q;
}
vector<vector<double>> s_pivot(vector<vector<double>>& q, int r1, int r2,
int a) {
    double r = q[r2][a];
    for(int i = 0; i < q[r2].size(); i++) {
        q[r2][i] /= r;
    }

    double p = q[r1][a];
    for(int i = 0; i < q[r2].size(); i++) {
        q[r1][i] -= q[r2][i] * p;
    }

    return q;
}

vector<double> b_substitution(const vector<vector<double>>& q, int n) {
    vector<double> result(n, 0);
    for (int i = n - 1; i >= 0; --i) {
        double s = 0;
        #pragma omp parallel for reduction(+:s)
        for (int j = i + 1; j < n; ++j) {
            s += result[j] * q[i][j];
        }
        result[i] = (q[i][n] - s) / q[i][i];
    }
    return result;
}
vector<double> s_b_substitution(const vector<vector<double>>& q, int n) {
    vector<double> result(n, 0);
    for (int i = n - 1; i >= 0; --i) {
        double s = 0;
        for (int j = i + 1; j < n; ++j) {
            s += result[j] * q[i][j];
        }
        result[i] = (q[i][n] - s) / q[i][i];
    }
    return result;
}

void run(int n,vector<vector<double>> &q){
    auto s = omp_get_wtime();
for(int i = 0; i < n - 1; i++) {
        for(int j = i + 1; j < n; j++) {
            {
            cout << "Thread " << omp_get_thread_num() << " reducing row " << j
    << endl;
            q = pivot(q, j, i, i);


            }
        }
    }
    vector<double> result = b_substitution(q, n);


    for(int i = 0; i < result.size(); i++) {
        cout << result[i] << " ";
    }
    cout << endl;
    auto e = omp_get_wtime();
    cout << "Execution time (parallel) : " << e - s << endl;
    s = omp_get_wtime();
for(int i = 0; i < n - 1; i++) {
        for(int j = i + 1; j < n; j++) {
            {
            // cout << "Thread " << omp_get_thread_num() << " reducing row "
<< j << endl;
            q = s_pivot(q, j, i, i);


            }
        }
    }
    vector<double> result_s = s_b_substitution(q, n);

    for(int i = 0; i < result.size(); i++) {
        cout << result_s[i] << " ";
    }
    cout << endl;
    e = omp_get_wtime();
    cout << "Execution time (serial) : " << e - s << endl;

}

int main() {
    int n, m;
    cin >> n >> m;
    omp_set_num_threads(3);

    vector<vector<double>> q(n, vector<double>(m + 1));

    for(int i = 0; i < n; i++) {
        for(int j = 0; j <= m; j++) {
            cin >> q[i][j];
        }
    }
    auto s=omp_get_wtime();
    run(n,q);
    auto e=omp_get_wtime();
    // cout << "executio"




    return 0;
}
```

Output :

```
     3 3
1 -1 1 8
2 3 -1 -2
3 -2 -9 9
Thread 0 reducing row 1
Thread 0 reducing row 2
Thread 0 reducing row 2
4 -3 1
Execution time (parallel) : 0.00900006
4 -3 1
Execution time (serial) : 0.00699997
```

Q2. a) Write a C++ program for serial bubble sort. What is the asymptotic time complexity for this algorithm if N is the size of the array? Test Case: N=10, Unsorted Array: -2 65 0 -4 56 -56 4 98 -3 1 Expected Output: -56 -4 -3 -2 0 1 4 56 65 98 b) Write C++ program for odd-even transposition sort, which is a variant of bubble sort. Verify the algorithm for the above test case. What is the asymptotic time complexity for this algorithm if N is the size of the array? c) Write an OpenMP program with C++ for Parallel odd-even transposition sort, which is a variant of bubble sort. Verify the algorithm for the above test case. What is the asymptotic time complexity for this algorithm if N is the size of the array, and P is the total number of threads used?

Program :

```
#include<bits/stdc++.h>

#include<omp.h>


using namespace std;


void takeInput(vector<int> &nums,int size) {

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

      nums[i] = rand() % 30;

   }

}


void serial_bubbleSort(vector<int> &nums,int size,int &swapCount) {


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

      for(int j = 0;j < size - i - 1; ++j) {

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

            {swapCount ++;

            swap(nums[j + 1],nums[j]);}

      }

   }
```

```
}


vector<int> oddEvenTransposition(vector<int> nums,int size,int
&swapCount_even,int &swapCount_odd) {
    for(int phase = 0;phase < size; ++phase) {
        if(phase % 2 == 0) {
            for(int i = 1;i < size; i += 2) {
                if(nums[i - 1] > nums[i]) {
                    swap(nums[i - 1], nums[i]);
                    swapCount_even ++;
                }
            }
        }
        else {
            for(int i = 2;i < size ; i += 2) {
                if(nums[i - 1] > nums[i]) {
                    swap(nums[i - 1],nums[i]);
                    swapCount_odd ++;
                }
            }
        }
    }
    return nums;
}


vector<int> parallel_oddEven(vector<int> nums, int size,int
&swapCount_even,int &swapCount_odd) {
    #pragma omp parallel default(none)
shared(nums,size,swapCount_even,swapCount_odd)
    {
        for(int phase = 0;phase < size; ++phase) {
        if(phase % 2 == 0) {
            #pragma omp for
            for(int i = 1;i < size; i += 2) {
                if(nums[i - 1] > nums[i]) {
                    swap(nums[i - 1], nums[i]);
                    swapCount_even ++;
                }
            }
        }
        else {
            #pragma omp for
            for(int i = 2;i < size ; i += 2) {
                if(nums[i - 1] > nums[i]) {
                    swap(nums[i - 1],nums[i]);
                    swapCount_odd ++;
                }
            }
        }
    }
    }
    return nums;
}
```

```
void print(vector<int> &nums) {
   for(int i : nums) {
      cout << i << " ,";


   }
   cout << "\n";
}


void run(int size) {
   vector<int> nums(size);
   takeInput(nums,size);
   cout << "\n<------------------------------- For n : " << size << " ---------------------
------------------>\n";
   int swapCount = 0,swapCount_even = 0, swapCount_odd = 0;
   // cout << "Array before sorting:" << endl;
   // print(nums);
   // cout << "array after sorting: " << endl;
   cout << "\nserial odd even :";
   auto s = omp_get_wtime();
   vector<int> temp =
oddEvenTransposition(nums,size,swapCount_even,swapCount_odd);
   auto e = omp_get_wtime();
   // print(temp);
   cout << "\nExecution time(serial odd even) :" << e - s << "\n";
   cout << "\nSwaps in even Phase :" << swapCount_even;
   cout << "\nSwaps in odd Phase :" << swapCount_odd;
   swapCount_even = 0;
```

```
    swapCount_odd = 0;
    cout << "\nparallel odd even transposition:";
    s = omp_get_wtime();
    vector<int> arr =
parallel_oddEven(nums,size,swapCount_even,swapCount_odd);
    e = omp_get_wtime();
    cout << "\nExecution time(parallel odd even) :" << e - s << "\n";
    // print(arr);
    cout << "Swaps in even Phase :" << swapCount_even;
    cout << "\nSwaps in odd Phase :" << swapCount_odd;
    cout << "\nserial sort :" ;
    s = omp_get_wtime();
    serial_bubbleSort(nums,size,swapCount);
    e = omp_get_wtime();
    // print(nums);
    cout << "No. of swaps in serial bubble sort :" << swapCount;
    cout << "\nExecution time(serial bubble sort) :" << e - s << "\n";
    cout << "<--------------------------------------------------------------------------------------------
->\n";
}


int main() {
    vector<int> size = {10,50,100,500};


    omp_set_num_threads(5);
    for(int n : size)
        run(n);
    return 0;
}
```

Output :

```
<------------------------------- For n : 10 -------------------------------------->

serial odd even :
Execution time(serial odd even) :0

Swaps in even Phase :8
Swaps in odd Phase :8
parallel odd even transposition:
Execution time(parallel odd even) :0.00499988
Swaps in even Phase :8
Swaps in odd Phase :8
serial sort :No. of swaps in serial bubble sort :16
Execution time(serial bubble sort) :0
<------------------------------------------------------------------------------------>

<------------------------------- For n : 50 -------------------------------------->

serial odd even :
Execution time(serial odd even) :0

Swaps in even Phase :230
Swaps in odd Phase :228
parallel odd even transposition:
```

```
Execution time(parallel odd even) :0.00300002

Swaps in even Phase :230

Swaps in odd Phase :228

serial sort :No. of swaps in serial bubble sort :458

Execution time(serial bubble sort) :0

<-------------------------------------------------------------------------------->


<------------------------------- For n : 100 -------------------------------------->


serial odd even :

Execution time(serial odd even) :0


Swaps in even Phase :1318

Swaps in odd Phase :1315

parallel odd even transposition:

Execution time(parallel odd even) :0.00300002

Swaps in even Phase :1318

Swaps in odd Phase :1315

serial sort :No. of swaps in serial bubble sort :2633

Execution time(serial bubble sort) :0

<-------------------------------------------------------------------------------->


<------------------------------- For n : 500 -------------------------------------->


serial odd even :

Execution time(serial odd even) :0.000999928
```

```
Swaps in even Phase :31614

Swaps in odd Phase :31582

parallel odd even transposition:

Execution time(parallel odd even) :0.0209999

Swaps in even Phase :31366

Swaps in odd Phase :31419

serial sort :No. of swaps in serial bubble sort :63196

Execution time(serial bubble sort) :0.000999928

<---------------------------------------------------------------------------------->
```