1. **Histogram Equalization**

**OBJECTIVE:**

Perform various profiling techniques on the code to obtain important insights from the same.

**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();**

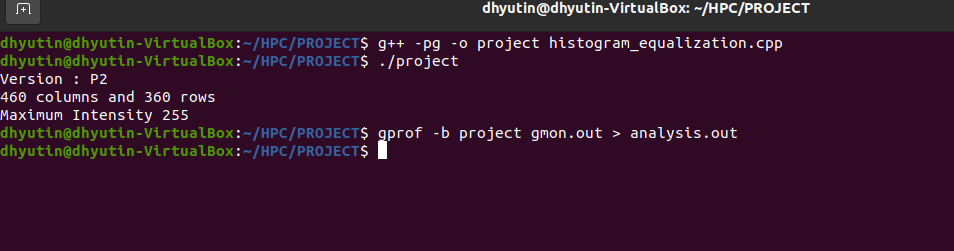
**return 0 ;**

**}**

**Functional Profiling - gprof**

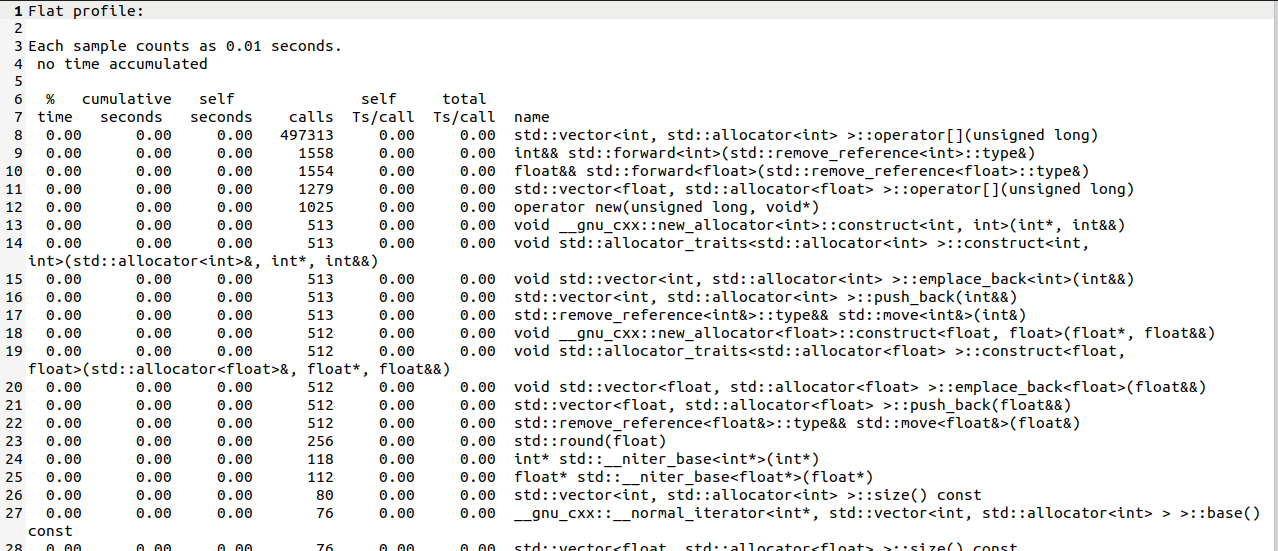
Commands used:

* g++ -pd -o project histogram\_equalization.cpp
* ./project
* grpof -b project gmon.out > analysis.out

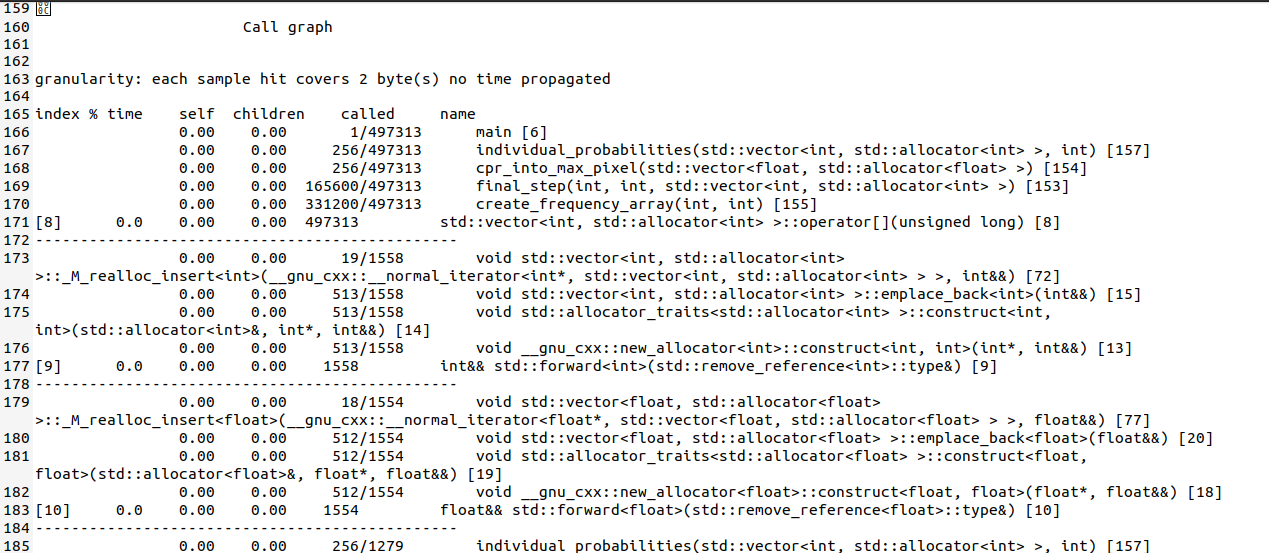
****

***analysis.out***

***Flat Profile:***

******

***Call Graph:***



**Functional Profiling Analysis**

Functional profiling is done to find out statistics about the function-wise spilt contribution in time usage by functions available in the function.

In my code, I have divided the functionality of my code into 5 functions.

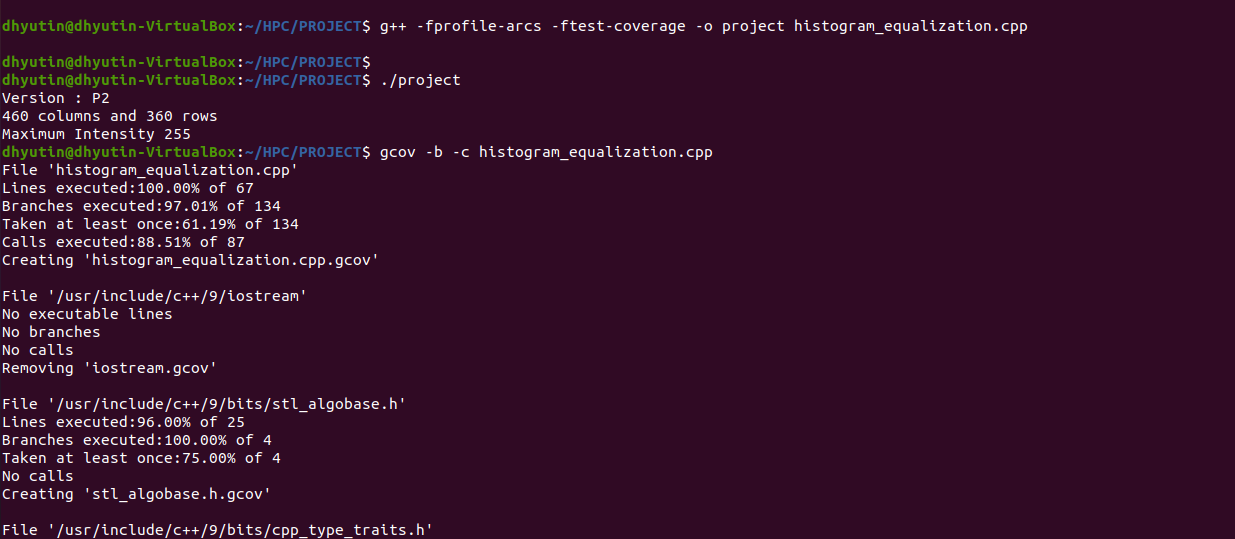
The “create\_frequency\_array()” function is the most expensive as it is called the most number of times and involves most work in my code.

It can also be concluded from this information that we can consider parallelizing the functions with most occurrences to optimize time.

**Linewise Profiling - gcov**

Commands used:

* g++ -fprofile-arcs -ftest-coverage -o project histogram\_equalization.cpp
* ./project
* gcov -b -c histogram\_equalization.cpp



File 'histogram\_equalization.cpp'

Lines executed:100.00% of 67

Branches executed:97.01% of 134

Taken at least once:61.19% of 134

Calls executed:88.51% of 87

Creating 'histogram\_equalization.cpp.gcov'

File '/usr/include/c++/9/iostream'

No executable lines

No branches

No calls

Removing 'iostream.gcov'

File '/usr/include/c++/9/bits/stl\_algobase.h'

Lines executed:96.00% of 25

Branches executed:100.00% of 4

Taken at least once:75.00% of 4

No calls

Creating 'stl\_algobase.h.gcov'

File '/usr/include/c++/9/bits/cpp\_type\_traits.h'

Lines executed:100.00% of 2

No branches

No calls

Creating 'cpp\_type\_traits.h.gcov'

File '/usr/include/c++/9/bits/stl\_iterator.h'

Lines executed:50.00% of 16

No branches

No calls

Creating 'stl\_iterator.h.gcov'

File '/usr/include/c++/9/bits/stl\_uninitialized.h'

Lines executed:83.33% of 18

No branches

No calls

Creating 'stl\_uninitialized.h.gcov'

File '/usr/include/c++/9/ext/new\_allocator.h'

Lines executed:81.25% of 16

No branches

No calls

Creating 'new\_allocator.h.gcov'

File '/usr/include/c++/9/bits/alloc\_traits.h'

Lines executed:83.33% of 12

No branches

No calls

Creating 'alloc\_traits.h.gcov'

File '/usr/include/c++/9/bits/stl\_vector.h'

Lines executed:98.68% of 76

No branches

No calls

Creating 'stl\_vector.h.gcov'

File '/usr/include/c++/9/bits/stl\_iterator\_base\_funcs.h'

Lines executed:0.00% of 5

No branches

Calls executed:0.00% of 2

Creating 'stl\_iterator\_base\_funcs.h.gcov'

File '/usr/include/c++/9/bits/stl\_iterator\_base\_types.h'

Lines executed:0.00% of 2

No branches

No calls

Creating 'stl\_iterator\_base\_types.h.gcov'

File '/usr/include/c++/9/ext/type\_traits.h'

Lines executed:0.00% of 2

No branches

No calls

Creating 'type\_traits.h.gcov'

File '/usr/include/c++/9/bits/stl\_construct.h'

Lines executed:100.00% of 7

No branches

No calls

Creating 'stl\_construct.h.gcov'

File '/usr/include/c++/9/bits/basic\_string.tcc'

Lines executed:0.00% of 13

Branches executed:0.00% of 22

Taken at least once:0.00% of 22

Calls executed:0.00% of 13

Creating 'basic\_string.tcc.gcov'

File '/usr/include/c++/9/bits/vector.tcc'

Lines executed:69.05% of 42

No branches

No calls

Creating 'vector.tcc.gcov'

File '/usr/include/c++/9/bits/move.h'

Lines executed:100.00% of 4

No branches

No calls

Creating 'move.h.gcov'

File '/usr/include/c++/9/bits/basic\_string.h'

Lines executed:0.00% of 6

Branches executed:0.00% of 4

Taken at least once:0.00% of 4

Calls executed:0.00% of 2

Creating 'basic\_string.h.gcov'

File '/usr/include/c++/9/bits/allocator.h'

Lines executed:100.00% of 4

No branches

No calls

Creating 'allocator.h.gcov'

File '/usr/include/c++/9/ext/alloc\_traits.h'

Lines executed:100.00% of 2

No branches

No calls

Creating 'alloc\_traits.h.gcov'

File '/usr/include/c++/9/cmath'

Lines executed:100.00% of 2

No branches

No calls

Creating 'cmath.gcov'

File '/usr/include/c++/9/bits/char\_traits.h'

Lines executed:0.00% of 2

No branches

No calls

Creating 'char\_traits.h.gcov'

File '/usr/include/c++/9/new'

Lines executed:100.00% of 2

No branches

No calls

Creating 'new.gcov'

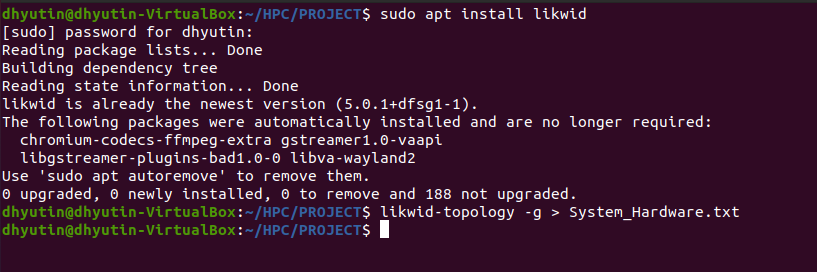
**Linewise Profiling Analysis**

Linewise profiling gives a much better insight about how the code is working as it gives a line-by-line analysis of the code instead of a function-wise analysis. It can be noticed that the main time-consuming tasks are lying in the “for” loops available in the codes irrespective of the function. This is an indication that we can try parallelizing the “for” loops in our code.

**Likwid Profiling**

Commands used:

* sudo apt install likwid
* likwid-topology -g



***System\_Hardware.txt***

**--------------------------------------------------------------------------------**

**CPU name: Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz**

**CPU type: Intel Kabylake processor**

**CPU stepping: 10**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Hardware Thread Topology**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Sockets: 1**

**Cores per socket: 4**

**Threads per core: 1**

**--------------------------------------------------------------------------------**

**HWThread Thread Core Socket Available**

**0 0 0 0 \***

**1 0 1 0 \***

**2 0 2 0 \***

**3 0 3 0 \***

**--------------------------------------------------------------------------------**

**Socket 0: ( 0 1 2 3 )**

**--------------------------------------------------------------------------------**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Cache Topology**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Level: 1**

**Size: 32 kB**

**Cache groups: ( 0 ) ( 1 ) ( 2 ) ( 3 )**

**--------------------------------------------------------------------------------**

**Level: 2**

**Size: 256 kB**

**Cache groups: ( 0 ) ( 1 ) ( 2 ) ( 3 )**

**--------------------------------------------------------------------------------**

**Level: 3**

**Size: 8 MB**

**Cache groups: ( 0 ) ( 1 ) ( 2 ) ( 3 )**

**--------------------------------------------------------------------------------**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**NUMA Topology**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**NUMA domains: 1**

**--------------------------------------------------------------------------------**

**Domain: 0**

**Processors: ( 0 1 2 3 )**

**Distances: 10**

**Free memory: 1757.24 MB**

**Total memory: 4965.58 MB**

**--------------------------------------------------------------------------------**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Graphical Topology**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Socket 0:**

**+---------------------------------------------+**

**| +--------+ +--------+ +--------+ +--------+ |**

**| | 0 | | 1 | | 2 | | 3 | |**

**| +--------+ +--------+ +--------+ +--------+ |**

**| +--------+ +--------+ +--------+ +--------+ |**

**| | 32 kB | | 32 kB | | 32 kB | | 32 kB | |**

**| +--------+ +--------+ +--------+ +--------+ |**

**| +--------+ +--------+ +--------+ +--------+ |**

**| | 256 kB | | 256 kB | | 256 kB | | 256 kB | |**

**| +--------+ +--------+ +--------+ +--------+ |**

**| +--------+ +--------+ +--------+ +--------+ |**

**| | 8 MB | | 8 MB | | 8 MB | | 8 MB | |**

**| +--------+ +--------+ +--------+ +--------+ |**

**LikwidProfiling Analysis**

Likwid profiling gives an insight into the hardware and software specifications of my computer/Virtual Box. It is noticeable that I am using an Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz Core which contains one socket with 4 cores and each core contains exactly one thread each. Following this, information about the cache availability was also provided, giving us an estimate of to what extent we can parallelize the given code.

**THE END**