Pontificia Universidad Javeriana Cali

First Laboratory Report

Parallel Programming Course

Submitted by:
Katherine Camacho Calderón
Juan José Hoyos Urcué

Submitted to the teacher:
Roger Gomez Nieto

Santiago de Cali, March 11, 2020

INTRODUCTION

This workshop is divided into two parts:

In the first part, a series of analyzes and modifications were carried out on a sequential algorithm which purpose is to obtain the quality of a video by the measure of a score. Initially the modifications were made on the sequential algorithm, then embarrassing parallel programming is used on the improved algorithm, and finally comparisons and conclusions are made on said algorithm in order to identify the improvements in its performance

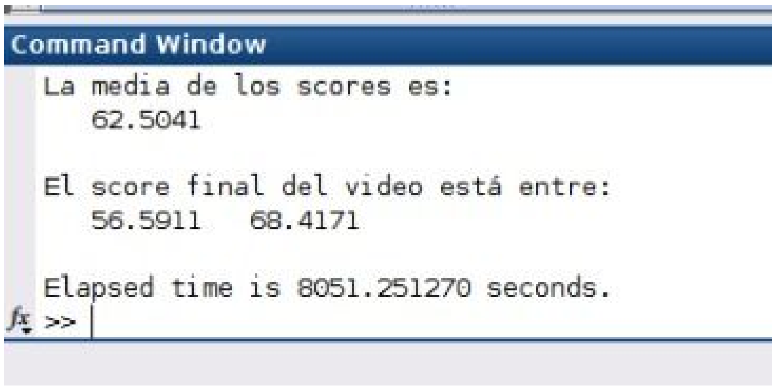
The second part will analyze the times of three algorithms that were executed sequentially and then concurrently, which allowed identifying the differences in performance between these two forms of execution.

First part

In the first part of this laboratory, a series of analyzes and modifications were carried out on the FRIQUEE algorithm, which is written in such a way that its execution is given sequentially, such analyzes and modifications were given as follows:

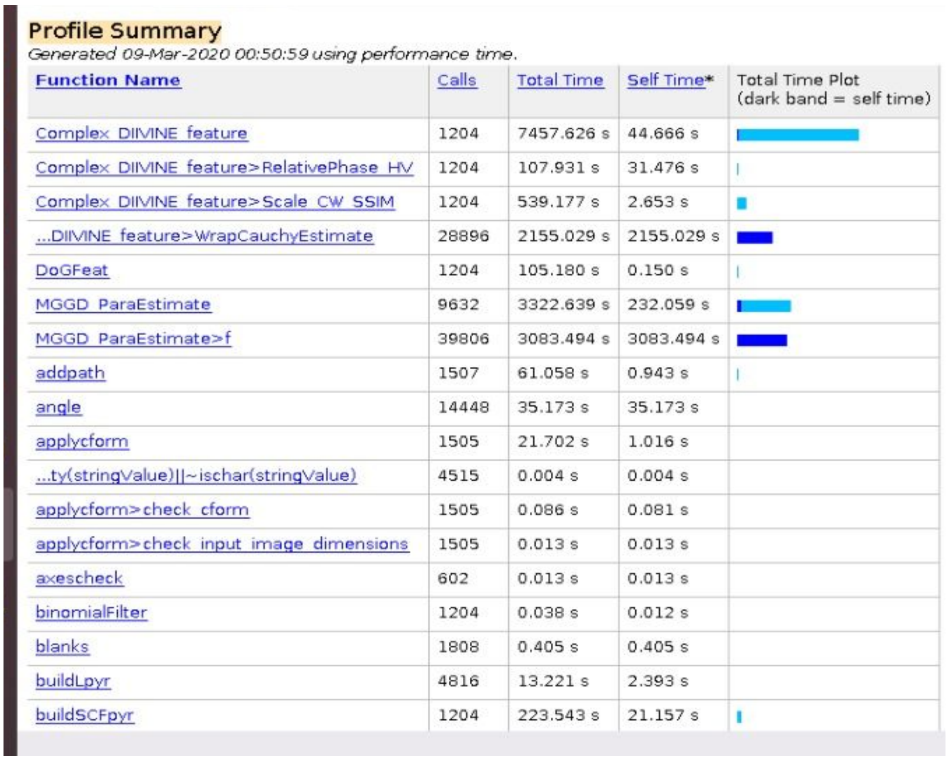
- First, the algorithm time was analyzed, without any modification, and what was obtained was:

  Mean of scores, video score(mean±std) and the total time were:



```
Command Window
  La media de los scores es:
     62.5041

  El score final del video está entre:
     56.5911   68.4171

  Elapsed time is 8051.251270 seconds.
fx >> |
```

  The time for each function was:



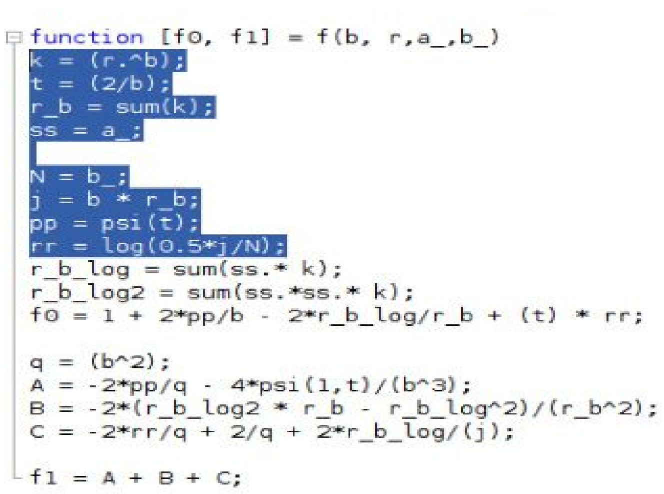
Profile Summary

Generated 09-Mar-2020 00:50:59 using performance time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
Complex_DIIVINE_feature	1204	7457.626 s	44.666 s	
Complex_DIIVINE_feature>RelativePhase_HV	1204	107.931 s	31.476 s	
Complex_DIIVINE_feature>Scale_CW_SSIM	1204	539.177 s	2.653 s	
...DIIVINE_feature>WrapCauchyEstimate	28896	2155.029 s	2155.029 s	
DoGFeat	1204	105.180 s	0.150 s	
MGGD_ParaEstimate	9632	3322.639 s	232.059 s	
MGGD_ParaEstimate>f	39806	3083.494 s	3083.494 s	
addpath	1507	61.058 s	0.943 s	
angle	14448	35.173 s	35.173 s	
applycform	1505	21.702 s	1.016 s	
...ty(stringValue)||~ischar(stringValue)	4515	0.004 s	0.004 s	
applycform>check_cform	1505	0.086 s	0.081 s	
applycform>check_input_image_dimensions	1505	0.013 s	0.013 s	
axescheck	602	0.013 s	0.013 s	
binomialFilter	1204	0.038 s	0.012 s	
blanks	1808	0.405 s	0.405 s	
buildLpyr	4816	13.221 s	2.393 s	
buildSCFpyr	1204	223.543 s	21.157 s	

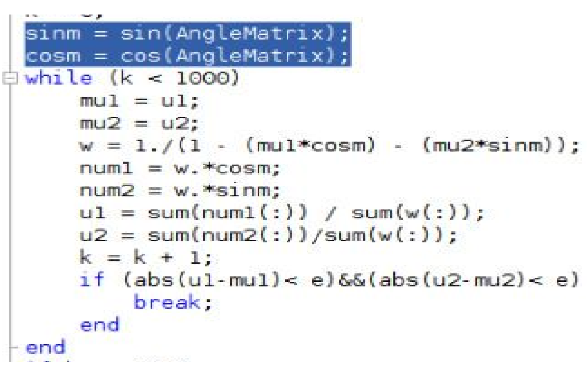
- Second, a series of modifications were made to the sequential algorithm, these modifications basically consisted of simplifying the number of mathematical operations, since there were repeated operations and taking out operations of some for loop, which did not depend on any value that was within it. Some of these modifications were:

    - In this image, the selected lines show operations that were previously repeated up to three times within this same function, so the simplification of the number of operations significantly reduced the execution time.



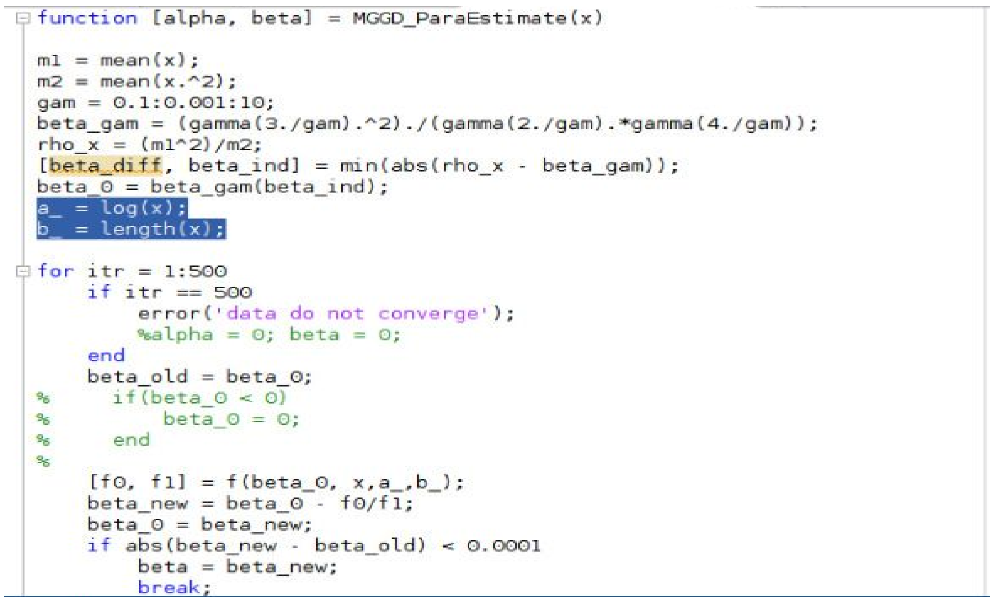
```
function [f0, f1] = f(b, r,a_,b_)
k = (r.^b);
t = (2/b);
r_b = sum(k);
ss = a_;

N = b_;
j = b * r_b;
pp = psi(t);
rr = log(0.5*j/N);
r_b_log = sum(ss.* k);
r_b_log2 = sum(ss.*ss.* k);
f0 = 1 + 2*pp/b - 2*r_b_log/r_b + (t) * rr;

q = (b^2);
A = -2*pp/q - 4*psi(1,t)/(b^3);
B = -2*(r_b_log2 * r_b - r_b_log^2)/(r_b^2);
C = -2*rr/q + 2/q + 2*r_b_log/(j);

f1 = A + B + C;
```

    - In this image, the selected lines show operations that were within the while loop and that were repeated unnecessarily many times, since their result was constant.Removing these operations from the while loop significantly improved the performance of the algorithm.



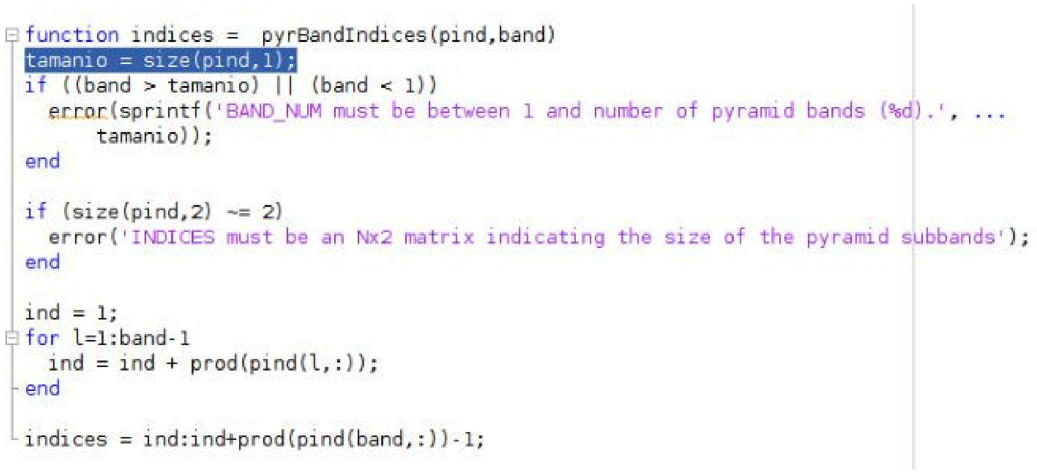
```
sinm = sin(AngleMatrix);
cosm = cos(AngleMatrix);
while (k < 1000)
    mu1 = u1;
    mu2 = u2;
    w = 1./(1 - (mu1*cosm) - (mu2*sinm));
    num1 = w.*cosm;
    num2 = w.*sinm;
    u1 = sum(num1(:)) / sum(w(:));
    u2 = sum(num2(:))/sum(w(:));
    k = k + 1;
    if (abs(u1-mu1)< e)&&(abs(u2-mu2)< e)
        break;
    end
end
```

- In this image the two lines selected were taken out from a for loop, since they were operations that did not depend on any value within it, their result within the for loop was always the same.



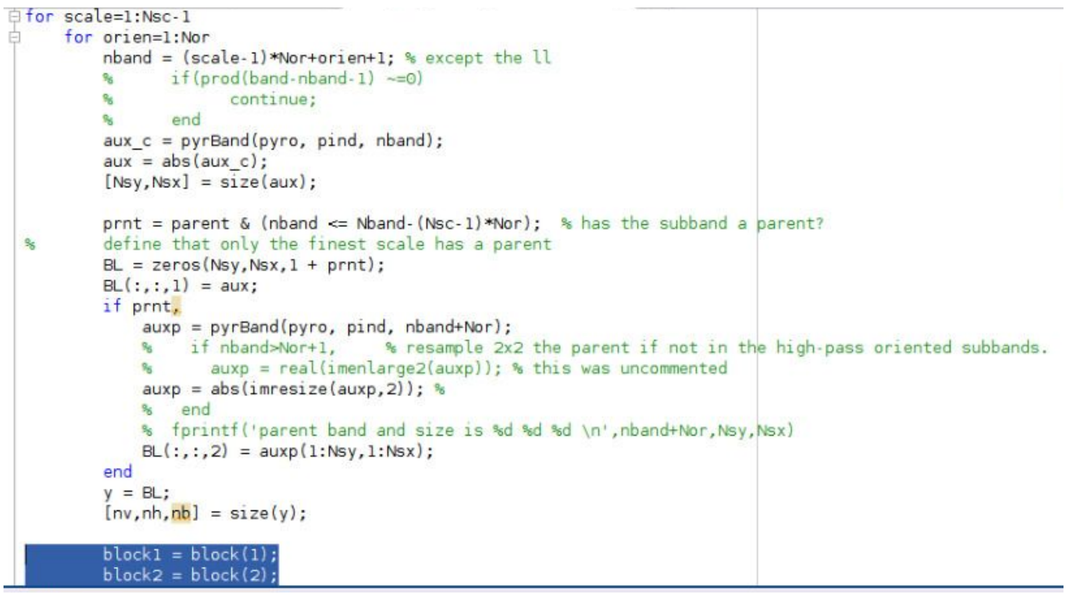
```
function [alpha, beta] = MGGD_ParaEstimate(x)

m1 = mean(x);
m2 = mean(x.^2);
gam = 0.1:0.001:10;
beta_gam = (gamma(3./gam).^2)./(gamma(2./gam).*gamma(4./gam));
rho_x = (m1^2)/m2;
[beta_diff, beta_ind] = min(abs(rho_x - beta_gam));
beta_0 = beta_gam(beta_ind);
a_ = log(x);
b_ = length(x);

for itr = 1:500
    if itr == 500
        error('data do not converge');
        %alpha = 0; beta = 0;
    end
    beta_old = beta_0;
%      if(beta_0 < 0)
%          beta_0 = 0;
%      end
%
    [f0, f1] = f(beta_0, x,a_,b_);
    beta_new = beta_0 - f0/f1;
    beta_0 = beta_new;
    if abs(beta_new - beta_old) < 0.0001
        beta = beta_new;
        break;
```

- In this image, the selected line shows an operation that was previously performed twice, which is expensive because it is within a function that can be called many times.



```
function indices =  pyrBandIndices(pind,band)
tamanio = size(pind,1);
if ((band > tamanio) || (band < 1))
  error(sprintf('BAND_NUM must be between 1 and number of pyramid bands (%d).', ...
      tamanio));
end

if (size(pind,2) ~= 2)
  error('INDICES must be an Nx2 matrix indicating the size of the pyramid subbands');
end

ind = 1;
for l=1:band-1
  ind = ind + prod(pind(l,:));
end

indices = ind:ind+prod(pind(band,:))-1;
```

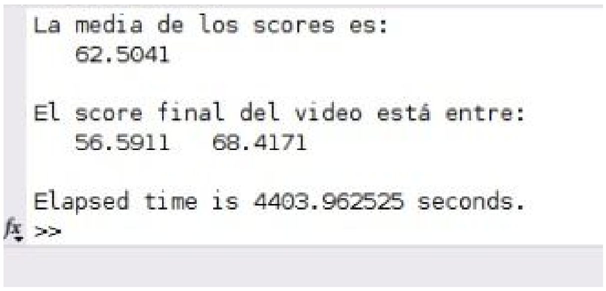
- In this image, the selected lines show two operations that were repeated twice within a double for loop, so saving them in a variable to avoid doing it twice, improves the performance of the algorithm.



```matlab
for scale=1:Nsc-1
    for orien=1:Nor
        nband = (scale-1)*Nor+orien+1; % except the ll
        %      if(prod(band-nband-1) ~=0)
        %            continue;
        %      end
        aux_c = pyrBand(pyro, pind, nband);
        aux = abs(aux_c);
        [Nsy,Nsx] = size(aux);

        prnt = parent & (nband <= Nband-(Nsc-1)*Nor);  % has the subband a parent?
%       define that only the finest scale has a parent
        BL = zeros(Nsy,Nsx,1 + prnt);
        BL(:,:,1) = aux;
        if prnt,
            auxp = pyrBand(pyro, pind, nband+Nor);
            %    if nband>Nor+1,     % resample 2x2 the parent if not in the high-pass oriented subbands.
            %       auxp = real(imenlarge2(auxp)); % this was uncommented
            auxp = abs(imresize(auxp,2)); %
            %   end
            %  fprintf('parent band and size is %d %d %d \n',nband+Nor,Nsy,Nsx)
            BL(:,:,2) = auxp(1:Nsy,1:Nsx);
        end
        y = BL;
        [nv,nh,nb] = size(y);

        block1 = block(1);
        block2 = block(2);
```

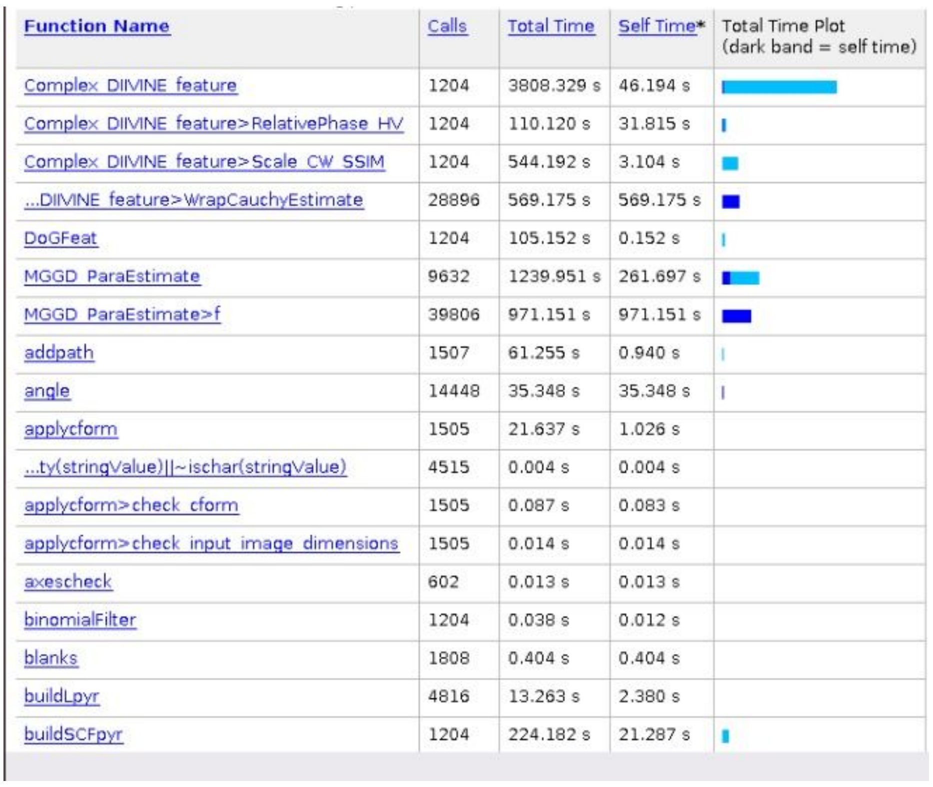
- Third, a report was made of the current time to perceive the improvements in the performance of the algorithm and this was obtained:

  The total time of the algorithm was:



```
La media de los scores es:
   62.5041

El score final del video está entre:
   56.5911   68.4171

Elapsed time is 4403.962525 seconds.
>>
```

  The time for each of the functions was:



Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
Complex DIIVINE feature	1204	3808.329 s	46.194 s	▬
Complex DIIVINE feature>RelativePhase HV	1204	110.120 s	31.815 s	▬
Complex DIIVINE feature>Scale CW SSIM	1204	544.192 s	3.104 s	▬
...DIIVINE feature>WrapCauchyEstimate	28896	569.175 s	569.175 s	▬
DoGFeat	1204	105.152 s	0.152 s	▬
MGGD ParaEstimate	9632	1239.951 s	261.697 s	▬
MGGD ParaEstimate>f	39806	971.151 s	971.151 s	▬
addpath	1507	61.255 s	0.940 s	▬
angle	14448	35.348 s	35.348 s	▬
applycform	1505	21.637 s	1.026 s	
...ty(stringValue)||~ischar(stringValue)	4515	0.004 s	0.004 s	
applycform>check cform	1505	0.087 s	0.083 s	
applycform>check input image dimensions	1505	0.014 s	0.014 s	
axescheck	602	0.013 s	0.013 s	
binomialFilter	1204	0.038 s	0.012 s	
blanks	1808	0.404 s	0.404 s	
buildLpyr	4816	13.263 s	2.380 s	
buildSCFpyr	1204	224.182 s	21.287 s	▬

Relevant optimizations:

Function: Complex DIVINE feature

7,457.626 ------------ 100%

3,808.329 --------------x

x = 51.06%. It's optimization was 48.94%

Function: MGGD ParaEstimate

3,322.639 ----------------100%

1,239.951 ----------------x

x = 37.32%. It's optimization was 62.68%

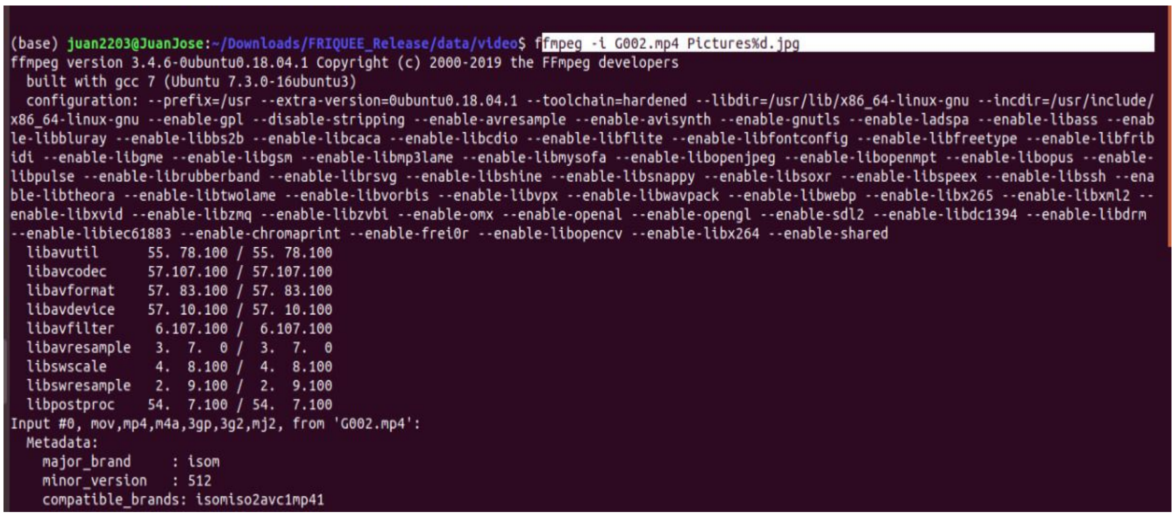
Function: MGGD ParaEstimate>f

3,083.494 -----------------100%

971.171 ------------------x

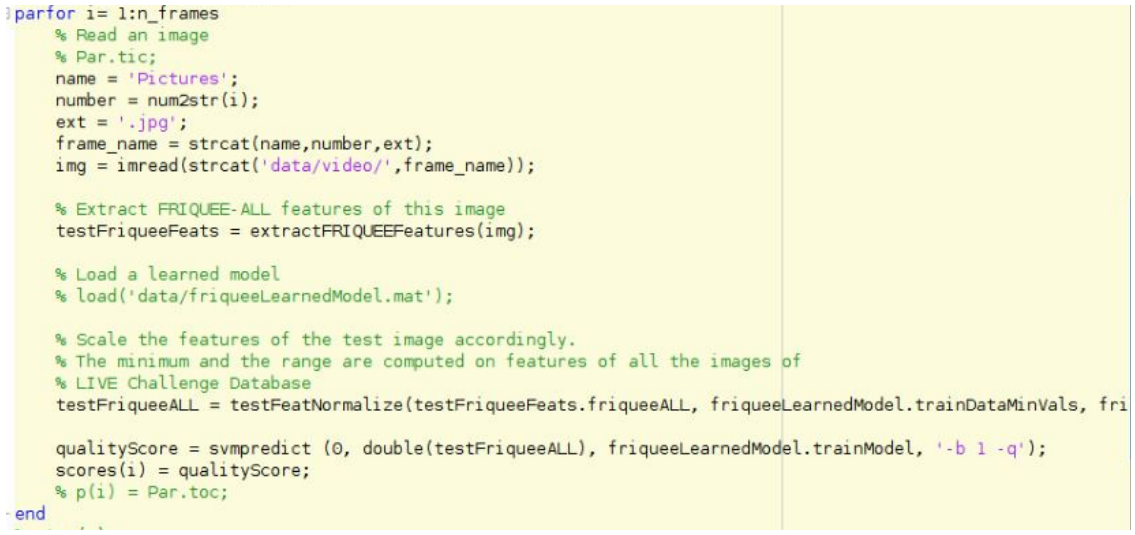
x = 31.5% .It's optimization was 68.5%

- Fourth, data decomposition was performed, so each core worked with a unique data set, a frame for each core, which significantly improved performance. The following line was used to perform data decomposition:



```
(base) juan2203@JuanJose:~/Downloads/FRIQUEE_Release/data/video$ ffmpeg -i G002.mp4 Pictures%d.jpg
ffmpeg version 3.4.6-0ubuntu0.18.04.1 Copyright (c) 2000-2019 the FFmpeg developers
  built with gcc 7 (Ubuntu 7.3.0-16ubuntu3)
  configuration: --prefix=/usr --extra-version=0ubuntu0.18.04.1 --toolchain=hardened --libdir=/usr/lib/x86_64-linux-gnu --incdir=/usr/include/
x86_64-linux-gnu --enable-gpl --disable-stripping --enable-avresample --enable-avisynth --enable-gnutls --enable-ladspa --enable-libass --enab
le-libbluray --enable-libbs2b --enable-libcaca --enable-libcdio --enable-libflite --enable-libfontconfig --enable-libfreetype --enable-libfrib
idi --enable-libgme --enable-libgsm --enable-libmp3lame --enable-libmysofa --enable-libopenjpeg --enable-libopenmpt --enable-libopus --enable-
libpulse --enable-librubberband --enable-librsvg --enable-libshine --enable-libsnappy --enable-libsoxr --enable-libspeex --enable-libssh --ena
ble-libtheora --enable-libtwolame --enable-libvorbis --enable-libvpx --enable-libwavpack --enable-libwebp --enable-libx265 --enable-libxml2 --
enable-libxvid --enable-libzmq --enable-libzvbi --enable-omx --enable-openal --enable-opengl --enable-sdl2 --enable-libdc1394 --enable-libdrm
--enable-libiec61883 --enable-chromaprint --enable-frei0r --enable-libopencv --enable-libx264 --enable-shared
  libavutil      55. 78.100 / 55. 78.100
  libavcodec     57.107.100 / 57.107.100
  libavformat    57. 83.100 / 57. 83.100
  libavdevice    57. 10.100 / 57. 10.100
  libavfilter     6.107.100 /  6.107.100
  libavresample   3.  7.  0 /  3.  7.  0
  libswscale      4.  8.100 /  4.  8.100
  libswresample   2.  9.100 /  2.  9.100
  libpostproc    54.  7.100 / 54.  7.100
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from 'G002.mp4':
  Metadata:
    major_brand     : isom
    minor_version   : 512
    compatible_brands: isomiso2avc1mp41
```

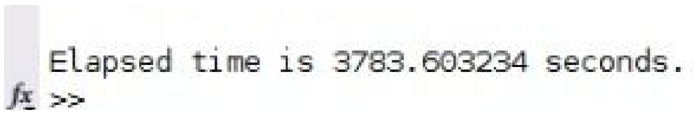
- Fifth, after having evaluated some for loops, we select the for loop that receive all the frames of the video, since at this point it is possible to divide the information in such a way that each core can return a value without relying on other core to finish performing a task . The selected for loop was the following:



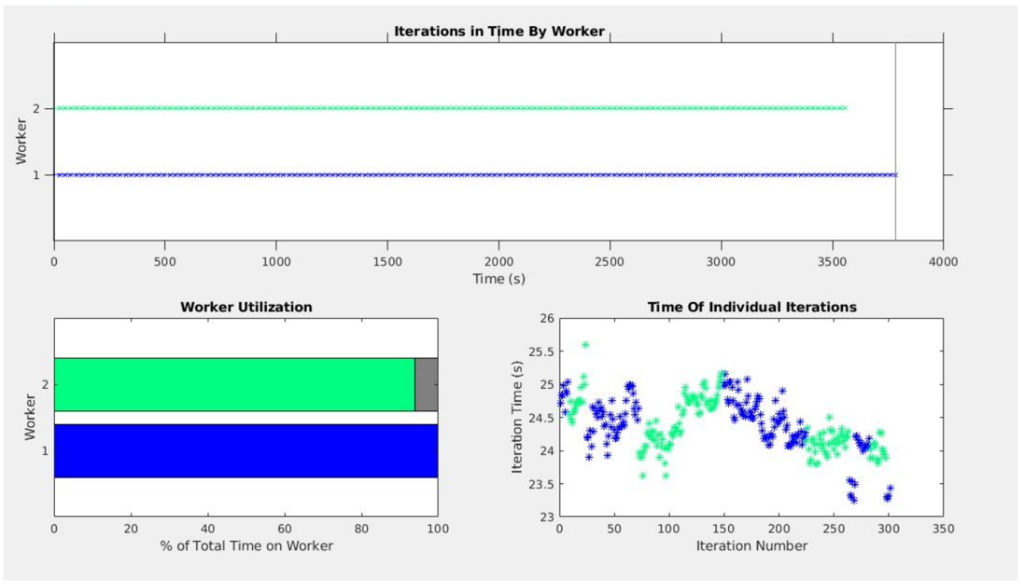
```matlab
parfor i= 1:n_frames
    % Read an image
    % Par.tic;
    name = 'Pictures';
    number = num2str(i);
    ext = '.jpg';
    frame_name = strcat(name,number,ext);
    img = imread(strcat('data/video/',frame_name));

    % Extract FRIQUEE-ALL features of this image
    testFriqueeFeats = extractFRIQUEEFeatures(img);

    % Load a learned model
    % load('data/friqueeLearnedModel.mat');

    % Scale the features of the test image accordingly.
    % The minimum and the range are computed on features of all the images of
    % LIVE Challenge Database
    testFriqueeALL = testFeatNormalize(testFriqueeFeats.friqueeALL, friqueeLearnedModel.trainDataMinVals, fri

    qualityScore = svmpredict (0, double(testFriqueeALL), friqueeLearnedModel.trainModel, '-b 1 -q');
    scores(i) = qualityScore;
    % p(i) = Par.toc;
end
```

- Sixth, the ParTicToc tool was implemented to evaluate the operation of the parfor and the following was obtained:

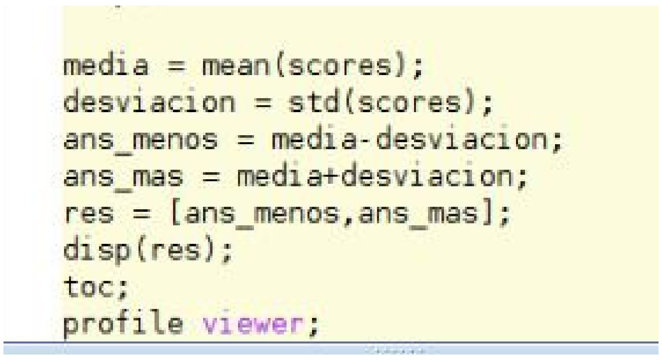
  The total time was:



```
Elapsed time is 3783.603234 seconds.
fx >>
```



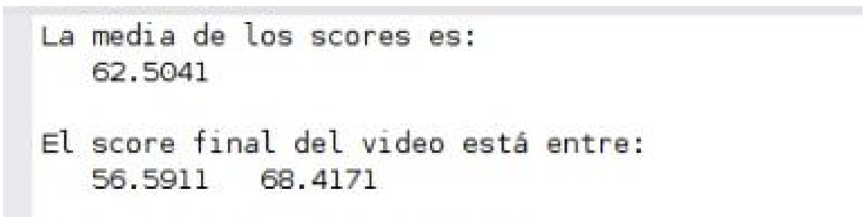
- Finally, the average of each frame was obtained and the range in which the video quality was found was calculated, the way in which this calculation was performed was:



```
media = mean(scores);
desviacion = std(scores);
ans_menos = media-desviacion;
ans_mas = media+desviacion;
res = [ans_menos,ans_mas];
disp(res);
toc;
profile viewer;
```

Where scores is an array that stores the score obtained in each one of the video frames.

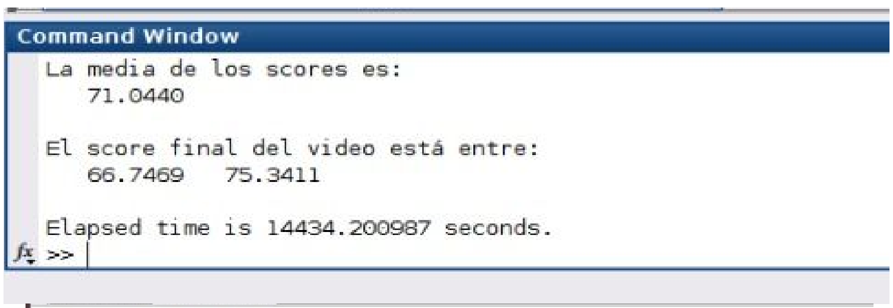
The range in which the score is found is:



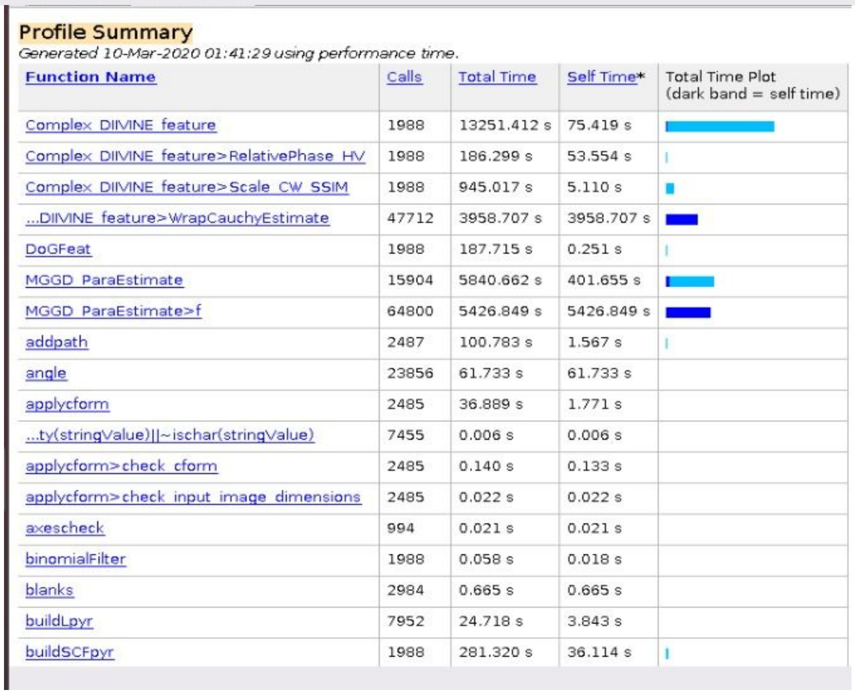
```
La media de los scores es:
   62.5041

El score final del video está entre:
   56.5911   68.4171
```

It was subsequently tested with another video and the following results were obtained:

- The algorithm time was analyzed, without any modification, and what was obtained was :



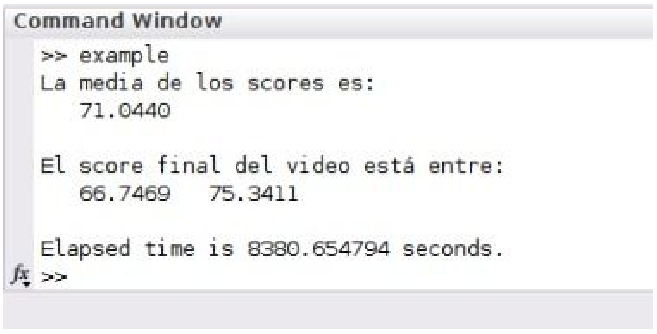
```
Command Window
La media de los scores es:
   71.0440

El score final del video está entre:
   66.7469   75.3411

Elapsed time is 14434.200987 seconds.
>> 
```



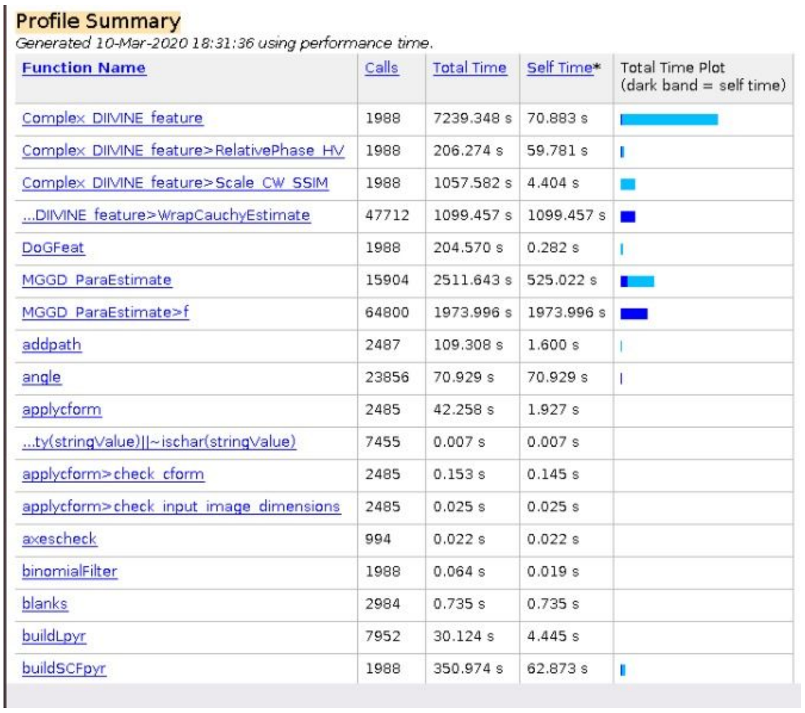
Profile Summary
Generated 10-Mar-2020 01:41:29 using performance time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
Complex_DIIVINE_feature	1988	13251.412 s	75.419 s	
Complex_DIIVINE_feature>RelativePhase_HV	1988	186.299 s	53.554 s	
Complex_DIIVINE_feature>Scale_CW_SSIM	1988	945.017 s	5.110 s	
...DIIVINE_feature>WrapCauchyEstimate	47712	3958.707 s	3958.707 s	
DoGFeat	1988	187.715 s	0.251 s	
MGGD_ParaEstimate	15904	5840.662 s	401.655 s	
MGGD_ParaEstimate>f	64800	5426.849 s	5426.849 s	
addpath	2487	100.783 s	1.567 s	
angle	23856	61.733 s	61.733 s	
applycform	2485	36.889 s	1.771 s	
...ty(stringValue)||~ischar(stringValue)	7455	0.006 s	0.006 s	
applycform>check_cform	2485	0.140 s	0.133 s	
applycform>check_input_image_dimensions	2485	0.022 s	0.022 s	
axescheck	994	0.021 s	0.021 s	
binomialFilter	1988	0.058 s	0.018 s	
blanks	2984	0.665 s	0.665 s	
buildLpyr	7952	24.718 s	3.843 s	
buildSCFpyr	1988	281.320 s	36.114 s	

- The algorithm time was analyzed after having made a series of modifications to the sequential algorithm, and the results were:



```
Command Window
>> example
La media de los scores es:
   71.0440

El score final del video está entre:
   66.7469   75.3411

Elapsed time is 8380.654794 seconds.
>>
```



Profile Summary

Generated 10-Mar-2020 18:31:36 using performance time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
Complex_DIIVINE_feature	1988	7239.348 s	70.883 s	
Complex_DIIVINE_feature>RelativePhase_HV	1988	206.274 s	59.781 s	
Complex_DIIVINE_feature>Scale_CW_SSIM	1988	1057.582 s	4.404 s	
...DIIVINE_feature>WrapCauchyEstimate	47712	1099.457 s	1099.457 s	
DoGFeat	1988	204.570 s	0.282 s	
MGGD_ParaEstimate	15904	2511.643 s	525.022 s	
MGGD_ParaEstimate>f	64800	1973.996 s	1973.996 s	
addpath	2487	109.308 s	1.600 s	
angle	23856	70.929 s	70.929 s	
applycform	2485	42.258 s	1.927 s	
...ty(stringValue)||~ischar(stringValue)	7455	0.007 s	0.007 s	
applycform>check_cform	2485	0.153 s	0.145 s	
applycform>check_input_image_dimensions	2485	0.025 s	0.025 s	
axescheck	994	0.022 s	0.022 s	
binomialFilter	1988	0.064 s	0.019 s	
blanks	2984	0.735 s	0.735 s	
buildLpyr	7952	30.124 s	4.445 s	
buildSCFpyr	1988	350.974 s	62.873 s	

Relevant optimizations:

Function: Complex DIVINE feature

13,251.412 ------------ 100%

7,239.348 --------------x

x = 54.6%. It's optimization was 45.4%

Function: MGGD ParaEstimate

5,840.662 ----------------100%

2,511.643 ----------------x

x = 43%. It’s optimization was 57%

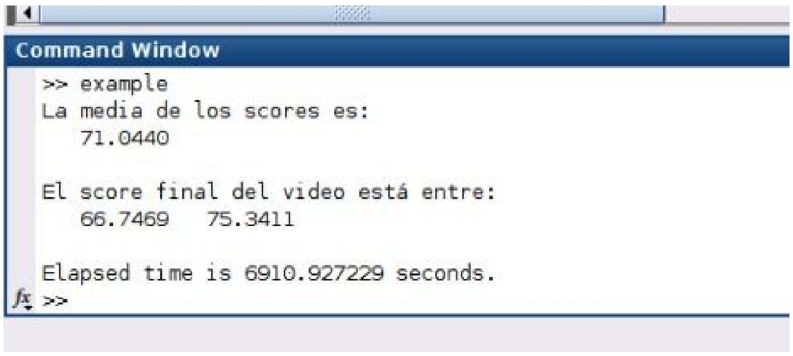
Function: MGGD ParaEstimate>f

5,426.849 -----------------100%

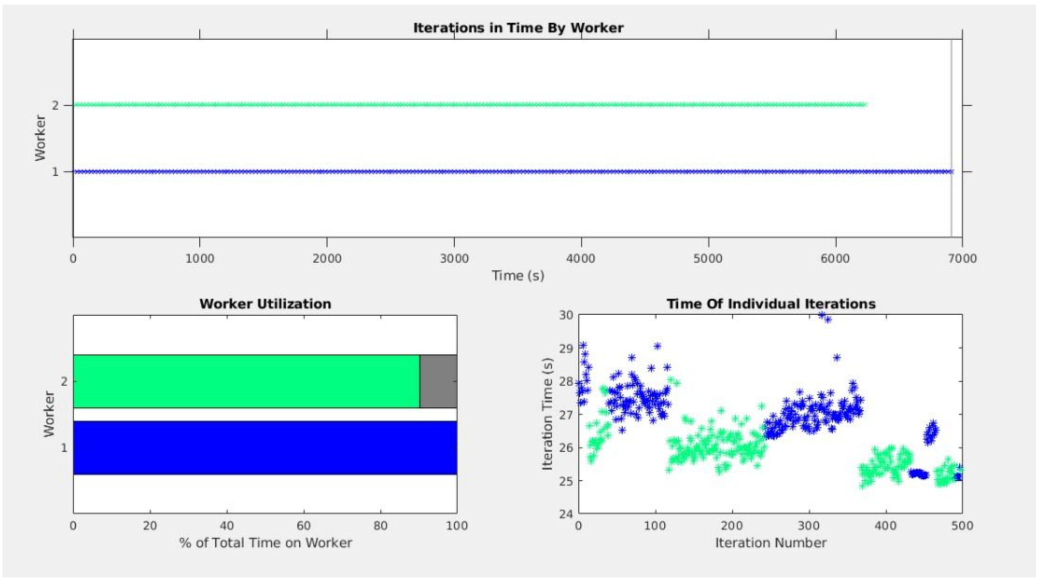
1,973.996 ------------------x

x = 36.37% .It’s optimization was 63.63%

- The algorithm time was analyzed after using the embarrassing parallel programming technique, and the results obtained were:



```
Command Window
>> example
La media de los scores es:
   71.0440

El score final del video está entre:
   66.7469   75.3411

Elapsed time is 6910.927229 seconds.
fx >>
```



Conclusion

Throughout the analysis of the algorithm, modifications and optimizations were made in different ways, the algorithm was first executed without any modification and an execution time of 2.23 hours was achieved, then sequential optimizations were made and an execution time of 1.22 hours was achieved with them. and finally the embarrassing parallel programming technique was implemented, achieving an execution time of 1.05 hours, where it was observed that each worker, took a different frame for a certain amount of time, and thanks to the graph we could detail that a worker finished first than another , but both worked together simultaneously most of the time, this allowed to significantly improve the execution time of the algorithm, so we conclude that using this technique correctly will allow the tasks to be divided and processed between the available cores and therefore maximum use of computer resources can be made.

By performing the above procedure with a new video, whose duration is 15 seconds and which was divided into 497 frames, it was obtained that sequential modifications improved performance much more than the algorithm that uses the embarrassing parallel programming technique, as the percentage of improvement with sequential modifications is 42% which is significantly greater than that achieved by the parallel algorithm 52%. Therefore, we conclude that it is important to analyze the algorithm sequentially first, and make all the optimizations in this way, and then if it is possible to use parallel programming to improve it further, the combination of both optimization techniques allowed to obtain a good result. compared to the one that initially was.

By executing the sequential code over the first video without any modification and the parallel code the **speedup** calculated was:

(8051.25127)/(3783.603234) = **2.12793223**

Second part

In the second part of this workshop, three high complexity algorithms were implemented, which were initially run sequentially and then concurrently.

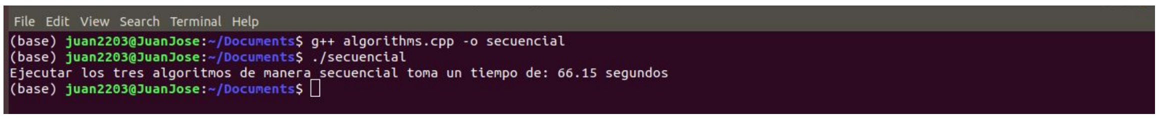
The first algorithm is responsible for finding all possible solutions of a linear equation of two variables until a number n (This equation is 3x + 5y = 100)

The second algorithm is responsible for finding all possible solutions of a linear equation of three variables until a number n (This equation is 3x + 5y +6z = 100)

The third algorithm is responsible for finding all possible solutions of a linear equation of four variables until a number n (This equation is 3x + 5y +6z+7w = 120)

These algorithms have time complexity of:
O (n ^ 2)
O (n ^ 3) and
O (n ^ 4)
respectively

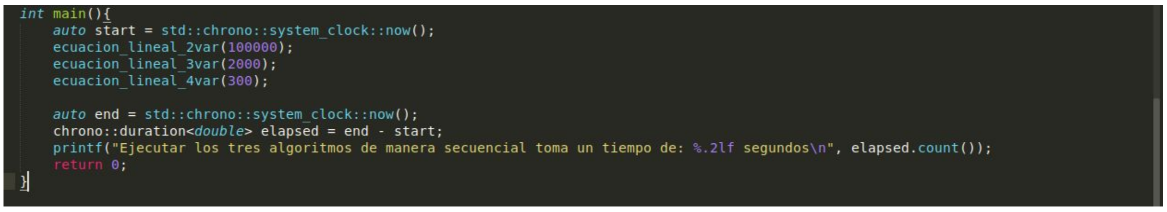
Sequentially



```
(base) juan2203@JuanJose:~/Documents$ g++ algorithms.cpp -o secuencial
(base) juan2203@JuanJose:~/Documents$ ./secuencial
Ejecutar los tres algoritmos de manera secuencial toma un tiempo de: 66.15 segundos
(base) juan2203@JuanJose:~/Documents$ 
```

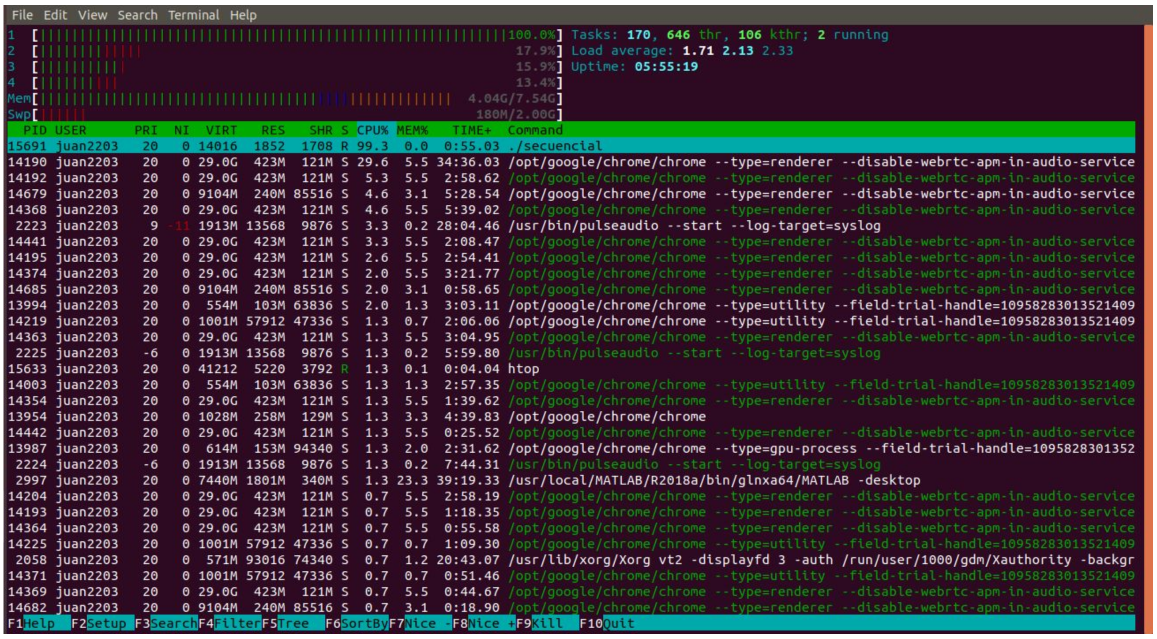
Running the algorithms sequentially takes a runtime of 66.15 seconds

They were executed in a main function one by one:



```
int main(){
    auto start = std::chrono::system_clock::now();
    ecuacion_lineal_2var(100000);
    ecuacion_lineal_3var(2000);
    ecuacion_lineal_4var(300);

    auto end = std::chrono::system_clock::now();
    chrono::duration<double> elapsed = end - start;
    printf("Ejecutar los tres algoritmos de manera secuencial toma un tiempo de: %.2lf segundos\n", elapsed.count());
    return 0;
}
```

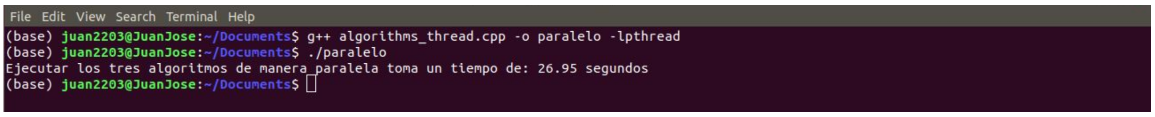
The processor looked like this when we ran the algorithms sequentially:



```
1  [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%] Tasks: 170, 646 thr, 106 kthr; 2 running
2  [|||||||||||||                                                17.9%] Load average: 1.71 2.13 2.33
3  [|||||||||||                                                  15.9%] Uptime: 05:55:19
4  [||||||||||                                                   13.4%]
Mem[|||||||||||||||||||||||||||||||||||||||||||||||||||||  4.04G/7.54G]
Swp[||||||                                                  180M/2.00G]
  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
15691 juan2203   20   0 14016  1852  1708 R 99.3  0.0  0:55.03 ./secuencial
14190 juan2203   20   0 29.0G  423M  121M S 29.6  5.5 34:36.03 /opt/google/chrome/chrome --type=renderer --disable-webrtc-apm-in-audio-service
14192 juan2203   20   0 29.0G  423M  121M S  5.3  5.5  2:58.62 /opt/google/chrome/chrome --type=renderer --disable-webrtc-apm-in-audio-service
14679 juan2203   20   0 9104M  240M 85516 S  4.6  3.1  5:28.54 /opt/google/chrome/chrome --type=renderer --disable-webrtc-apm-in-audio-service
14368 juan2203   20   0 29.0G  423M  121M S  4.6  5.5  5:39.02 /opt/google/chrome/chrome --type=renderer --disable-webrtc-apm-in-audio-service
 2223 juan2203    9 -11 1913M 13568  9876 S  3.3  0.2 28:04.46 /usr/bin/pulseaudio --start --log-target=syslog
14441 juan2203   20   0 29.0G  423M  121M S  3.3  5.5  2:08.47 /opt/google/chrome/chrome --type=renderer --disable-webrtc-apm-in-audio-service
14195 juan2203   20   0 29.0G  423M  121M S  2.6  5.5  2:54.41 /opt/google/chrome/chrome --type=renderer --disable-webrtc-apm-in-audio-service
14374 juan2203   20   0 29.0G  423M  121M S  2.0  5.5  3:21.77 /opt/google/chrome/chrome --type=renderer --disable-webrtc-apm-in-audio-service
14685 juan2203   20   0 9104M  240M 85516 S  2.0  3.1  0:58.65 /opt/google/chrome/chrome --type=renderer --disable-webrtc-apm-in-audio-service
13994 juan2203   20   0  554M  103M 63836 S  2.0  1.3  3:03.11 /opt/google/chrome/chrome --type=utility --field-trial-handle=10958283013521409
14219 juan2203   20   0 1001M 57912 47336 S  1.3  0.7  2:06.06 /opt/google/chrome/chrome --type=utility --field-trial-handle=10958283013521409
14363 juan2203   20   0 29.0G  423M  121M S  1.3  5.5  3:04.95 /opt/google/chrome/chrome --type=renderer --disable-webrtc-apm-in-audio-service
 2225 juan2203   -6   0 1913M 13568  9876 S  1.3  0.2  5:59.80 /usr/bin/pulseaudio --start --log-target=syslog
15633 juan2203   20   0 41212  5220  3792 R  1.3  0.1  0:04.04 htop
14003 juan2203   20   0  554M  103M 63836 S  1.3  1.3  2:57.35 /opt/google/chrome/chrome --type=utility --field-trial-handle=10958283013521409
14354 juan2203   20   0 29.0G  423M  121M S  1.3  5.5  1:39.62 /opt/google/chrome/chrome --type=renderer --disable-webrtc-apm-in-audio-service
13954 juan2203   20   0 1028M  258M  129M S  1.3  3.3  4:39.83 /opt/google/chrome/chrome
14442 juan2203   20   0 29.0G  423M  121M S  1.3  5.5  0:25.52 /opt/google/chrome/chrome --type=renderer --disable-webrtc-apm-in-audio-service
13987 juan2203   20   0  614M  153M 94340 S  1.3  2.0  2:31.62 /opt/google/chrome/chrome --type=gpu-process --field-trial-handle=1095828301352
 2224 juan2203   -6   0 1913M 13568  9876 S  1.3  0.2  7:44.31 /usr/bin/pulseaudio --start --log-target=syslog
 2997 juan2203   20   0 7440M 1801M  340M S  1.3 23.3 39:19.33 /usr/local/MATLAB/R2018a/bin/glnxa64/MATLAB -desktop
14204 juan2203   20   0 29.0G  423M  121M S  0.7  5.5  2:58.19 /opt/google/chrome/chrome --type=renderer --disable-webrtc-apm-in-audio-service
14193 juan2203   20   0 29.0G  423M  121M S  0.7  5.5  1:18.35 /opt/google/chrome/chrome --type=renderer --disable-webrtc-apm-in-audio-service
14364 juan2203   20   0 29.0G  423M  121M S  0.7  5.5  0:55.58 /opt/google/chrome/chrome --type=renderer --disable-webrtc-apm-in-audio-service
14225 juan2203   20   0 1001M 57912 47336 S  0.7  0.7  1:09.30 /opt/google/chrome/chrome --type=utility --field-trial-handle=10958283013521409
 2058 juan2203   20   0  571M 93016 74340 S  0.7  1.2 20:43.07 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/Xauthority -backgr
14371 juan2203   20   0 1001M 57912 47336 S  0.7  0.7  0:51.46 /opt/google/chrome/chrome --type=utility --field-trial-handle=10958283013521409
14369 juan2203   20   0 29.0G  423M  121M S  0.7  5.5  0:44.67 /opt/google/chrome/chrome --type=renderer --disable-webrtc-apm-in-audio-service
14682 juan2203   20   0 9104M  240M 85516 S  0.7  3.1  0:18.90 /opt/google/chrome/chrome --type=renderer --disable-webrtc-apm-in-audio-service
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

As we can see, core 1 is being 100% used , that is, all this resource is being used to execute the algorithms sequentially, and it is evident that the other core is being wasted.

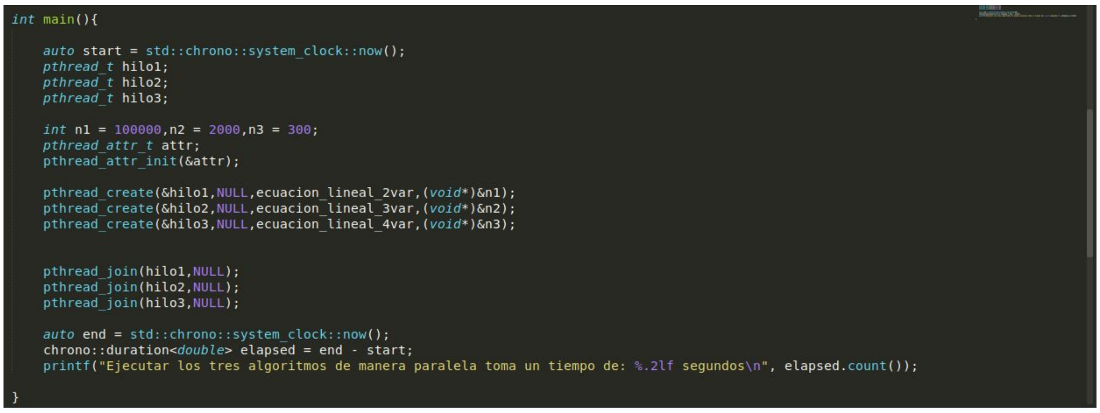
Concurrently Execution



```
(base) juan2203@JuanJose:~/Documents$ g++ algorithms_thread.cpp -o paralelo -lpthread
(base) juan2203@JuanJose:~/Documents$ ./paralelo
Ejecutar los tres algoritmos de manera paralela toma un tiempo de: 26.95 segundos
(base) juan2203@JuanJose:~/Documents$ 
```

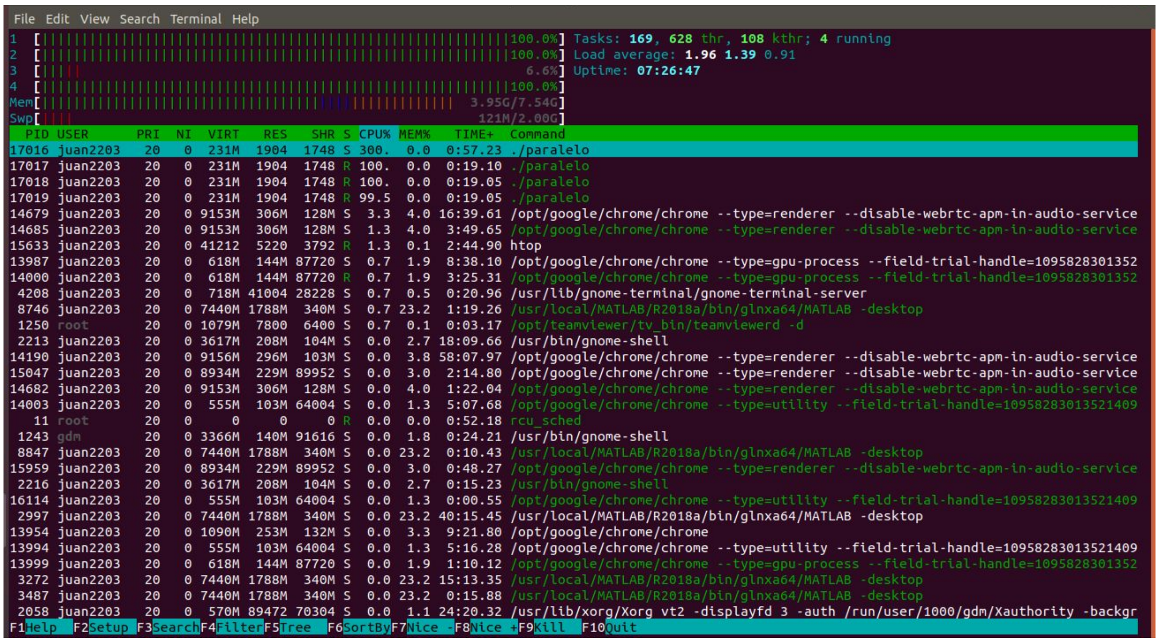
Running the algorithms concurrently takes a runtime of 26.95 seconds

They were all executed in a different thread each one:



```
int main(){

    auto start = std::chrono::system_clock::now();
    pthread_t hilo1;
    pthread_t hilo2;
    pthread_t hilo3;

    int n1 = 100000,n2 = 2000,n3 = 300;
    pthread_attr_t attr;
    pthread_attr_init(&attr);

    pthread_create(&hilo1,NULL,ecuacion_lineal_2var,(void*)&n1);
    pthread_create(&hilo2,NULL,ecuacion_lineal_3var,(void*)&n2);
    pthread_create(&hilo3,NULL,ecuacion_lineal_4var,(void*)&n3);


    pthread_join(hilo1,NULL);
    pthread_join(hilo2,NULL);
    pthread_join(hilo3,NULL);

    auto end = std::chrono::system_clock::now();
    chrono::duration<double> elapsed = end - start;
    printf("Ejecutar los tres algoritmos de manera paralela toma un tiempo de: %.2lf segundos\n", elapsed.count());

}
```

The processor looked like this when we ran the algorithms concurrently:



```
1  [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%] Tasks: 169, 628 thr, 108 kthr; 4 running
2  [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%] Load average: 1.96 1.39 0.91
3  [|||||                                                          6.6%] Uptime: 07:26:47
4  [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%]
Mem[||||||||||||||||||||||||||||||||||||||||||||||||||||||   3.95G/7.54G]
Swp[||||                                                     121M/2.00G]
  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
17016 juan2203   20   0  231M  1904  1748 S 300.  0.0  0:57.23 ./paralelo
17017 juan2203   20   0  231M  1904  1748 R 100.  0.0  0:19.10 ./paralelo
17018 juan2203   20   0  231M  1904  1748 R 100.  0.0  0:19.05 ./paralelo
17019 juan2203   20   0  231M  1904  1748 R 99.5  0.0  0:19.05 ./paralelo
14679 juan2203   20   0 9153M  306M  128M S  3.3  4.0 16:39.61 /opt/google/chrome/chrome --type=renderer --disable-webrtc-apm-in-audio-service
14685 juan2203   20   0 9153M  306M  128M S  1.3  4.0  3:49.65 /opt/google/chrome/chrome --type=renderer --disable-webrtc-apm-in-audio-service
15633 juan2203   20   0 41212  5220  3792 R  1.3  0.1  2:44.90 htop
13987 juan2203   20   0  618M  144M 87720 S  0.7  1.9  8:38.10 /opt/google/chrome/chrome --type=gpu-process --field-trial-handle=1095828301352
14000 juan2203   20   0  618M  144M 87720 R  0.7  1.9  3:25.31 /opt/google/chrome/chrome --type=gpu-process --field-trial-handle=1095828301352
 4208 juan2203   20   0  718M 41004 28228 S  0.7  0.5  0:20.96 /usr/lib/gnome-terminal/gnome-terminal-server
 8746 juan2203   20   0 7440M 1788M  340M S  0.7 23.2  1:19.26 /usr/local/MATLAB/R2018a/bin/glnxa64/MATLAB -desktop
 1250 root       20   0 1079M  7800  6400 S  0.7  0.1  0:03.17 /opt/teamviewer/tv_bin/teamviewerd -d
 2213 juan2203   20   0 3617M  208M  104M S  0.0  2.7 18:09.66 /usr/bin/gnome-shell
14190 juan2203   20   0 9156M  296M  103M S  0.0  3.8 58:07.97 /opt/google/chrome/chrome --type=renderer --disable-webrtc-apm-in-audio-service
15047 juan2203   20   0 8934M  229M 89952 S  0.0  3.0  2:14.80 /opt/google/chrome/chrome --type=renderer --disable-webrtc-apm-in-audio-service
14682 juan2203   20   0 9153M  306M  128M S  0.0  4.0  1:22.04 /opt/google/chrome/chrome --type=renderer --disable-webrtc-apm-in-audio-service
14003 juan2203   20   0  555M  103M 64004 S  0.0  1.3  5:07.68 /opt/google/chrome/chrome --type=utility --field-trial-handle=10958283013521409
   11 root       20   0     0     0     0 R  0.0  0.0  0:52.18 rcu_sched
 1243 gdm        20   0 3366M  140M 91616 S  0.0  1.8  0:24.21 /usr/bin/gnome-shell
 8847 juan2203   20   0 7440M 1788M  340M S  0.0 23.2  0:10.43 /usr/local/MATLAB/R2018a/bin/glnxa64/MATLAB -desktop
15959 juan2203   20   0 8934M  229M 89952 S  0.0  3.0  0:48.27 /opt/google/chrome/chrome --type=renderer --disable-webrtc-apm-in-audio-service
 2216 juan2203   20   0 3617M  208M  104M S  0.0  2.7  0:15.23 /usr/bin/gnome-shell
16114 juan2203   20   0  555M  103M 64004 S  0.0  1.3  0:00.55 /opt/google/chrome/chrome --type=utility --field-trial-handle=10958283013521409
 2997 juan2203   20   0 7440M 1788M  340M S  0.0 23.2 40:15.45 /usr/local/MATLAB/R2018a/bin/glnxa64/MATLAB -desktop
13954 juan2203   20   0 1090M  253M  132M S  0.0  3.3  9:21.80 /opt/google/chrome/chrome
13994 juan2203   20   0  555M  103M 64004 S  0.0  1.3  5:16.28 /opt/google/chrome/chrome --type=utility --field-trial-handle=10958283013521409
13999 juan2203   20   0  618M  144M 87720 S  0.0  1.9  1:10.12 /opt/google/chrome/chrome --type=gpu-process --field-trial-handle=1095828301352
 3272 juan2203   20   0 7440M 1788M  340M S  0.0 23.2 15:13.35 /usr/local/MATLAB/R2018a/bin/glnxa64/MATLAB -desktop
 3487 juan2203   20   0 7440M 1788M  340M S  0.0 23.2  0:15.88 /usr/local/MATLAB/R2018a/bin/glnxa64/MATLAB -desktop
 2058 juan2203   20   0  570M 89472 70304 S  0.0  1.1 24:20.32 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/Xauthority -backgr
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

As we can see, there are 3 cores that are being 100% used , this is how the great execution of the 3 algorithms is achieved concurrently, reducing sequential time in a drastic way

How fast does concurrent vs. sequential programming work in this case?

66.15 sec --------> 100%
26.95 sec --------> x

x = ((26.95)*100)/66.15 = 40.74%

This means that the concurrent algorithm only takes 40.74% of the time the sequential algorithm takes, therefore, we can say that the concurrent algorithm is 59.26% faster than the sequential algorithm.

Conclusion

We can then conclude that the use of concurrent programming in the execution of independent programs using threads is a very good solution for the problem of the large amount of time that these algorithms require when executing them sequentially, in addition, we can take the maximum advantage of computational resources that we have at home, making optimal use of them, thus increasing the performance of our programs.