Internship report

Internship by Régis JUBEAU at the University of Mälardalen in Sweden.
(for the 4^{th} year of Microelectronics and Automation of the Polytech Engineering School at the University of Sciences and Letters of Montpellier)

Thanks

I would like to thank in particular Mrs Cristina SECELEANU, my tutor at this university for her welcome.

I would also like to thank Mr Tiberiu SECELEANU for his support and information.

I would also like to thank Mr Jakob DANILESSON for his project topic, support, help and advice.

Table of contents

Introduction

Mälardalen Högskola is a Swedish university with a campus in Västerås. It has six different research specializations: educational sciences and mathematics, embedded systems, energy of the future, health and well-being, economics and management industries, and innovation and product production. The University of Mälardalen is organized into four schools including the School of Innovation, Design and Engineering where I do my internship.

My main objective of my internship was to create a graphical interface on the Qt framework coded in C++ that would be added to another graphical interface to complete the overall project. The global GUI should make it easier to run, display, and compare miss caches against the various Performance Application Programming Interface (PAPI) events that you will discover as this report progresses. You will also follow the path of these three months spent with researchers and PhD students. You will discover all the problems and resolutions taken to finish this GUI and assembly.

1 – Progress of my project

1.1 - Introduction of the internship project

To introduce my internship topic, here is a scenario. Multi-core systems are becoming the de facto norm in the fields of commercial computing and embedded computing. Multi-core processors have greater computing capacity while offering a decrease in size and weight compared to their single-core predecessors. Multi-core processors enable application- and system-level parallelization and simultaneous execution of different applications on different cores. Multi-core systems often implement an internal structure of shredding resources to increase the speed of communication between cores. Examples of shared resources include Translation lookaside Buffers (TLB), Last Level Cache (LLC), memory bus, and general purpose I/O device. Here is a typical shared resource structure in the figure below:

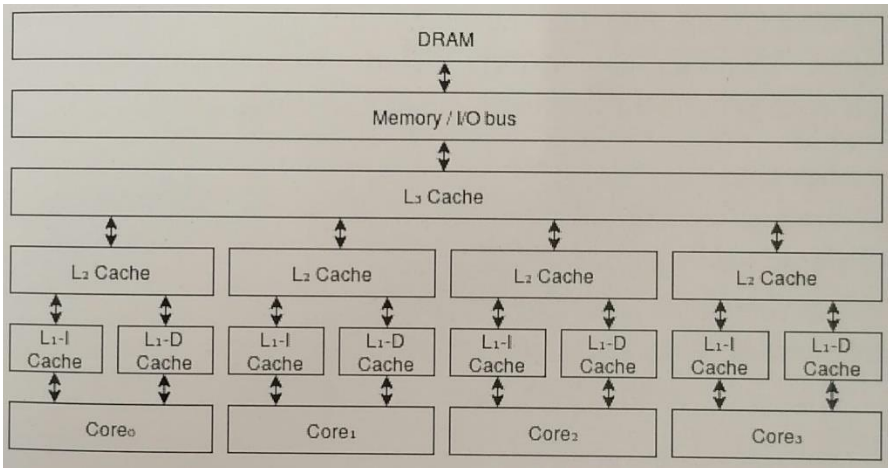

Figure 1: Example of a 4-core system

My internship topic was proposed by Dr. Danielsson in relation to his thesis, « [Automatic Characterization and Mitigation of Shared-Resource Contention in Multi-core Systems](#) », focused on homogeneous multi-core systems that use two or more cores to run applications. It targeted the Last-level cache (LLC), a term encompassing all caches last in the memory hierarchy. The LLC is most often physically shared between the different cores of a processor and connects the connection between a processor's local cache and the memory bus. It is particularly prone to the challenge of shared resources since it is shared between different hearts. During his thesis, his research focused mainly on challenging shared resources, which occurs due to the simultaneous use of multiple cores.

During his thesis, he made a code where he used PAPI (Performance Application Programming Interface) which is a portable programming interface allowing access to hardware counters specific to modern microprocessors. PAPI is used to collect low-level information, such as the number of floating-point operations per second, the number of cache misses during code execution, and so on. This methodology has a list of events and here is an example that I was able to meet during my internship:

```
================================================================================
 PAPI Preset Events
================================================================================
    Name        Code    Avail Deriv Description (Note)
PAPI_L1_DCM  0x80000000  Yes   No   Level 1 data cache misses
PAPI_L1_ICM  0x80000001  Yes   No   Level 1 instruction cache misses
PAPI_L2_DCM  0x80000002  Yes   Yes  Level 2 data cache misses
PAPI_L2_ICM  0x80000003  Yes   No   Level 2 instruction cache misses
PAPI_L3_DCM  0x80000004  No    No   Level 3 data cache misses
PAPI_L3_ICM  0x80000005  No    No   Level 3 instruction cache misses
PAPI_L1_TCM  0x80000006  Yes   Yes  Level 1 cache misses
PAPI_L2_TCM  0x80000007  Yes   No   Level 2 cache misses
PAPI_L3_TCM  0x80000008  Yes   No   Level 3 cache misses
PAPI_CA_SNP  0x80000009  No    No   Requests for a snoop
PAPI_CA_SHR  0x8000000a  No    No   Requests for exclusive access to shared cache line
PAPI_CA_CLN  0x8000000b  No    No   Requests for exclusive access to clean cache line
PAPI_CA_INV  0x8000000c  No    No   Requests for cache line invalidation
PAPI_CA_ITV  0x8000000d  No    No   Requests for cache line intervention
PAPI_L3_LDM  0x8000000e  No    No   Level 3 load misses
PAPI_L3_STM  0x8000000f  No    No   Level 3 store misses
PAPI_BRU_IDL 0x80000010  No    No   Cycles branch units are idle
PAPI_FXU_IDL 0x80000011  No    No   Cycles integer units are idle
PAPI_FPU_IDL 0x80000012  No    No   Cycles floating point units are idle
PAPI_LSU_IDL 0x80000013  No    No   Cycles load/store units are idle
PAPI_TLB_DM  0x80000014  Yes   Yes  Data translation lookaside buffer misses
PAPI_TLB_IM  0x80000015  Yes   No   Instruction translation lookaside buffer misses
PAPI_TLB_TL  0x80000016  No    No   Total translation lookaside buffer misses
PAPI_L1_LDM  0x80000017  Yes   No   Level 1 load misses
PAPI_L1_STM  0x80000018  Yes   No   Level 1 store misses
PAPI_L2_LDM  0x80000019  No    No   Level 2 load misses
PAPI_L2_STM  0x8000001a  Yes   No   Level 2 store misses
PAPI_BTAC_M  0x8000001b  No    No   Branch target address cache misses
PAPI_PRF_DM  0x8000001c  No    No   Data prefetch cache misses
PAPI_L3_DCH  0x8000001d  No    No   Level 3 data cache hits
PAPI_TLB_SD  0x8000001e  No    No   Translation lookaside buffer shootdowns
PAPI_CSR_FAL 0x8000001f  No    No   Failed store conditional instructions
```

Figure 2: Example of a PAPI event

When executing this code, depending on the counters chosen, a .txt file was returned, consisting of 3 columns: an index, the number of operations per second and the number of miss caches in this form:

```
0    6819753.000000 0.000000
1    8741416.000000 169730.000000
2    11457841.000000    208470.000000
3    12503279.000000    32557.000000
4    12459414.000000    477.000000
5    12507119.000000    673.000000
6    12484805.000000    649.000000
7    12581108.000000    989.000000
8    12555750.000000    884.000000
9    12585567.000000    1322.000000
10   12561676.000000    1363.000000
11   12508574.000000    1822.000000
12   12467212.000000    1833.000000
13   12566951.000000    3023.000000
14   12294836.000000    3011.000000
15   12327418.000000    5907.000000
16   12465443.000000    5061.000000
17   12323771.000000    5031.000000
18   12534302.000000    10286.000000
19   12486959.000000    9603.000000
20   12499975.000000    15786.000000
21   12450040.000000    16834.000000
22   12481974.000000    22996.000000
23   12387516.000000    23777.000000
24   12444505.000000    25748.000000
25   12437045.000000    33891.000000
26   12394063.000000    35635.000000
27   12426911.000000    33164.000000
28   12393687.000000    39463.000000
29   12427847.000000    41665.000000
30   12387581.000000    40985.000000
31   12393431.000000    51318.000000
32   12347306.000000    55473.000000
33   12296835.000000    48061.000000
34   12263438.000000    57342.000000
35   12411682.000000    55717.000000
36   12400546.000000    53899.000000
37   12373889.000000    59604.000000
38   12404837.000000    63893.000000
39   12384544.000000    64095.000000
40   12380998.000000    64485.000000
41   12164959.000000    66216.000000
42   12294313.000000    75860.000000
43   12105512.000000    69732.000000
44   12319725.000000    68812.000000
45   12346738.000000    69810.000000
46   12354788.000000    73674.000000
47   12242969.000000    75121.000000
48   12372814.000000    75408.000000
49   12377678.000000    73204.000000
50   12156448.000000    83059.000000
```

Figure 3: Sample .txt file

So, the objective of my internship was to realize a graphical interface on the Qt framework coded in C++. This GUI had to read files .txt, and analyze them. The goal is to have files of the same event but made with different applications. And assign to each of the applications, a number of partitions according to the number of missed caches.

1.2 - Realization of the graphical interface

An image of the GUI will be put at the end of this part (Figure 18).
For the examples, it will always be the same event "PAPI_L2_DCA" with the same applications ("FAST", "SIFT", "SUSAN") to see the evolution as the explanation progresses.

1.2.1 - Reading .txt files selected by the user

For the project, it was necessary that it was the user who had the choice of the files .txt he wanted to analyze. For this, I created a button that takes us into the files of the computer. Then the user only has to select what he wants.

```
void MainWindow::on_btn_select_file_clicked()
{
    if (numberCore < QThread::idealThreadCount())
    {
        file_name = QFileDialog::getOpenFileName(this, "Open files",
"/home/mdh/Desktop/Summerwork2022/new_small_tests");
        file_path.append(file_name);
        numberFile++;
        numberCore++;
        ui->lineEdit->setText(QString::number(numberFile));
    }
    else
    {
        QMessageBox::warning(this, "Alert",
"You can't add more plots because your core count is " + QString::number(QThread::
idealThreadCount(), 10));
    }
}
```

Figure 4: Function of the « OPEN » button

We can see in this image, the code for the button. Thanks to Qt and its functions, as soon as the user clicks on the button, he will enter this function. This function retrieves the name of the selected file that it will save by adding it in the vector "file_path". This function will also check the number of open files against the core numbers that the computer has, hence the open file flag. If there is a higher number, an alert will open.

For the reading part of the file, it is the "create_vector" function (Appendix 1) that takes care of it. Namely, the .txt files are written as columns and they are all separated by the "tab" character. The function retrieves the vector containing the name of each of the selected files and opens them one by one. Once opened, the function reads each (double) value and adds it to a vector that corresponds to it. Then its vectors are themselves added in another vector which allows me to have in a single vector all the values of the number of instructions or the number of miss caches. Here is the result of this function in image:

```
Nombre d'instruction par seconde.
 QVector(QVector(2.01887e+06, 1.86867e+06, 3.98747e+06, 2.45868e+06, 5.92974e+06, 1.16236e+07, 1.17027e+07, 1.34605e+07, 1.41926e+07, 1.54225e+07, 1.61381e+07, 1.54542e+07, 1.71611e+07, 1.6959e+07, 1.5652e+07, 1.6254e+07,
1.75231e+07, 1.71428e+07, 1.76871e+07, 1.22193e+07, 3.73553e+06, 2.65164e+06, 2.45868e+06, 2.45868e+06, 2.45868e+06, 2.45868e+06, 2.46276e+06), QVector(2.01896e+06, 1.87281e+06, 4.44802e+06, 1.14854e+07, 1.32754e+07,
1.3297e+07, 1.32931e+07, 1.33357e+07, 1.3425e+07, 1.33177e+07, 1.33432e+07, 1.33132e+07, 1.32937e+07, 1.33651e+07, 1.22739e+07, 5.51471e+06, 5.77134e+06, 1.30669e+07, 1.60208e+07, 1.51991e+07, 1.58257e+07, 1.57526e+07))

 Nombre de caches misses:
 QVector(QVector(0, 99719, 90453, 154563, 114636, 184970, 298776, 300192, 184325, 131181, 132156, 129803, 129303, 132396, 131901, 129585, 129498, 132850, 132169, 133181, 207299, 321955, 145899, 114636, 114636, 114636, 114636),
QVector(0, 98897, 89547, 225591, 184895, 124981, 123926, 124017, 121737, 118974, 122964, 121675, 122223, 122842, 120696, 160572, 338639, 314658, 431819, 355434, 373625, 341058))
```

Figure 5: Result of the « create_vector » function

By opening only three files, we see that a vector includes three.

1.2.2 - Scan miss caches based on files

For this part, I had to share a number of partitions (16 partitions) according to the number of missed caches of each application made on an event. For this, I proceeded in stages.

First, I had to perform a function that found the segmentations of my vectors. Here is an approximate and representative diagram of my function:

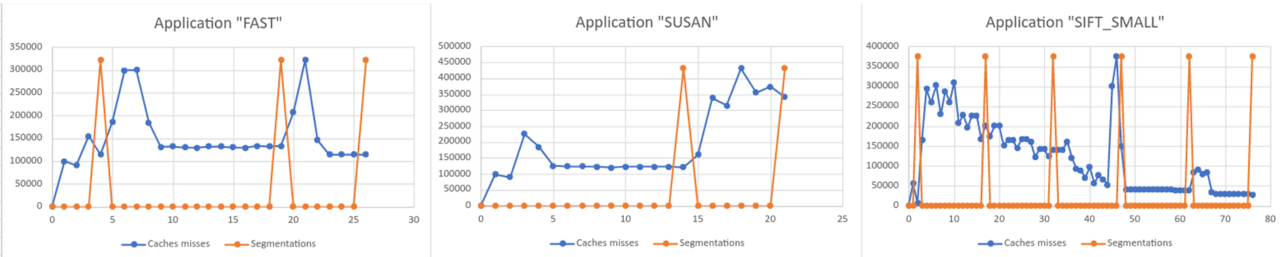

Figure 6: File segmentation scheme

To create it, I created another function that returned to me the overall average of the vector that I put as input, and I then analyzed, each value of the segment in relation to it. If one of the values was less than 80% of the average, then the function creates segmentation. This function is referred to as "find_segmentation" (Appendix 2).

```
void MainWindow::find_segmentation(QVector<QVector<double>> *tb, QVector<QVector<double>> *
segm)
{
    QVector<double> segmentation;
    QVector<double> colonne;
    for (int j = 0; j < tb->size(); j++)
    {
        colonne = tb->at(j);

        double avg = AVG(&colonne);
        int i;
        avg *= 0.8;

        for (i = 0; i < colonne.size(); i++)
        {
            if (colonne.at(i) < avg)
            {
                segmentation.append(i);
            }

        }
        colonne.clear();
        segmentation.append(i);
        segm->append(segmentation);
        segmentation.clear();
    }

}
```

Figure 7: Function « find_segmentation »

```
QVector(QVector(0, 1, 2, 4, 23, 24, 25, 26, 27), QVector(0, 1, 2, 37, 38, 39, 41, 42, 43, 44, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77), QVector(0,
1, 2, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 22))
```

Figure 8: Result of the « find_segmentation » function

From the results, we can see that there are consecutive segmentations too close that we will remove later.

For each segmentation vector, I checked that the segmentations were not too close or too far away to avoid having too large or small number of segmentations. For this, I checked if the difference of each consecutive value was greater than 10 or greater than 20. In these cases, either I removed the lower value to form a single segmentation, or I added a segmentation to about +15 compared to the smallest value.

```
QVector(QVector(4, 19, 27), QVector(2, 17, 32, 47, 62, 77), QVector(14, 22))
```

Figure 9: Result of the « correct_segm » function

According to the result, we went for the first vector from 9 segmentations to 3, for the second from 15 segmentations to 2 and for the third from 40 to 6. In this result, we can see for the first vector, that there is the segmentation "19" which did not appear as a basis. It was added because the difference between 27 and 4 was greater than 20. This allows the segmentation vectors to be more homogeneous.

Then, these files .txt must be compared with each other, for this, I decided to create only one vector of segmentations bringing together all the others as shown in the following example:

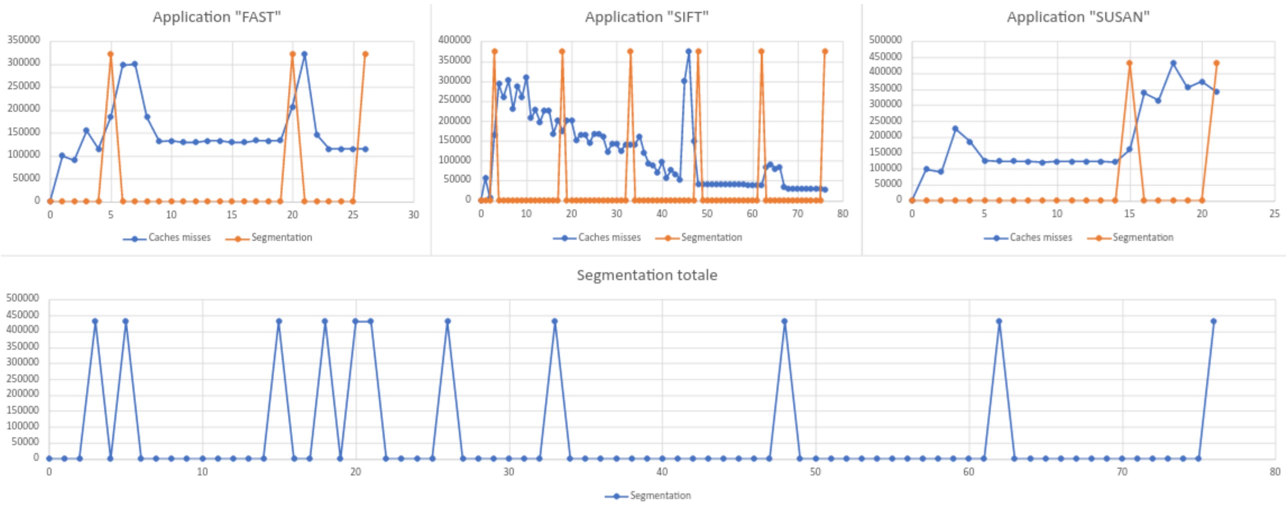

Figure 10: Diagram of the grouping of the different segmentations according to the files

On the diagram below, all segmentations have been added to form only one vector. Of course, for the application "FAST" and the application "SUSAN", the index does not go to more than 30, but here it is application graphs made previously by Mr. Danielsson that do not have the same duration.

To create this vector, I performed a function, which adds the segmentations of all the other vectors to a single vector, named "create_segmentation_and_avg_total" (Appendix 14). In addition, it sorts it and removes duplicates as the following result:

```
Vecteurs de segmentation par fichier:
 QVector(QVector(4, 19, 27), QVector(2, 17, 32, 47, 62, 77), QVector(14, 22))
Vecteur de toutes les segmentations:
 QVector(2, 4, 14, 17, 19, 22, 27, 32, 47, 62, 77)
```

Figure 11: Result of the first part of the « create_segmentation_and_avg_total » function

So we see that the last vector is the sum of the other three vectors.

This function also has another use. It calculates the average number of miss caches between each segmentation of this new vector to be able to analyze them afterwards. Here is the result:

```
Vecteur de toutes les segmentations:
 QVector(2, 4, 14, 17, 19, 22, 27, 32, 47, 62, 77)

Moyenne entre chaque segmentations :
 QVector(QVector(99719, 245016, 1.73774e+06, 390984, 265019, 662435, 604443, 0, 0, 0, 0), QVector(55820, 173255, 2.57518e+06, 620561, 375834, 552447, 807723, 691127, 1.97961e+06, 715909, 675761), QVector(98897, 315138,
1.28823e+06, 619907, 746477, 1.07012e+06, 0, 0, 0, 0, 0))
```

Figure 12: Result of the second artie of the « create_segmentation_and_avg_total » function

In this result, we can see that some mean sde vector have "0s". This is due to the difference in file size and therefore the program adapts the vectors. Indeed, if an application finishes earlier than the others, it is logical that its number of miss caches is 0 and therefore its average too.

Finally, all these intermediate steps completed, I was able to start creating a function that calculated the number of partitions based on the number of miss caches. To illustrate this, here is a diagram:

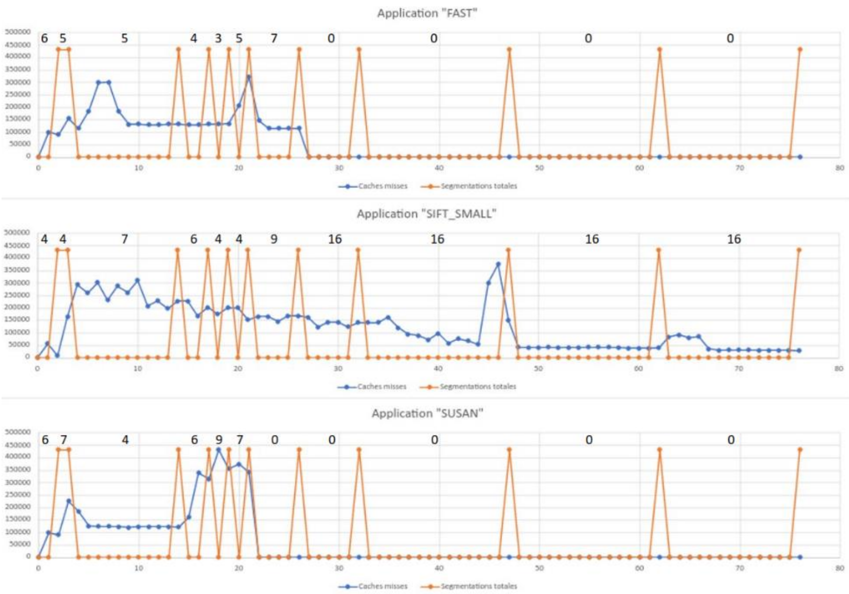

Figure 13: Scheme of partitioning applications according to their respective number of miss caches

The numbers at the top of each curve represent the number of partitions that could be granted to these different applications depending on the number of miss caches. But to achieve this, I created a function called "create_partition" (Appendix 15). This function makes a sum of all the averages between two segmentations, then makes a rule of three and multiplies by the number of available partitions (here 16 partitions). Once this is done, the partition values are put in their respective vectors, and its vectors are put in the partition vector as shown in the following result:

```
Vecteur des partitions en fonction du fichier :
QVector(QVector(6, 5, 5, 4, 3, 5, 7, 0, 0, 0, 0), QVector(4, 4, 7, 6, 4, 4, 9, 16, 16, 16, 16), QVector(6, 7, 4, 6, 9, 7, 0, 0, 0, 0, 0))
```

Figure 14: Result of the « create_partition » function

We therefore notice that the sharing corresponds roughly with the averages obtained before. In addition, the values are integer, this is due to a function I created because a partition is necessarily integer.

To be sure that my results are correct, I made the same calculations with the same files on the "excel" application and here are the results:

Segmentation	AVG "FAST"	AVG "SIFT_SMALL"	AVG "SUSAN"	SUM TOTAL	PART "FAST"	PART "SIFT_SMALL"	PART "SUSAN"
(0 - 2)	190172	63782	188444	442398	7	2	7
(2 - 4)	269199	458211	410486	1137896	4	6	6
(4 - 14)	1755003	2509094	1408930	5673027	5	7	4
(14 - 17)	391933	595156	813869	1800958	3	5	7
(17 - 19)	265350	374567	787253	1427170	3	4	9
(19 - 22)	529254	352286	714683	1596223	5	4	7
(22 - 27)	604443	807723	0	1412166	7	9	0
(27 - 32)	0	831829	0	831829	0	16	0
(32 - 47)	0	1988583	0	1988583	0	16	0
(47 - 62)	0	604802	0	604802	0	16	0
(62 - 77)	0	637195	0	637195	0	16	0

Figure 15: Result of partitioning applications according to their respective number of miss caches on "Excel"

We notice that the results are mostly the same between "excel" and my program. So I deduced that my calculations were good.

1.2.3 - Display of partitions and graphics according to the application

For the display part which is the most important part of my graphical interface, I proceeded in the same way as it is for the display of the scores as for the display of the graphics.

First of all, to display the results on the graphical interface, I had thought of doing as in Figure 13, that is, displaying the values above their respective curves. Neither succeeding, I had the idea to create a table where I will display all the partitions according to their application. For this, I created a table where the first row corresponded to the segmentations, and the other rows were created according to the number of files opened. Then, I just had to add the values inside it, as shown in the result below:

Segmentation	2	4	14	17	19	22
fast_normal	6	5	5	4	3	5
sift_small	4	4	7	6	4	4
susan_small	6	7	4	6	9	7

19	22	27	32	47	62	77
3	5	7	0	0	0	0
4	4	9	16	16	16	16
9	7	0	0	0	0	0

Figure 16: Viewing partitioning on the GUI

Then, to display the charts on the GUI, I also created a table that adds as many charts as open files. Since they are in a table, just scroll. I just had to adjust the parameters of Qt for the "plot" part and send it the desired values to obtain this result:

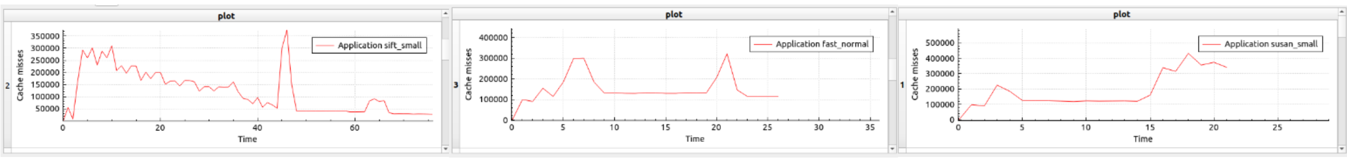

Figure 17: Display of the graphs of the "SIFT_SMALL", "FAST" and "SUSAN" applications on the graphical interface

We can zoom in and move the chart based on what you want to see as I was able to do for some of those charts.

1.2.4 - The rest of the GUI

In addition to all these features, I have two buttons that I use, one to launch the program once the user has selected his files that he wants to analyze them, and the other, to clean everything and select new file to avoid having to close the application and relaunch it each time.

1.3 - Conclusion for the GUI

To conclude, the graphical interface is finished and I have achieved the objectives set up with my tutor Mrs. Seceleanu and the doctoral student Mr. Danielsson. Several problems arose during the project but were quickly solved thanks to the debugging of it. To see the final result, I put you below the images of the gui:

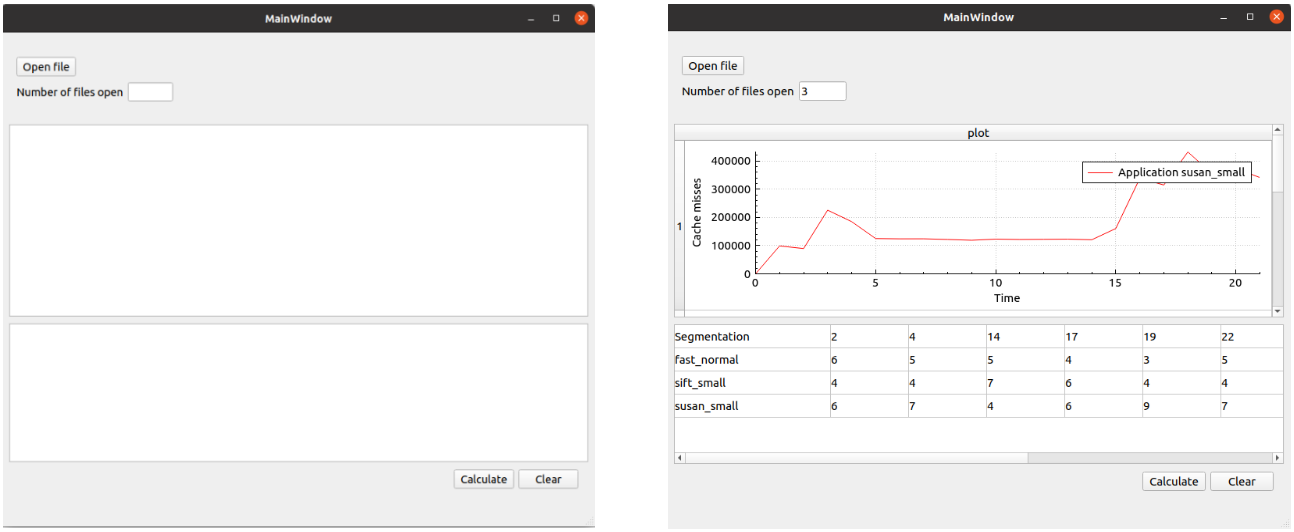

Figure 18: "Unused" and "used" GUI

2 – Assembling projects

After the realization of our two respective projects, we had to bring them together to form a single graphical interface. For this assembly, I had to create a class with all my functions and the GUI in relation to it. Once the class was made, I had to connect the "Analysis" button of the graphical interface of Mr. LAURENDEAU to this classse. For this, I created a function that when you click on this button, my graphical interface is displayed on a new window. Then after that, the user only has to use the interface as shown before. When the application is launched, we can access my graphical interface permanently except when the "Run" function of Mr. LAURENDEAU is running. This is a choice between him and me for a matter of practice in relation to the GUI. The "Run" function allows to display in real time, the values returned by the PAPI events and therefore for safety, no other function can be performed during the execution of it until it ends, or if you press the "Stop" button.

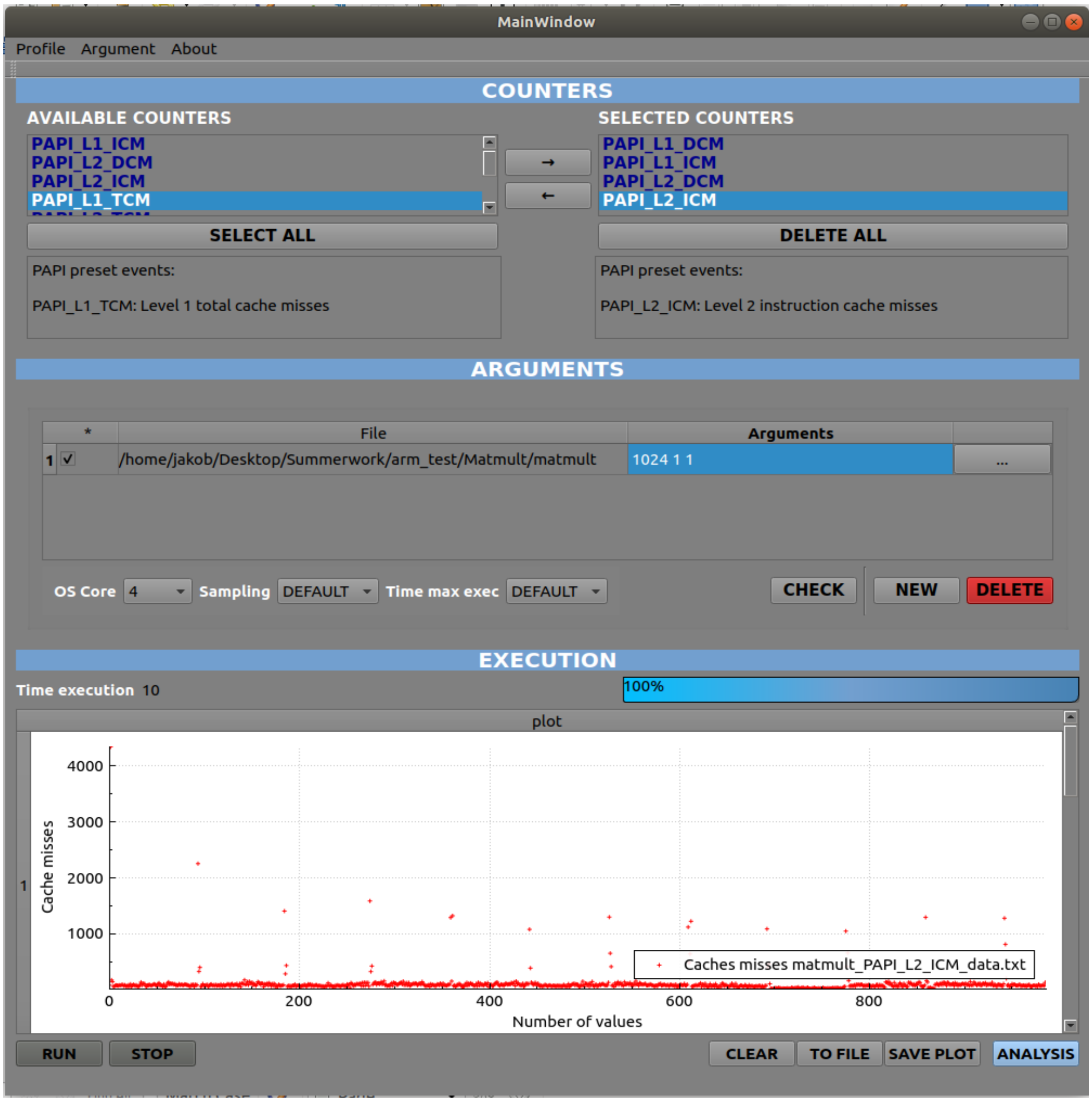

Figure 19: Full GUI

And when we click on "Analysis", we have this window that opens:

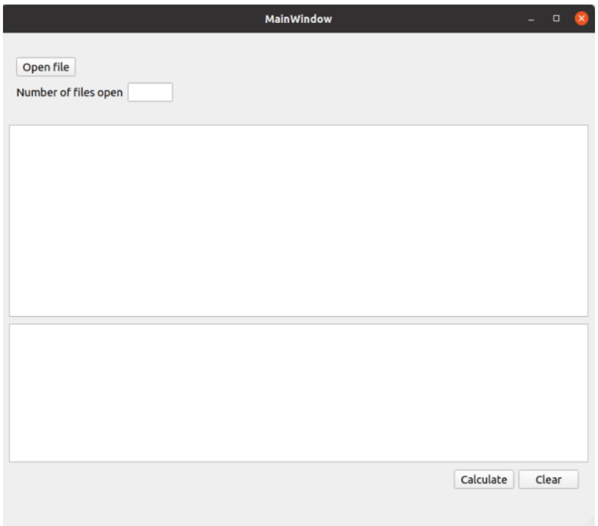

Figure 20: "Analysis" part of the GUI

3 – Conclusion

This internship, with the fact of traveling, learning and improving in English, was an excellent experience. The beginning was complicated between the subject, the understanding and the start of the program. But in the end, I was able to counter each difficulty to today make a good result that was accepted by my tutor Mrs. Seceleanu and the doctoral student Mr. Danielsson. This project will allow our PhD student and our tutors to visualize and compare live the miss caches of the computer that will be used during the applications.