PROFILING ON
FINITE IMPULSE RESPONSE

High Performance Computing Project Report

Problem Statement: Parallel simulation of moving average finite impulse response filter

Faculty guide : Dr. Noor Mahammad

By,

PALETI KRISHNASAI

CED18I039

INTRODUCTION

In signal processing, a **finite impulse response** (FIR) **filter** is a filter whose impulse response (or response to any finite length input) is of *finite* duration, because it settles to zero in finite time. This is in contrast to infinite impulse response (IIR) filters, which may have internal feedback and may continue to respond indefinitely (usually decaying).

The impulse response (that is, the output in response to a Kronecker delta input) of an Nth-order discrete-time FIR filter lasts exactly *N + 1* samples (from first nonzero element through last nonzero element) before it then settles to zero.

For a causal discrete-time FIR filter of order *N*, each value of the output sequence is a weighted sum of the most recent input values:

$$\begin{aligned} y[n] &= b_0 x[n] + b_1 x[n-1] + \cdots + b_N x[n-N] \\ &= \sum_{i=0}^{N} b_i \cdot x[n-i], \end{aligned}$$

- $x[n]$ is the input signal,
- $y[n]$ is the output signal,
- N is the filter order; an N^{th}-order filter has $N+1$ terms on the right-hand side
- b_i is the value of the impulse response at the *i*'th instant for $0 \le i \le N$ of an N^{th}-order FIR filter. If the filter is a direct form FIR filter then b_i is also a coefficient of the filter.

MOVING AVERAGE FIR FILTER ANALYSIS

A moving average filter is a very simple FIR filter. It is sometimes called a boxcar filter, especially when followed by decimation. The filter coefficients, $b_0, \ldots, b_N$, are found via the following equation:

$$b_i = \frac{1}{N+1}$$

To provide a more specific example, we select the filter order:

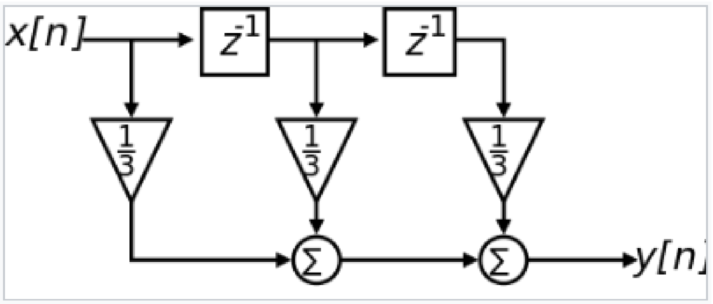
$$N = 2$$

The impulse response of the resulting filter is:

$$h[n] = \frac{1}{3}\delta[n] + \frac{1}{3}\delta[n-1] + \frac{1}{3}\delta[n-2]$$

The block diagram below shows the second-order moving-average filter discussed below. The transfer function is:

$$H(z) = \frac{1}{3} + \frac{1}{3}z^{-1} + \frac{1}{3}z^{-2} = \frac{1}{3}\frac{z^2+z+1}{z^2}.$$



Block diagram of a simple FIR filter (second-order/3-tap filter in this case, implementing a moving average smoothing filter)

SERIAL CODE

```
/*
Author : Paleti Krishnasai CED18I039
Simulation of N-order moving average FIR filter
   N : order of the filter
   n : instance
   filter equation : output_signal[n] = (input_signal[n-1] +
input_signal[n] + input_signal[n1+1]) / (N+1)
*/

// g++ Norder_gl.cpp -O0 -fopenmp -lGL -lGLU -lglut -lGLEW -o Norder_gl


#include <bits/stdc++.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <GL/glew.h>
#include <GL/freeglut.h>
#include <GL/glut.h>
#include <math.h>
#include <time.h>
#include <omp.h>

using namespace std;

#define N 6 // filter order
#define size 200000 // size of 1-D signal

long double input_signal[size], output_signal[size]; // store in heap

void changeViewPort(int w, int h)
{
   if(w>=h)
       glViewport(w/2-h/2, 0, h, h);
   else
       glViewport(0, h/2-w/2, w, w);
}
```

```
void myinit(void)
{
   glClearColor(0.8,0.8,0.8,0.0);
   glMatrixMode(GL_PROJECTION);
   gluOrtho2D (0.0,200000.0,0.0,10000.0);
   glMatrixMode(GL_MODELVIEW);
}

void filter()
{
   myinit();
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   float startTime, endTime,execTime;
   long double random_input;

   // initialize input signal using random values
   for (int i = 0; i < size; i++)
   {
       random_input = static_cast <long double> (rand());
       input_signal[i] =random_input;
   }
   startTime = omp_get_wtime();
   // filter equation

       output_signal[0] = (input_signal[0] + input_signal[1]) / (N+1); //
edge case 1
       for (int i = 1; i < size-1; i++)
       {
           output_signal[i] = (input_signal[i-1] + input_signal[i] +
input_signal[i+1]) / (N+1);
               glColor3f(0,0,0);
               glPointSize(3);
               glBegin(GL_POINTS);
               glVertex2f(i,(output_signal[i])/100000);
               glEnd();
               glFlush();
               glutSwapBuffers();
       }
       output_signal[size - 1] = (input_signal[size - 2] +
input_signal[size - 1]) / (N+1); // edge case 2

   endTime = omp_get_wtime();
   execTime = endTime - startTime;
   cout << execTime << endl;
   glutLeaveMainLoop();
}


int main (int argc,char** argv)
{
   srand (time(0));

   glutInit(&argc,argv);
   glutInitDisplayMode(GLUT_SINGLE);
   glutInitWindowSize(1000, 1000);
   glutInitWindowPosition(200, 200);


   glutCreateWindow("CED18I039");
   glutDisplayFunc(filter);


   glutReshapeFunc(changeViewPort);
   GLenum err = glewInit();
   if (GLEW_OK != err) {
       fprintf(stderr, "GLEW error");
       return 1;
   }
   glutMainLoop();


   return 0;
}
```

This code simulates the N order moving order FIR filter by plotting every point generated filter equation in real time on a glut canvas using OpenGL.

If the Filter is simulated the same way it works in the real world , then each of the input instances would have to be initialized one after the other and calculate the filter output in the same way. This would lead to few output instances to miss as there is an overlap in the input instances required for a particular output instance .

The points and color are added by `glColor3fglColor3f` and `glVertex3f`. `glFlush` and `glutSwapBuffers` flushes the display buffer causing it to redisplay the updated graph for a real-time visualization

```
startTime = omp_get_wtime();
endTime = omp_get_wtime();
execTime = endTime - startTime;
cout << execTime << endl;
```

These parts of the code calculate the wall clock time of the serial code which require the `#include <omp.h>` header file.

Furthermore the output instances are scaled down by a factor of 10^5 while plotting on the glut canvas.

Due to the lack of authentic open source datasets of 1-D signals, the student has simulated the filter using random values generated using rand() and srand() API's.

PROFILING

In a performance engineering context performance profiling means to relate performance metric measurements to source code execution. Data sources are typically either operating systems, execution environments or measurement facilities in the hardware.

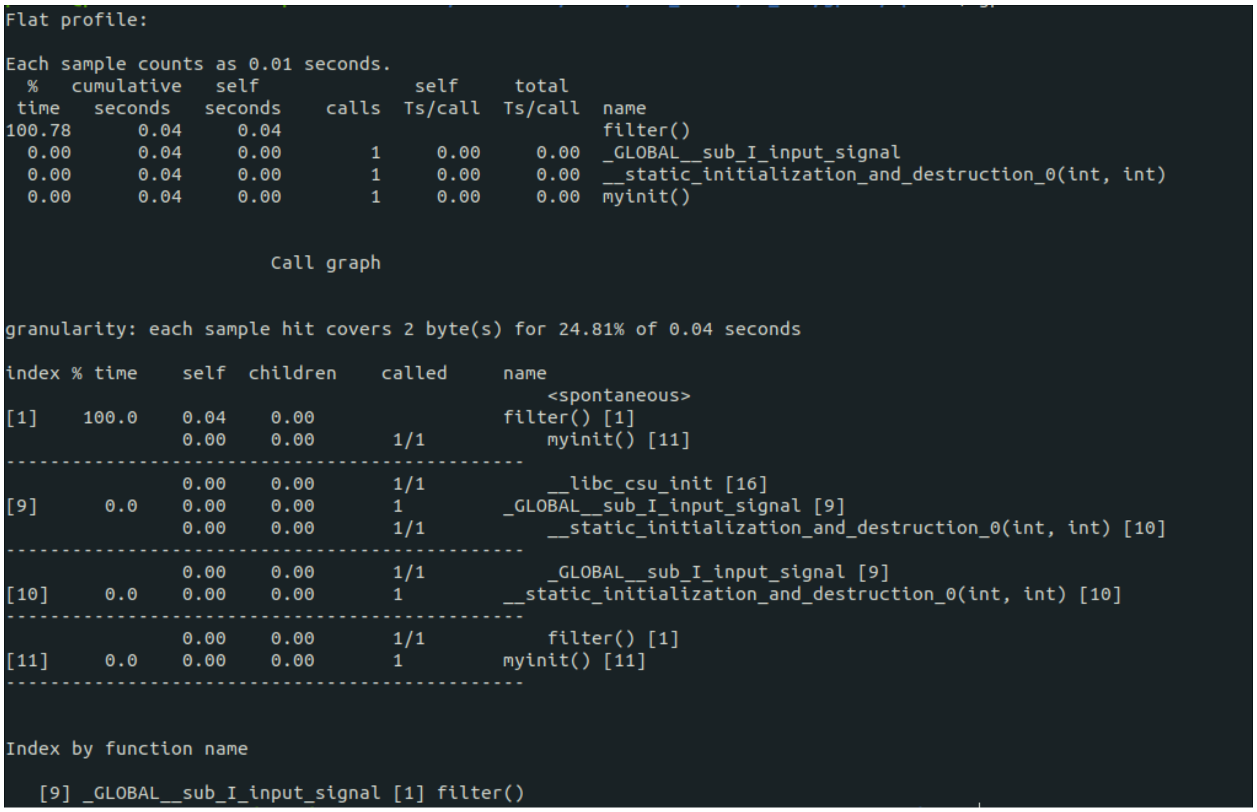
Tools used for profiling:

- Function-based profiling using gprof
- Line-based profiling using gcov
- Hardware profiling using likwid

GPROF

Gprof is used to get the frequency of each function calls, to determine the computation intensive function.

- Enable profiling during the compilation (-pg)
- Execute the binary to generate the profiling data
- The data is stored in a file "gmon.out" in the root directory.
- Use gprof -b <executable file>



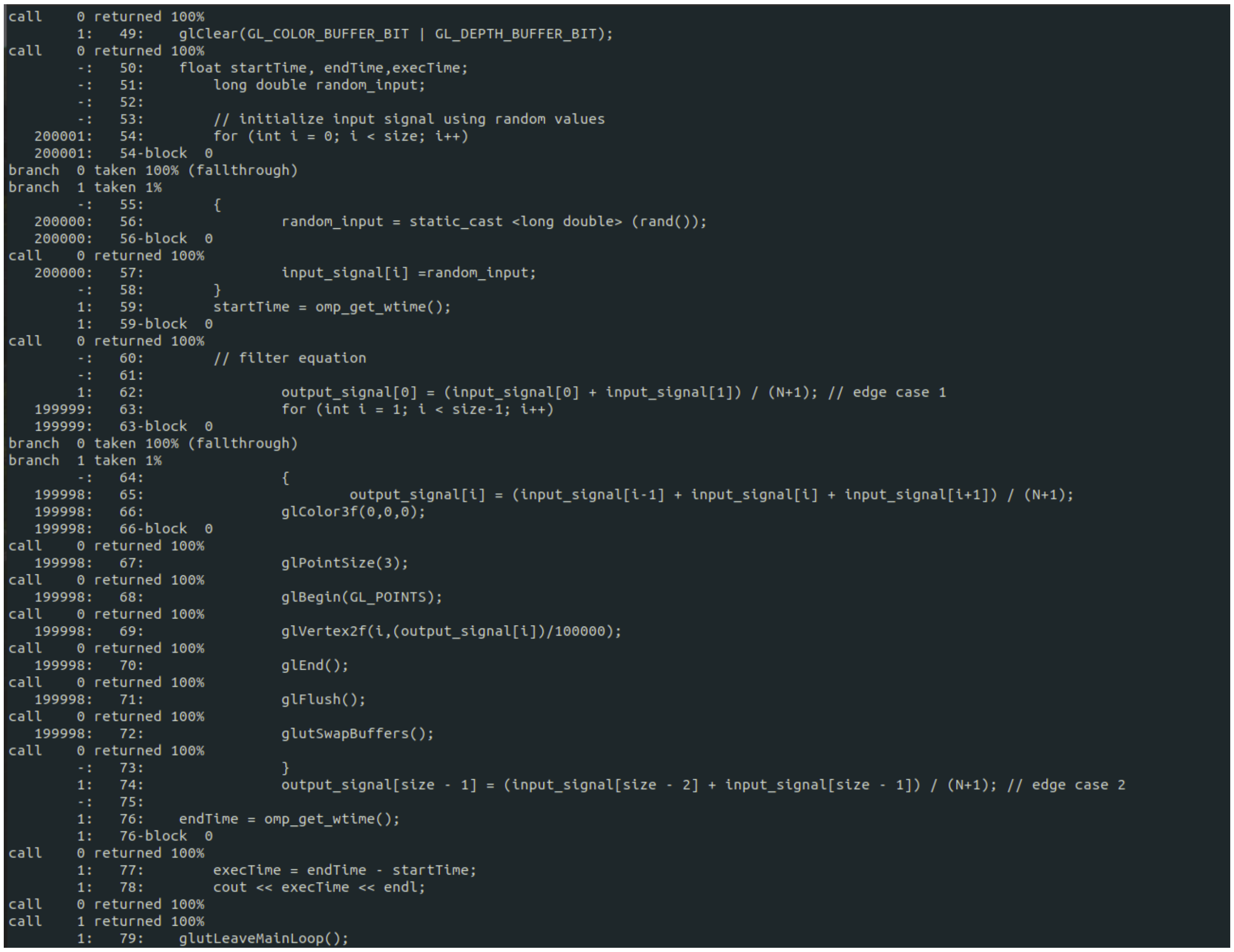
```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
100.78      0.04     0.04                             filter()
  0.00      0.04     0.00        1     0.00     0.00  _GLOBAL__sub_I_input_signal
  0.00      0.04     0.00        1     0.00     0.00  __static_initialization_and_destruction_0(int, int)
  0.00      0.04     0.00        1     0.00     0.00  myinit()


                        Call graph


granularity: each sample hit covers 2 byte(s) for 24.81% of 0.04 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.04    0.00                 filter() [1]
                0.00    0.00       1/1           myinit() [11]
-----------------------------------------------
                0.00    0.00       1/1           __libc_csu_init [16]
[9]      0.0    0.00    0.00       1         _GLOBAL__sub_I_input_signal [9]
                0.00    0.00       1/1           __static_initialization_and_destruction_0(int, int) [10]
-----------------------------------------------
                0.00    0.00       1/1           _GLOBAL__sub_I_input_signal [9]
[10]     0.0    0.00    0.00       1         __static_initialization_and_destruction_0(int, int) [10]
-----------------------------------------------
                0.00    0.00       1/1           filter() [1]
[11]     0.0    0.00    0.00       1         myinit() [11]
-----------------------------------------------


Index by function name

   [9] _GLOBAL__sub_I_input_signal [1] filter()
```

As shown in the outputs, all the functions are being called exactly once.

GCOV

Gprof is used to get the frequency of each line, to determine the computation intensive section.

- Enable profiling during the compilation (-fprofile-arcs -ftest-coverage)
- Execute the binary to generate the profiling data
- Run gcov to generate the coverage data (gcov -b -c <file-name>.c)
- The data is stored in a file "<file-name>.c.gcov" in the root directory.

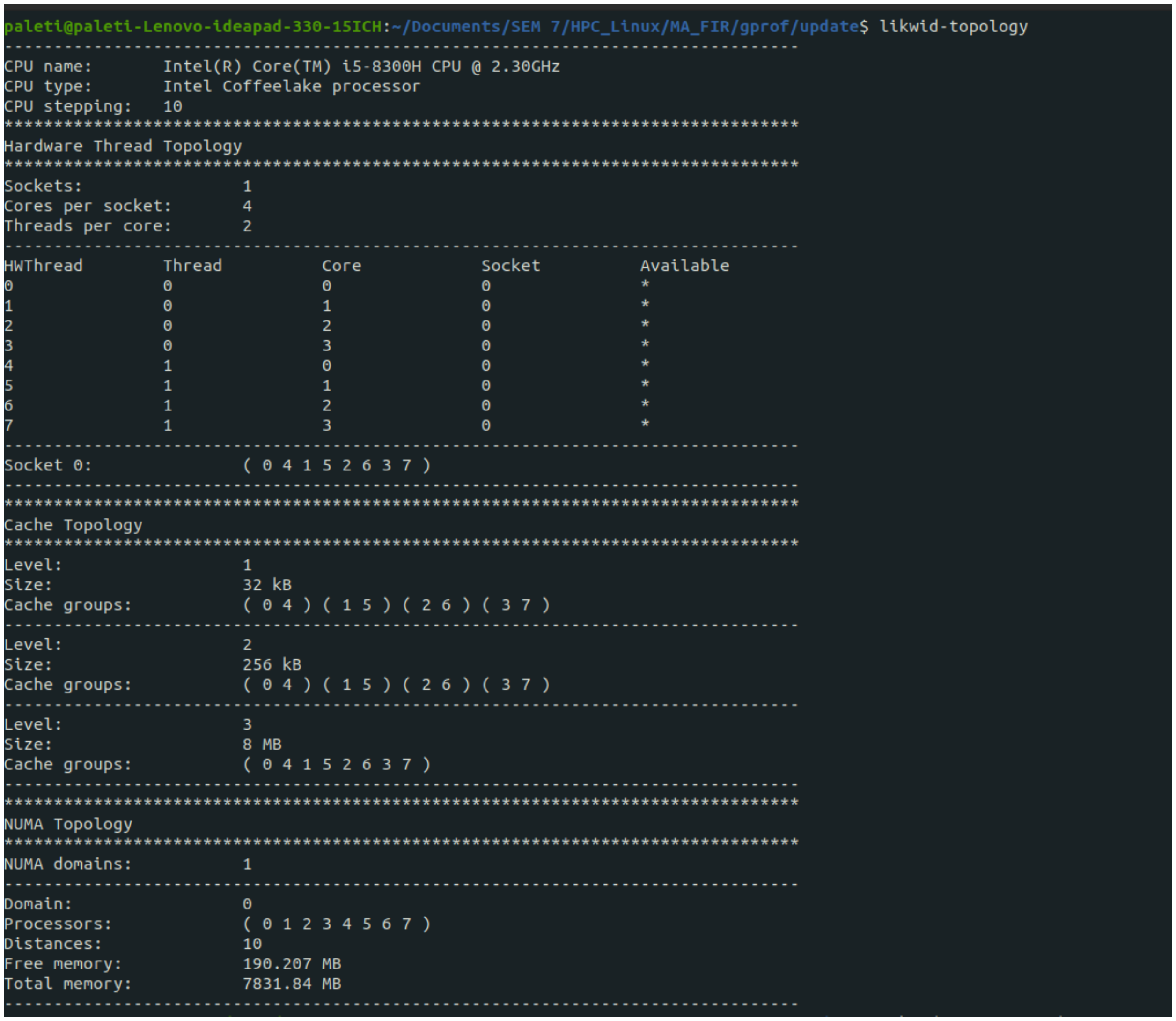


```
call    0 returned 100%
        1:   49:    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
call    0 returned 100%
        -:   50:    float startTime, endTime,execTime;
        -:   51:        long double random_input;
        -:   52:
        -:   53:        // initialize input signal using random values
   200001:   54:        for (int i = 0; i < size; i++)
   200001:   54-block  0
branch  0 taken 100% (fallthrough)
branch  1 taken 1%
        -:   55:        {
   200000:   56:                random_input = static_cast <long double> (rand());
   200000:   56-block  0
call    0 returned 100%
   200000:   57:                input_signal[i] =random_input;
        -:   58:        }
        1:   59:        startTime = omp_get_wtime();
        1:   59-block  0
call    0 returned 100%
        -:   60:        // filter equation
        -:   61:
        1:   62:                output_signal[0] = (input_signal[0] + input_signal[1]) / (N+1); // edge case 1
   199999:   63:                for (int i = 1; i < size-1; i++)
   199999:   63-block  0
branch  0 taken 100% (fallthrough)
branch  1 taken 1%
        -:   64:                {
   199998:   65:                        output_signal[i] = (input_signal[i-1] + input_signal[i] + input_signal[i+1]) / (N+1);
   199998:   66:                glColor3f(0,0,0);
   199998:   66-block  0
call    0 returned 100%
   199998:   67:                glPointSize(3);
call    0 returned 100%
   199998:   68:                glBegin(GL_POINTS);
call    0 returned 100%
   199998:   69:                glVertex2f(i,(output_signal[i])/100000);
call    0 returned 100%
   199998:   70:                glEnd();
call    0 returned 100%
   199998:   71:                glFlush();
call    0 returned 100%
   199998:   72:                glutSwapBuffers();
call    0 returned 100%
        -:   73:                }
        1:   74:                output_signal[size - 1] = (input_signal[size - 2] + input_signal[size - 1]) / (N+1); // edge case 2
        -:   75:
        1:   76:    endTime = omp_get_wtime();
        1:   76-block  0
call    0 returned 100%
        1:   77:        execTime = endTime - startTime;
        1:   78:        cout << execTime << endl;
call    0 returned 100%
call    1 returned 100%
        1:   79:    glutLeaveMainLoop();
```

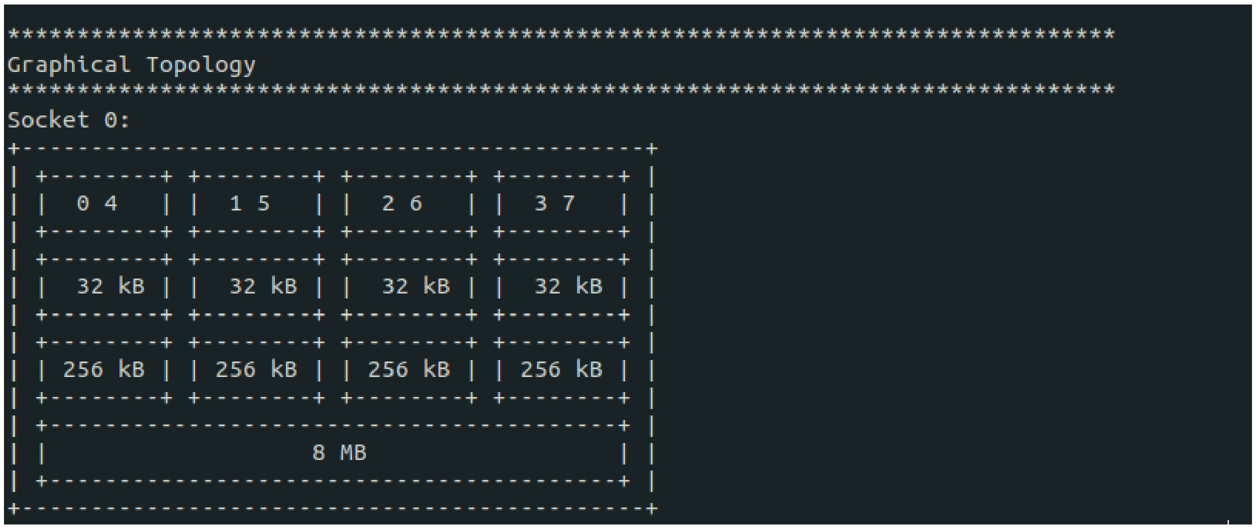
As we can see in the above outputs, the most frequent lines are in the loop that generates the output instances and plots which are in the filter function.

LIKWID

likwid-topology output

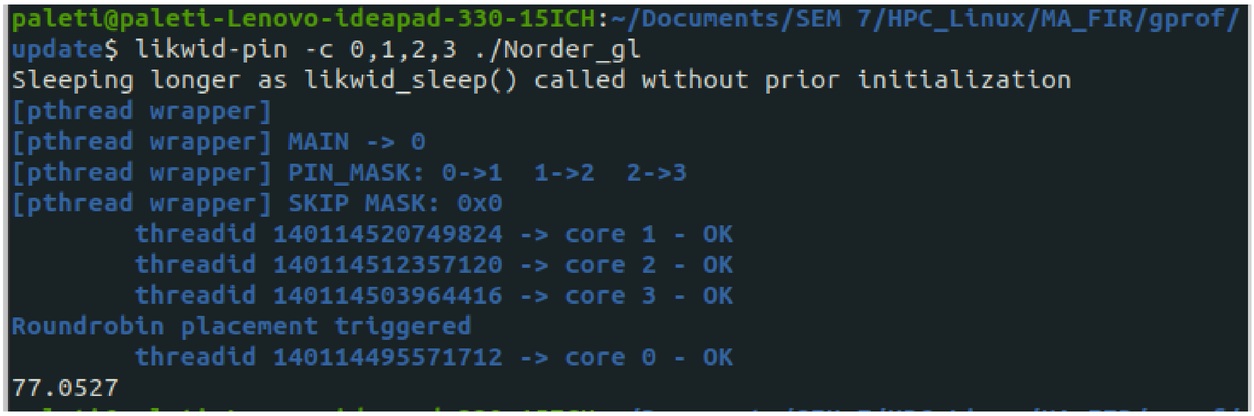


```
paleti@paleti-Lenovo-ideapad-330-15ICH:~/Documents/SEM 7/HPC_Linux/MA_FIR/gprof/update$ likwid-topology
--------------------------------------------------------------------------------
CPU name:       Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz
CPU type:       Intel Coffeelake processor
CPU stepping:   10
********************************************************************************
Hardware Thread Topology
********************************************************************************
Sockets:                1
Cores per socket:       4
Threads per core:       2
--------------------------------------------------------------------------------
HWThread        Thread          Core            Socket          Available
0               0               0               0               *
1               0               1               0               *
2               0               2               0               *
3               0               3               0               *
4               1               0               0               *
5               1               1               0               *
6               1               2               0               *
7               1               3               0               *
--------------------------------------------------------------------------------
Socket 0:               ( 0 4 1 5 2 6 3 7 )
--------------------------------------------------------------------------------
********************************************************************************
Cache Topology
********************************************************************************
Level:                  1
Size:                   32 kB
Cache groups:           ( 0 4 ) ( 1 5 ) ( 2 6 ) ( 3 7 )
--------------------------------------------------------------------------------
Level:                  2
Size:                   256 kB
Cache groups:           ( 0 4 ) ( 1 5 ) ( 2 6 ) ( 3 7 )
--------------------------------------------------------------------------------
Level:                  3
Size:                   8 MB
Cache groups:           ( 0 4 1 5 2 6 3 7 )
--------------------------------------------------------------------------------
********************************************************************************
NUMA Topology
********************************************************************************
NUMA domains:           1
--------------------------------------------------------------------------------
Domain:                 0
Processors:             ( 0 1 2 3 4 5 6 7 )
Distances:              10
Free memory:            190.207 MB
Total memory:           7831.84 MB
--------------------------------------------------------------------------------
```



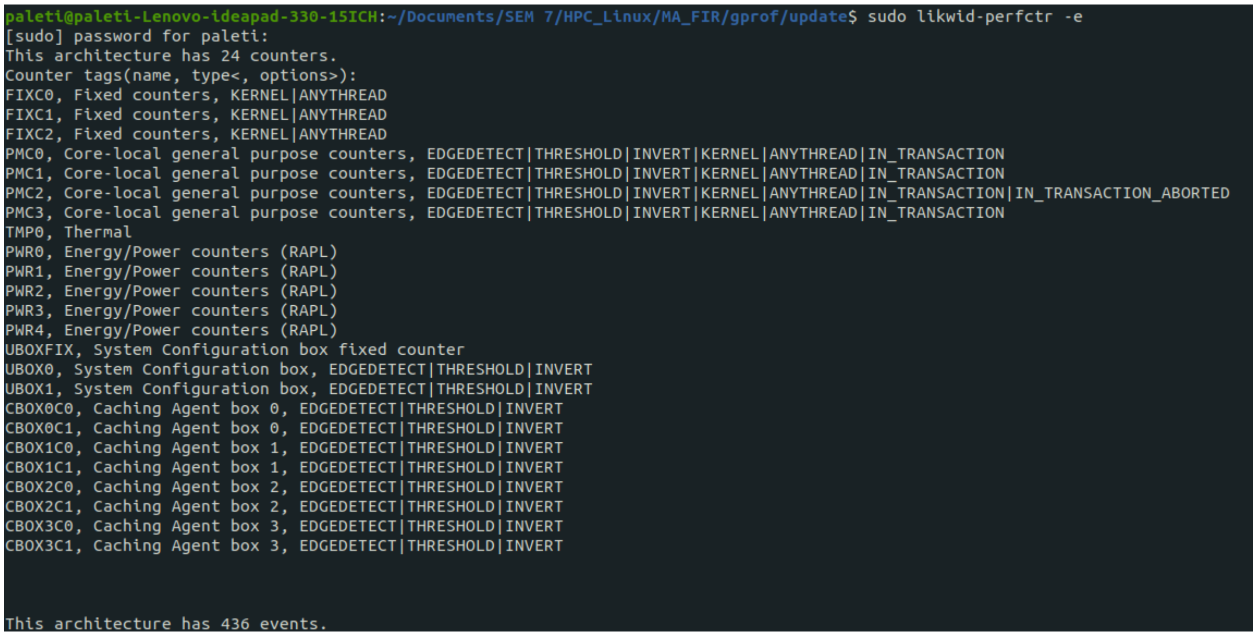
```
********************************************************************************
Graphical Topology
********************************************************************************
Socket 0:
+---------------------------------------------+
| +--------+ +--------+ +--------+ +--------+ |
| |  0 4   | |  1 5   | |  2 6   | |  3 7   | |
| +--------+ +--------+ +--------+ +--------+ |
| +--------+ +--------+ +--------+ +--------+ |
| |  32 kB | |  32 kB | |  32 kB | |  32 kB | |
| +--------+ +--------+ +--------+ +--------+ |
| +--------+ +--------+ +--------+ +--------+ |
| | 256 kB | | 256 kB | | 256 kB | | 256 kB | |
| +--------+ +--------+ +--------+ +--------+ |
| +-----------------------------------------+ |
| |                   8 MB                  | |
| +-----------------------------------------+ |
+---------------------------------------------+
```

likwid-pin -c output: As the program is not currently parallelized, the program just runs in the core 0.



```
paleti@paleti-Lenovo-ideapad-330-15ICH:~/Documents/SEM 7/HPC_Linux/MA_FIR/gprof/
update$ likwid-pin -c 0,1,2,3 ./Norder_gl
Sleeping longer as likwid_sleep() called without prior initialization
[pthread wrapper]
[pthread wrapper] MAIN -> 0
[pthread wrapper] PIN_MASK: 0->1  1->2  2->3
[pthread wrapper] SKIP MASK: 0x0
        threadid 140114520749824 -> core 1 - OK
        threadid 140114512357120 -> core 2 - OK
        threadid 140114503964416 -> core 3 - OK
Roundrobin placement triggered
        threadid 140114495571712 -> core 0 - OK
77.0527
```

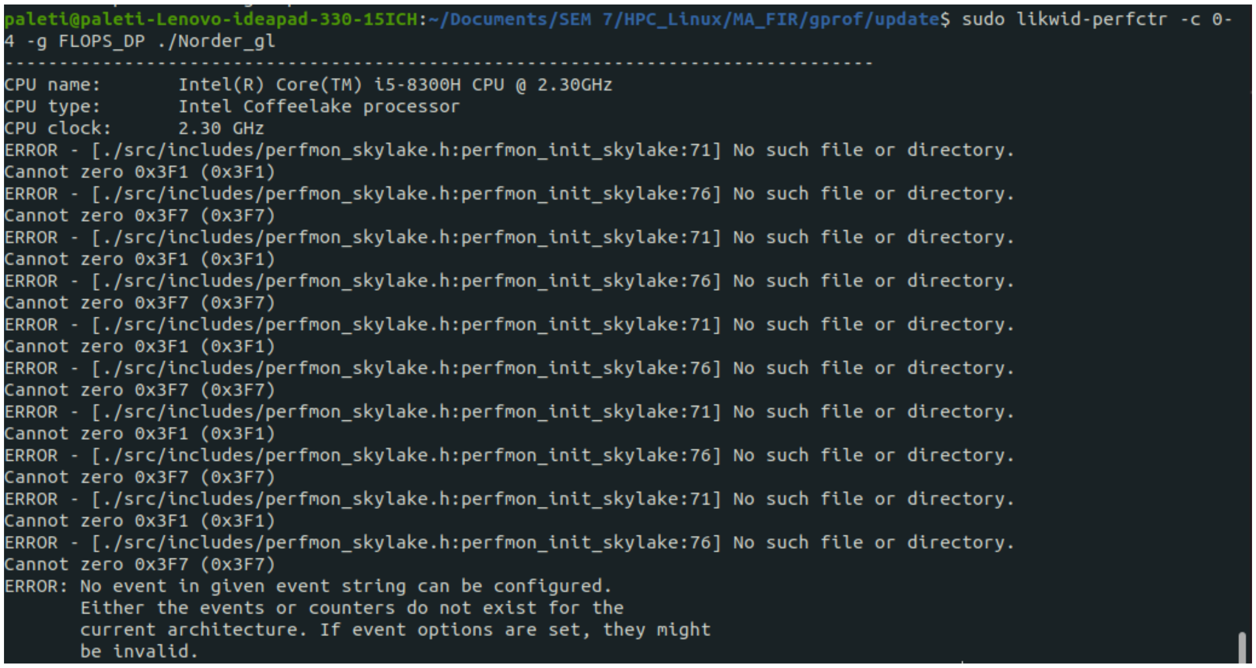
likwid-perfctr -e output:



```
paleti@paleti-Lenovo-ideapad-330-15ICH:~/Documents/SEM 7/HPC_Linux/MA_FIR/gprof/update$ sudo likwid-perfctr -e
[sudo] password for paleti:
This architecture has 24 counters.
Counter tags(name, type<, options>):
FIXC0, Fixed counters, KERNEL|ANYTHREAD
FIXC1, Fixed counters, KERNEL|ANYTHREAD
FIXC2, Fixed counters, KERNEL|ANYTHREAD
PMC0, Core-local general purpose counters, EDGEDETECT|THRESHOLD|INVERT|KERNEL|ANYTHREAD|IN_TRANSACTION
PMC1, Core-local general purpose counters, EDGEDETECT|THRESHOLD|INVERT|KERNEL|ANYTHREAD|IN_TRANSACTION
PMC2, Core-local general purpose counters, EDGEDETECT|THRESHOLD|INVERT|KERNEL|ANYTHREAD|IN_TRANSACTION|IN_TRANSACTION_ABORTED
PMC3, Core-local general purpose counters, EDGEDETECT|THRESHOLD|INVERT|KERNEL|ANYTHREAD|IN_TRANSACTION
TMP0, Thermal
PWR0, Energy/Power counters (RAPL)
PWR1, Energy/Power counters (RAPL)
PWR2, Energy/Power counters (RAPL)
PWR3, Energy/Power counters (RAPL)
PWR4, Energy/Power counters (RAPL)
UBOXFIX, System Configuration box fixed counter
UBOX0, System Configuration box, EDGEDETECT|THRESHOLD|INVERT
UBOX1, System Configuration box, EDGEDETECT|THRESHOLD|INVERT
CBOX0C0, Caching Agent box 0, EDGEDETECT|THRESHOLD|INVERT
CBOX0C1, Caching Agent box 0, EDGEDETECT|THRESHOLD|INVERT
CBOX1C0, Caching Agent box 1, EDGEDETECT|THRESHOLD|INVERT
CBOX1C1, Caching Agent box 1, EDGEDETECT|THRESHOLD|INVERT
CBOX2C0, Caching Agent box 2, EDGEDETECT|THRESHOLD|INVERT
CBOX2C1, Caching Agent box 2, EDGEDETECT|THRESHOLD|INVERT
CBOX3C0, Caching Agent box 3, EDGEDETECT|THRESHOLD|INVERT
CBOX3C1, Caching Agent box 3, EDGEDETECT|THRESHOLD|INVERT



This architecture has 436 events.
```

On checking the list of supported architectures using “likwid-perfctr -i”, the current processor “Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz” is not listed.

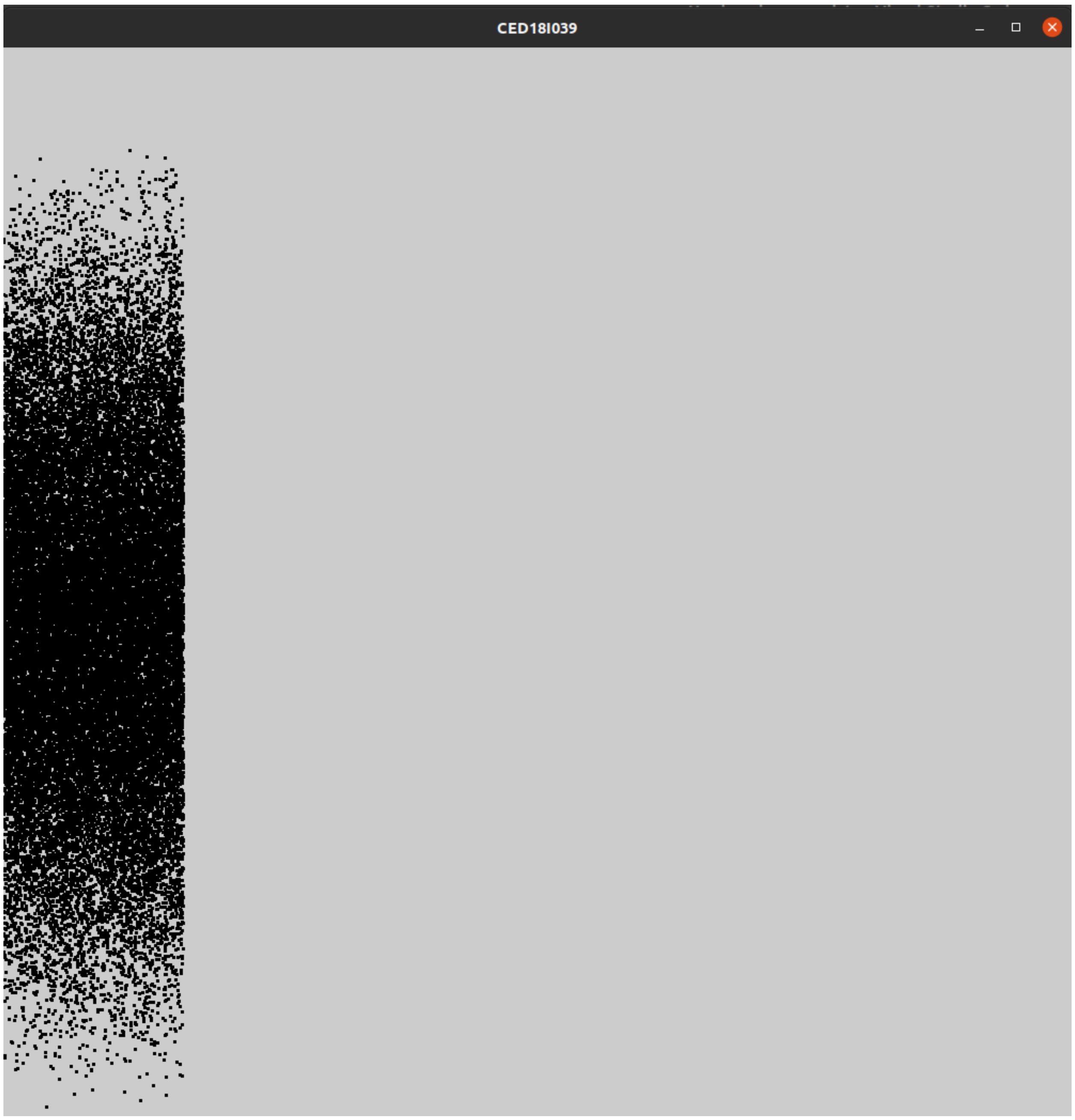
Output for the “ sudo likwid-perfctr -c 0-4 -g FLOPS_DP ./Norder_gl “



```
paleti@paleti-Lenovo-ideapad-330-15ICH:~/Documents/SEM 7/HPC_Linux/MA_FIR/gprof/update$ sudo likwid-perfctr -c 0-
4 -g FLOPS_DP ./Norder_gl
--------------------------------------------------------------------------------
CPU name:       Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz
CPU type:       Intel Coffeelake processor
CPU clock:      2.30 GHz
ERROR - [./src/includes/perfmon_skylake.h:perfmon_init_skylake:71] No such file or directory.
Cannot zero 0x3F1 (0x3F1)
ERROR - [./src/includes/perfmon_skylake.h:perfmon_init_skylake:76] No such file or directory.
Cannot zero 0x3F7 (0x3F7)
ERROR - [./src/includes/perfmon_skylake.h:perfmon_init_skylake:71] No such file or directory.
Cannot zero 0x3F1 (0x3F1)
ERROR - [./src/includes/perfmon_skylake.h:perfmon_init_skylake:76] No such file or directory.
Cannot zero 0x3F7 (0x3F7)
ERROR - [./src/includes/perfmon_skylake.h:perfmon_init_skylake:71] No such file or directory.
Cannot zero 0x3F1 (0x3F1)
ERROR - [./src/includes/perfmon_skylake.h:perfmon_init_skylake:76] No such file or directory.
Cannot zero 0x3F7 (0x3F7)
ERROR - [./src/includes/perfmon_skylake.h:perfmon_init_skylake:71] No such file or directory.
Cannot zero 0x3F1 (0x3F1)
ERROR - [./src/includes/perfmon_skylake.h:perfmon_init_skylake:76] No such file or directory.
Cannot zero 0x3F7 (0x3F7)
ERROR - [./src/includes/perfmon_skylake.h:perfmon_init_skylake:71] No such file or directory.
Cannot zero 0x3F1 (0x3F1)
ERROR - [./src/includes/perfmon_skylake.h:perfmon_init_skylake:76] No such file or directory.
Cannot zero 0x3F7 (0x3F7)
ERROR: No event in given event string can be configured.
       Either the events or counters do not exist for the
       current architecture. If event options are set, they might
       be invalid.
```

OUTPUT

Output screenshot of a frame.



OBSERVATIONS

On application of the mentioned profiling tools, namely, GPROF, GCOV, LIKWID, the hot spot of the program is determined.

GPROF: All functions are called only once.
GCOV: The most commonly run lines are the ones within the loop that generate and plot the instances of the filter output which depends on the three successive input instances, which is the potential parallelizable section of the code.
LIKWID: No inference was drawn from these outputs as the code is not yet parallelized and likwid-perfctr does not support the student's processor.