Indian Institute of Information Technology, Design and Manufacturing, Kancheepuram

High Performance Computing Practice - COM403P

EXPERIMENT 4

Name	Roll Number
N Sree Dhyuti	CED19I027

---

1. Histogram Equalization

OBJECTIVE:

Based on profiling results, apply OpenMP for Histogram Equalization

---

Serial Code:

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <math.h>
#include <vector>

using namespace std;

int array[1000][1000];
int arr[1000][1000];
vector<int> pf;
vector<float> pr;
vector<float> cpr;
vector<int> fin;

// Function that creates a frequency array of pixels in the image
vector<int> create_frequency_array(int numrows, int numcols){
    for(int i = 0; i < 257; i++){
        pf.push_back(0);
```

```
    }

    for(int row = 1; row <= numrows; row++){
        for(int col = 1; col <= numcols; col++){
        pf[256]++;
        pf[array[row][col]]++;
    }
    }
    return pf;
}

// Function to find individual probabilities of occurence of each of 256
values of pixel
vector<float> individual_probabilities(vector<int> pixel_frequency, int
num_pixels){
    for(int i = 0; i < 256; i++){
        pr.push_back(0);
    }

    for(int i = 0; i < 256; i++){
        pr[i] = ((float)pixel_frequency[i])/((float)num_pixels);
    }

    return pr;
}

// Function to find cumulative probability of each of 256 values of pixel
vector<float> cumulative_probability(vector<float> pr){

    for(int i = 0; i < 256; i++){
        cpr.push_back(0);
    }

    cpr[0] = pr[0];
    for(int i = 1; i < 256; i++){
        cpr[i] = pr[i] + cpr[i-1];
    }
    return cpr;
}

// Function to calculate C(r) X (L-1)
vector<int> cpr_into_max_pixel(vector<float> cpr){
    for(int i = 0; i < 256; i++){
        fin.push_back(0);
    }
    for(int i = 0; i < 256; i++){
        fin[i] = round(cpr[i]*255);
    }
    return fin;
```

```
}

// Function to update with new pixel values
void final_step(int numrows, int numcols, vector<int> finall){
    for(int row = 1; row <= numrows; row++){
    for(int col = 1; col <= numcols; col++){
        arr[row][col] = finall[array[row][col]];
    }
    }
}

int main()
{
    int row = 0, col = 0, numrows = 0, numcols = 0,MAX=0;
    ifstream infile("Images/casablanca.ascii.pgm");
    stringstream ss;
    string inputLine = "";

    // First line : version
    getline(infile,inputLine);
    if(inputLine.compare("P2") != 0) cerr << "Version error" << endl;
    else cout << "Version : " << inputLine << endl;

    // Continue with a stringstream
    ss << infile.rdbuf();

    // Secondline : size of image
    ss >> numcols >> numrows >> MAX;

    //print total number of rows, columns and maximum intensity of image
    cout << numcols << " columns and " << numrows << " rows" <<endl<<"Maximum
Intensity "<< MAX <<endl;


    //Initialize a new array of same size of image with 0
    for(row = 0; row <= numrows; ++row){
        array[row][0]=0;
        //arr[row][0] = 0;
        for(col = 0; col <= numcols; col++){
            array[0][col] = 0;
            //arr[0][col] = 0;
        }
    }

    // Following lines : data
    for(row = 1; row <= numrows; ++row)
    {
        for (col = 1; col <= numcols; ++col)
        {
```

```
            //original data store in new array
            ss >> array[row][col];
        }
    }


    // Histogram Equalization begins

    // Step 1: Find frequencies of each pixel value
    vector<int> pixel_frequency = create_frequency_array(numrows, numcols);
    int num_pixels = pixel_frequency[256];

    // Step 2: P(r)
    vector<float> pr = individual_probabilities(pixel_frequency, num_pixels);

    // Step 3: Cumulative Frequency
    vector<float> cpr = cumulative_probability(pr);

    // Step 4: C(r) X (L-1)
    vector<int> finall = cpr_into_max_pixel(cpr);


    // Step 5: Updare new image with respective updation
    final_step(numrows, numcols, finall);


    ofstream outfile;

    //new file open to store the output image
    outfile.open("AfterHistogramEqualization.ascii.pgm");
    outfile<<"P2"<<endl;
    outfile<<numcols<<" "<<numrows<<endl;
    outfile<<"255"<<endl;

    for(row = 1; row <= numrows; ++row)
    {
        for (col = 1; col <= numcols; ++col)
        {
            //store resultant pixel values to the output file
            outfile << arr[row][col]<<" ";
        }
    }

    outfile.close();
    infile.close();
    return 0 ;
}
```

Output:

```
anuhya@anuhya-HP-Laptop-15q-ds0xxx:~/Desktop/hpc/dhyuti/week3/CED19I027_Serial_Code/PROJECT$ g++ -fopenmp  histogram_equalization.cpp && export OMP_NUM_THREADS=1 && ./a.out
Version : P2
460 columns and 360 rows
Maximum Intensity 255
Time: 0.0117262
anuhya@anuhya-HP-Laptop-15q-ds0xxx:~/Desktop/hpc/dhyuti/week3/CED19I027_Serial_Code/PROJECT$
```

Parallel Code:

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <math.h>
#include <vector>
#include "omp.h"

using namespace std;

int array[1000][1000];
int arr[1000][1000];
vector<int> pf;
vector<float> pr;
vector<float> cpr;
vector<int> fin;

// Function that creates a frequency array of pixels in the image

vector<int> create_frequency_array(int numrows, int numcols){

    #pragma omp parallel
    {
    #pragma omp for
    for(int i = 0; i < 257; i++){
        pf.push_back(0);
    }

    }

    #pragma omp parallel
    {
    #pragma omp for
    for(int row = 1; row <= numrows; row++){
        for(int col = 1; col <= numcols; col++){
        pf[256]++;
        pf[array[row][col]]++;
    }
    }
    }
    return pf;
```

```
}

// Function to find individual probabilities of occurence of each of 256
values of pixel
vector<float> individual_probabilities(vector<int> pixel_frequency, int
num_pixels){


    for(int i = 0; i < 256; i++){
        pr.push_back(0);
    }


    for(int i = 0; i < 256; i++){
        pr[i] = ((float)pixel_frequency[i])/((float)num_pixels);
    }

    return pr;
}

// Function to find cumulative probability of each of 256 values of pixel
vector<float> cumulative_probability(vector<float> pr){


    for(int i = 0; i < 256; i++){
        cpr.push_back(0);
    }


    cpr[0] = pr[0];

    for(int i = 1; i < 256; i++){
        cpr[i] = pr[i] + cpr[i-1];
    }

    return cpr;
}

// Function to calculate C(r) X (L-1)
vector<int> cpr_into_max_pixel(vector<float> cpr){


    for(int i = 0; i < 256; i++){
        fin.push_back(0);
    }


    for(int i = 0; i < 256; i++){
        fin[i] = round(cpr[i]*255);
```

```
    }
    return fin;
}

// Function to update with new pixel values
void final_step(int numrows, int numcols, vector<int> finall){

    #pragma omp parallel
    {
    #pragma omp for
    for(int row = 1; row <= numrows; row++){
    for(int col = 1; col <= numcols; col++){
        arr[row][col] = finall[array[row][col]];
    }
    }
    }
}

int main()
{
    int row = 0, col = 0, numrows = 0, numcols = 0,MAX=0;
    ifstream infile("Images/casablanca.ascii.pgm");
    stringstream ss;
    string inputLine = "";

    // First line : version
    getline(infile,inputLine);
    if(inputLine.compare("P2") != 0) cerr << "Version error" << endl;
    else cout << "Version : " << inputLine << endl;

    // Continue with a stringstream
    ss << infile.rdbuf();

    // Secondline : size of image
    ss >> numcols >> numrows >> MAX;

    //print total number of rows, columns and maximum intensity of image
    cout << numcols << " columns and " << numrows << " rows" <<endl<<"Maximum
Intensity "<< MAX <<endl;


    //Initialize a new array of same size of image with 0
    for(row = 0; row <= numrows; ++row){
        array[row][0]=0;
        //arr[row][0] = 0;
        for(col = 0; col <= numcols; col++){
            array[0][col] = 0;
            //arr[0][col] = 0;
        }
```

```
}

// Following lines : data
for(row = 1; row <= numrows; ++row)
{
    for (col = 1; col <= numcols; ++col)
    {
        //original data store in new array
        ss >> array[row][col];
    }
}


// Histogram Equalization begins
    double wallclock_initial = omp_get_wtime();
// Step 1: Find frequencies of each pixel value
vector<int> pixel_frequency = create_frequency_array(numrows, numcols);
int num_pixels = pixel_frequency[256];

// Step 2: P(r)
vector<float> pr = individual_probabilities(pixel_frequency, num_pixels);

// Step 3: Cumulative Frequency
vector<float> cpr = cumulative_probability(pr);

// Step 4: C(r) X (L-1)
vector<int> finall = cpr_into_max_pixel(cpr);


// Step 5: Updare new image with respective updation
final_step(numrows, numcols, finall);


ofstream outfile;

//new file open to store the output image
outfile.open("AfterHistogramEqualization.ascii.pgm");
outfile<<"P2"<<endl;
outfile<<numcols<<" "<<numrows<<endl;
outfile<<"255"<<endl;

for(row = 1; row <= numrows; ++row)
{
    for (col = 1; col <= numcols; ++col)
    {
        //store resultant pixel values to the output file
        outfile << arr[row][col]<<" ";
    }
}
```

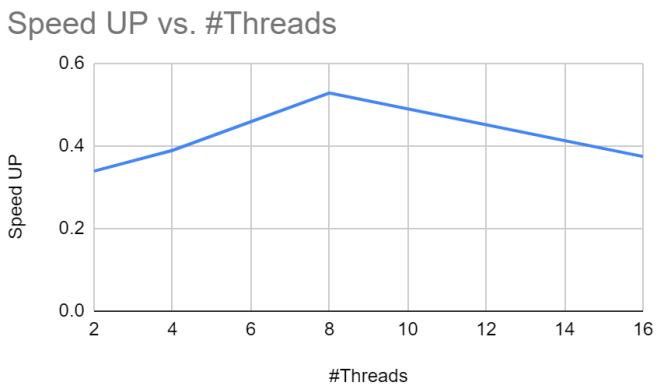
```
    double wallclock_final = omp_get_wtime();
    cout<<"Time: "<<wallclock_final-wallclock_initial<<endl;
    outfile.close();
    infile.close();
    return 0 ;
}
```

Output:

```
anuhya@anuhya-HP-Laptop-15q-ds0xxx:~/Desktop/hpc/dhyuti/week4/CED19I027_Serial_Code/PROJECT$ g++ -fopenmp  histogram_equalization.cpp && export OMP_NUM_THREADS=2 &&./a.out
Version : P2
460 columns and 360 rows
Maximum Intensity 255
Time: 0.0153344
anuhya@anuhya-HP-Laptop-15q-ds0xxx:~/Desktop/hpc/dhyuti/week4/CED19I027_Serial_Code/PROJECT$
```

```
anuhya@anuhya-HP-Laptop-15q-ds0xxx:~/Desktop/hpc/dhyuti/week4/CED19I027_Serial_Code/PROJECT$ g++ -fopenmp  histogram_equalization.cpp && export OMP_NUM_THREADS=4 &&./a.out
Version : P2
460 columns and 360 rows
Maximum Intensity 255
Time: 0.0133753
anuhya@anuhya-HP-Laptop-15q-ds0xxx:~/Desktop/hpc/dhyuti/week4/CED19I027_Serial_Code/PROJECT$
```

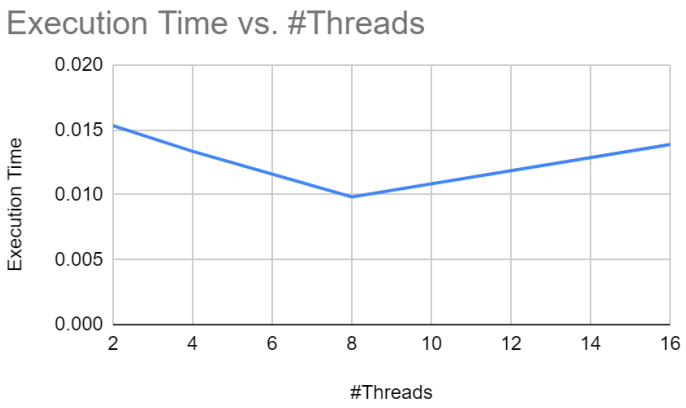
Speedup V/S Number of Processors:



Inference:

It can be inferred from the above graph that Speedup value increases till 8 threads, and then it continues to decrease.

Execution Time V/S Number of Threads:



Inference:
Similar to the previous graph, it can be seen that we attain minimum execution time when we choose 8 threads to execute the program.
This means choosing 8 threads for this program will give us maximum time efficiency.

Parallelization Factor (f):

#Threads	Execution Time	Speed UP	Efficiency (in %)	f
1	0.01172	1	100	n/a
2	0.0153344	0.3391720576	16.95860288	0.6167918089
4	0.0133573	0.3893750983	9.734377457	-0.1862684869
8	0.0098456	0.5282562769	6.603203461	0.1827791321
16	0.01388	0.3747118156	2.341948847	-0.1965870307

Inference:
It can be noticed from the pattern of ‘f’ values that taking 2 threads will give utmost parallelization. Choosing 2 threads to execute this program will give the most efficient parallelization.

THE END
