MANUAL

Tom Laurendeau - French engineer in Microelectronic and Automation 06/22 - 08/22

Table of Contents

Acknowledgments

I would like to express my sincere gratitude towards my supervisors for their patience, help, and valuable discussions throughout this project of 3 months.

I would like to acknowledge each supervisor’s special contribution to my project; Tiberiu Seceleanu, for these instructions and kindness, who enlightened me. I am grateful to Jackob Danielsson for his knowledge and ability to guide me when I have become stuck.

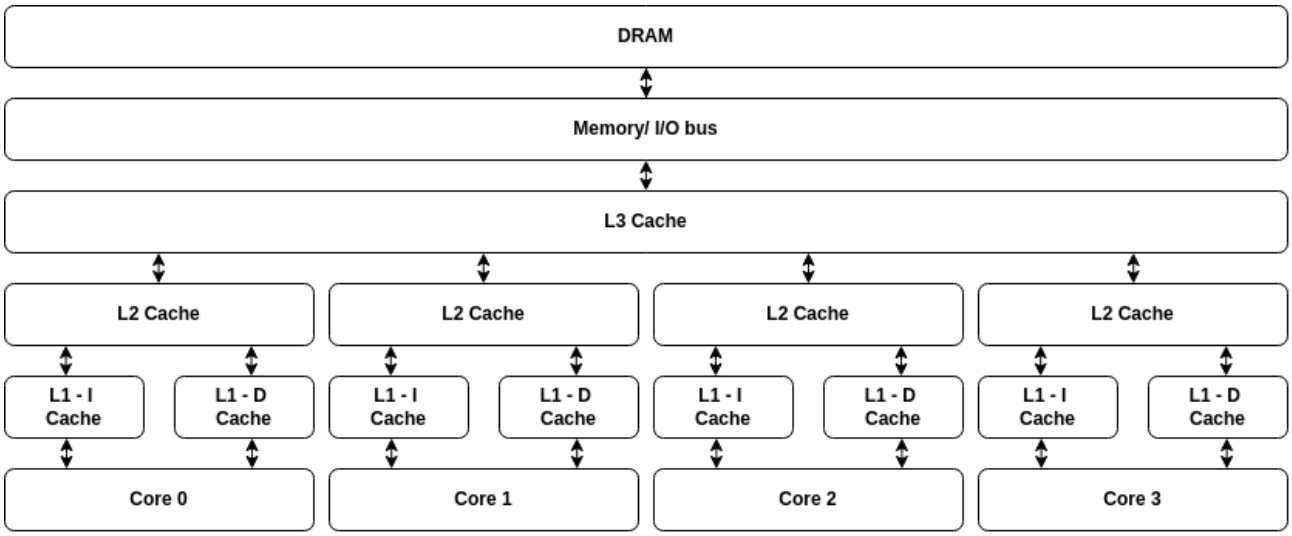
I want to thank Régis Jubeau, my lifelong and closest colleague. I am very grateful that I got the opportunity to work alongside you for the last few months and for the technical discussions that we have had.

Tom Laurendeau
Västerås, 2022

Introduction

Multi-core systems are becoming the de-facto standard in both commercial off-the-shelf (CoTS) and embedded computation domains. Multi-core processors present greater computational capacity while offering a decrease in size and weight than their single-core predecessors. Multi-core systems enable rationalization at both the application-level and system-level and simultaneously execute different applications on different cores. Multi-core systems often implement an internal shared-resource structure to increase inter-core communication speeds.

Jakob Danielsson, was a PhD student at Mälardalan University, his research interest includes software execution in multi-core computers. His thesis was on "Automatic Characterization and Mitigation of Shared-Resource Contention in Multi-core Systems". In his paper, he evaluated different methods for forecasting the resource usage. He evaluates auto-regressive, spline regressive, and exponential smoothing as approaches for modeling applications and their usage of CPU L2-cache L3-cache. Exemplify a typical shared resource-structure in Figure 1.



1: Example of a typical common 4-core system

The main objective was to create a graphic interface that would be added to run, display, and compare cache misses against the various Performance Application Programming Interface (*PAPI*) events.
PAPI enables software engineers to see, in near real time, the relation between software performance and processor events. In addition, PAPI provides access to a collection of components that expose performance measurement opportunities across the hardware and software stack (check with the link at the end to the github, all file are in the file papi).

In this manual we describe the set of tools used in the process. *Qt Creator*, which is a cross-platform integrated development environment that is part of the Qt framework. It is therefore oriented towards programming in C++. Through this graphic interface you will be able to find a lot of classes used, like :
The QMessageBox class provides a modal dialog for informing the user or for asking the user a question and receiving an answer.

The QTableWidget class provides an item-based table view with a default model.

The QListWidget class provides an item-based list widget.

The QDir class provides access to directory structures and their contents.

The QFile class provides an interface for reading from and writing to files.

The QTextStream class provides a convenient interface for reading and writing text.

The QFileDialog class provides a dialog that allow users to select files or directories.

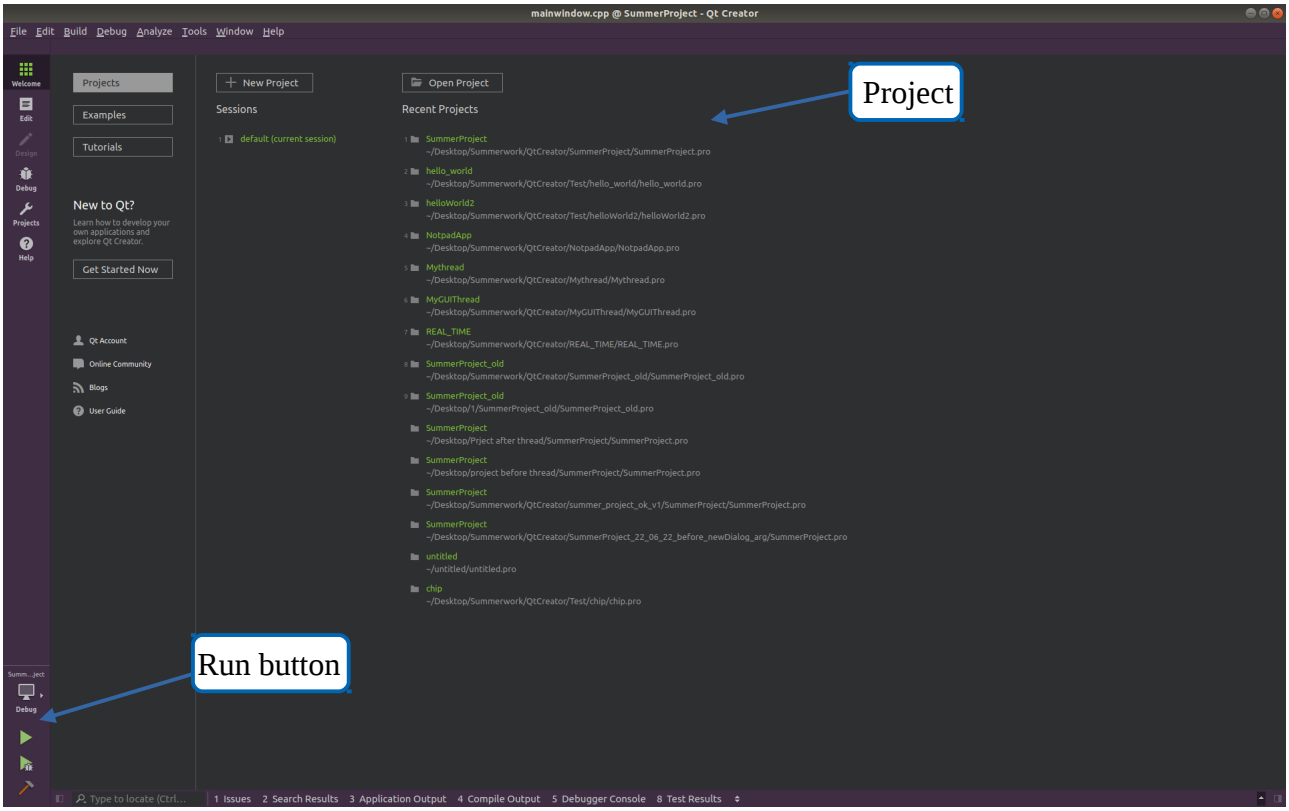
The QProcess class is used to start external programs and to communicate with them.

The QVector class is a template class that provides a dynamic array.

And still more, …

Start the interface

When you go to open Qt creator you will find this page, where you can see projects already opened in the past. If not go to the open project button, a dialog window will appear. Just go to where you saved the project in your computer and then, once you are in this directory, go to QtCreator→SummerProject. Finally open SummerProject.pro. Once the right project is open, just run it with the green play button at the bottom left.



2: Qt Creator menu

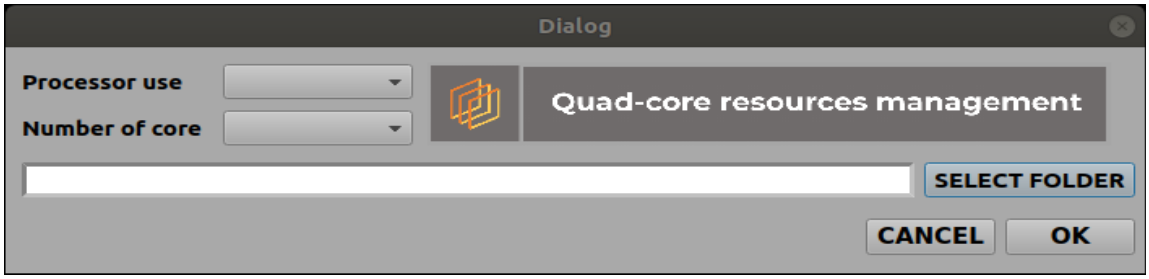
Newer Linux kernels have a sysf tunable “/proc/sys/kernel/perf_event_paranoid” which allows the user to adjust the available functionality of perf_events for non-root users, with higher numbers being more secure (offering correspondingly less functionality):

From the kernel documenation we have the following behavior for the various values:

> perf_event_paranoid:
>
> Controls use of the performance events system by unprivileged users (without CAP_SYS_ADMIN). The default value is 2.
>
> -1: Allow use of (almost) all events by all users Ignore mlock limit after perf_event_mlock_kb without CAP_IPC_LOCK
>
> >=0: Disallow ftrace function tracepoint by users without CAP_SYS_ADMIN Disallow raw tracepoint access by users without CAP_SYS_ADMIN
>
> >=1: Disallow CPU event access by users without CAP_SYS_ADMIN
>
> >=2: Disallow kernel profiling by users without CAP_SYS_ADMIN

For this application try to fix the kernel to 0. To do that go to a terminal and make the following command : “~$ sudo nano /proc/sys/kernel/perf_event_paranoid”
Enter the password, change the value to 0, ctrl+X then Y and enter to exit.

Dialog window



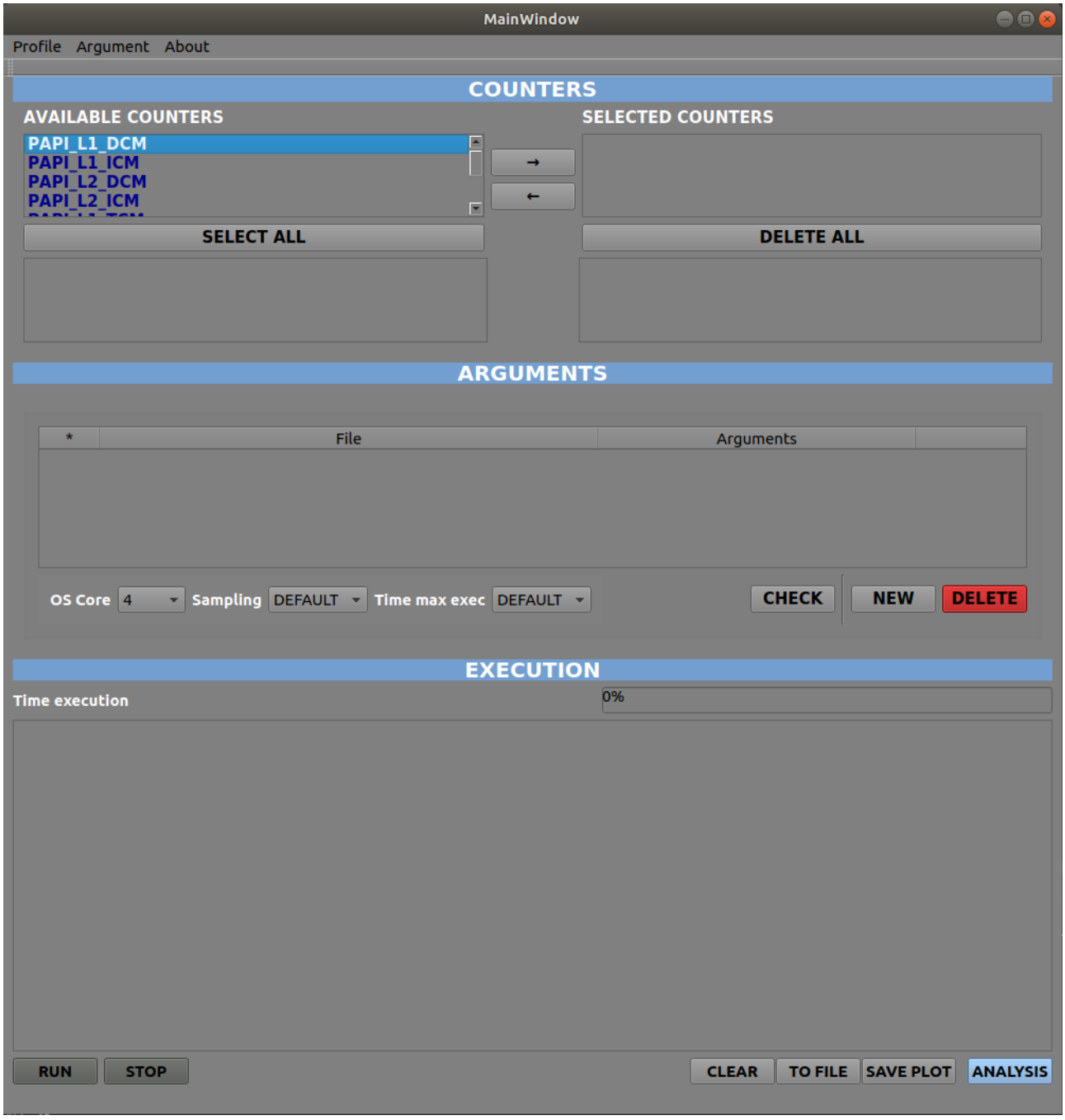
3: Dialog window

Once done, you should see a window called "dialog" appear.

This window is composed of:
- Two combo boxes, used to indicate the processor use and the number of cores ; if the number of cores is not known select the free option which finds the number thanks to a system command
- Buttons, the “select folder” button allows to open a dialog window to select where is saving the project, the other is useful to validate the information or exiting the window.
- One line edit, which will be filled with the access path selected previously with the help of the “select button” button.

When done, click “OK”, if any of the information is missing, a message box appears and informs what is missing (Processor, Core, Path).

Main window



4:Main window

Once this step is validated, the “MainWindow” appears.
It is separated into four parts: Tools Bar, Counters, Arguments, Execution.

Tools Bar

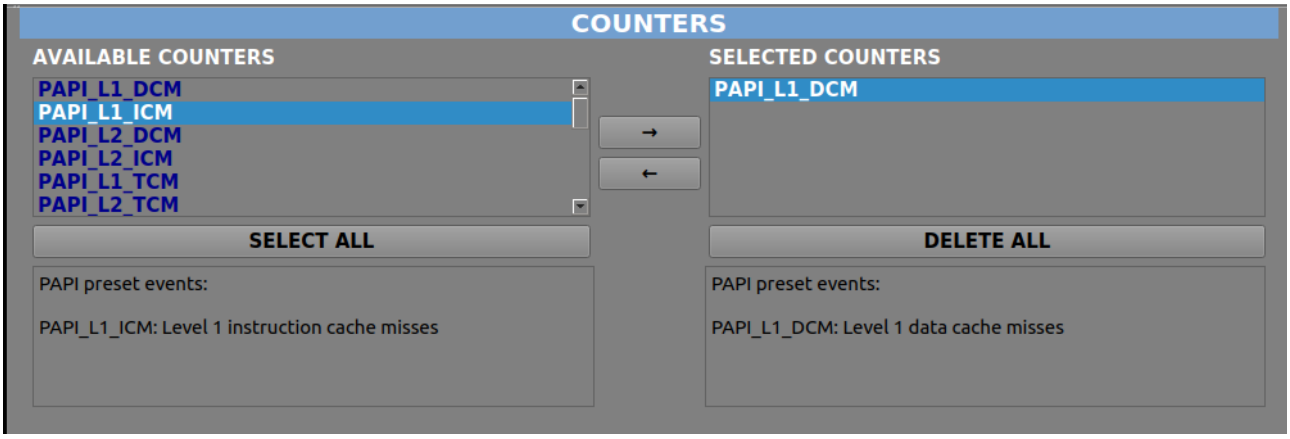
The tool bar allows you to make backups, especially those of the counters and arguments parts.

For counters, if we had to select X counters and we wanted to save this configuration, just go to the tools bar at the top of the profile tab and use either open/save/delete to do what we wish. Depending on the option chosen, there are different things to do, but in principle, a dialog window appears and allows you to choose the name if you save, the file if you want to delete, or open a file. In the background, for example, for the action "open profile", once the file is selected, the name is returned, the selected list and the description of counters are cleared. After that, the.txt file is read line by line and put the value in a table which it uses to increment by its value the selected list.

For the argument, as for the counter part it's possible to save, delete or open some argument profiles. It's possible to save the number you want to profile. Just check the line using the checkbox and in the background, a .txt file will be created with the line chosen. It can be open or deleted, depending on the chosen action.

Counters

First of all, you have to retrieve the available counters with the help of the PAPI library, which you will have to display in a list in order to be able to select them. To do this, I had to edit a "papi_avail" script and run it through the QProcess class. followed by the "-a" argument to specify that only available ones are displayed. This gives the following result, as you can see in the available counters list.
To find the script: "*Summerwork→executable→Print_Events_Avail.cpp*". Once you compile this code in the terminal (~$ gcc ../Print_Events_Avail.cpp -lpapi //don't forget the -lpapi, it's a reference to the PAPI library), you will have an output file, and just rename it as papi_avail if you want to change it. It's the only external process in this project. I think it will be possible to include it in the project in the future.



5: Counters part

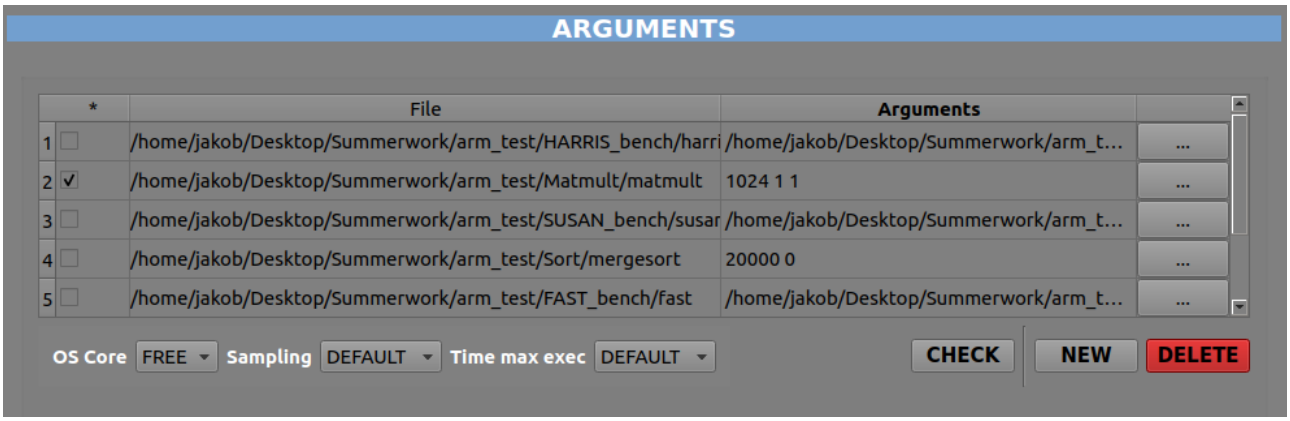
The two buttons with arrows are used to add or remove counters from the right list (selected counter), but I also possible to double-click a counter to move it in the selected list.
However, if in available counters have none counter a message will be displayed and will tell you what to modify in the kernel.
Be careful, it is possible to select everything with the "select all" button, but the maximum number of counters that can select for the execution part is 4. It is also possible to delete everything in the right list with the "delete all" button.

A description also seemed important to understand what the counters do. When a counter is selected in one of the lists, the software will read a.txt file and display the description corresponding to the counter selected in one of the lists.

Arguments

It is in this part that the settings are made. To properly launch the application, it needs, in addition to the path from where it is, some arguments. As you can see in the next picture, some examples of launched applications.

In fact, to execute one of them, each application needs its own specific arguments. Sometimes one argument is necessary, but more in other cases. As shown in Figure 8 (an example of how to use the application), it can be an image (.bmp,.pgm,...) or numbers. All arguments must be filled in manually. A free line edit is made for that, and everything is more specifically specified below, in particular with the "new" button.



6: Arguments part

The core value that was chosen in the first dialog window is defined and then initialized in a combo box, moreover, it is possible to change this value at any time. With the two others combo box they are useful to define the sampling (millisecond) and the time maximum of the execution of the application (second).

The “New” button opens a dialog window to select where is located the application, once validated a new dialog window appears and this time is to select the argument how is needed for the application, it can be numbers, (so cancel for application like matmult, mergesort, ...), or an image which the path is needed (like fast, susan, harris, ...).

When the second dialog window is validating a new row appears in the tableWidget with the path of the application in the second column and that of the argument in the third column.

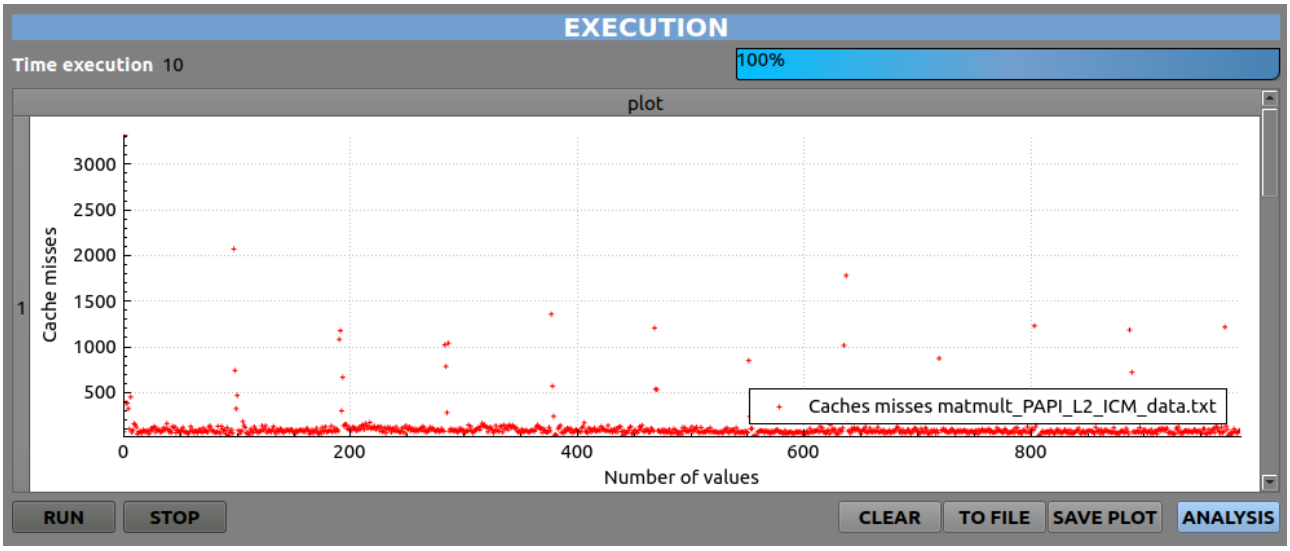
In the first and last column there are a checkbox and a button. The checkbox is used to select the application who will gunning (it is possible to select more that one, but only one can be run so just select the right application). The button is here to change the path of the argument, thanks to to a dialog window. To change the path of application double click on one box in the second column and another dialog window select a new application.
The “delete” button just removes the selected row.
The "check" button, it’s check if the selected counters are between 1 and 4, if only one row in the table widget in checked. If not, then a message appears to tell you what is wrong.

Don’t worry if you don’t remember how runs some application, there is a little recap just right here.

```
root@xilinx-zcu102-zu9-es2-rev1_0-2018:~/LLM_shark/my_files# cat programs.txt
/home/root/Hello/sift /home/root/Hello/256KBimg.pgm
/home/root/Hello/susan /home/root/Hello/landing_small.bmp
/home/root/Hello/fast /home/root/Hello/landing.bmp
/home/root/Hello/harris /home/root/Hello/landing_small.bmp
/home/root/Hello/matmult 200 0 0
/home/root/Hello/mergesort 20000 0
```

7: Example of the different way to use the application

Execution



8: Execution part

It is in this part that the display of the results is done.

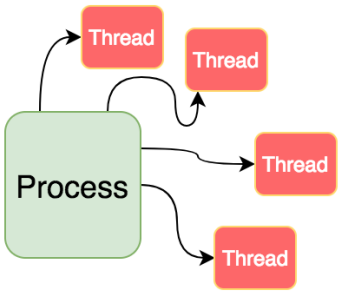
When the “run” button is clicked it calls the function of the “check” button to see if everything is good next it can run the selected application with the selected counters.
The application runs until the “stop” button is clicked or the time has reached the max time. To know the execution progress there are different indicators such as a progress bar in % or the execution time in second.

During the execution the plot displays only the last 500 points to have a clearer approach of what is happening, it displayed the cache misses in function of the number of values, approximately 100 values per second. The name of the plot is composed of the name of the running application and the name of the counter. There are the same number of plots as the counters selected, so the maximum number of plots is 4.

When the execution is finish the plot displays all the values saved and write a .txt file with the name of the application and the counter who had run for this plot. All .txt files are saved in a directory with the date and the hours of the ending execution.

At first, when the application was running, it was not possible to do something during its execution. When the plot(s) are finished, they are displayed. The problem with this method is that it cannot be

made a "real-time" application because we are just waiting for the end of the application. It can also not be stopped at any time. The perfect solution it’s to integrate some thread.



9: Thread illustration

A thread of execution is a sequence of instructions that can be executed concurrently with other such sequences in multi-threading environments, while sharing a same address space. And with a mutex, it is possible to trace or do something else during the sleep time of a thread which executes the application. The mutex is locked during the execution of one loop of the running application, and during the sleep time (sampling), it’s unlocked and locked by another thread who can plot during this time. The user can also click on the stop button during this time.

The “clear” button, remove all plot, this is useful if you want to plot another result and you don't want the others plots.

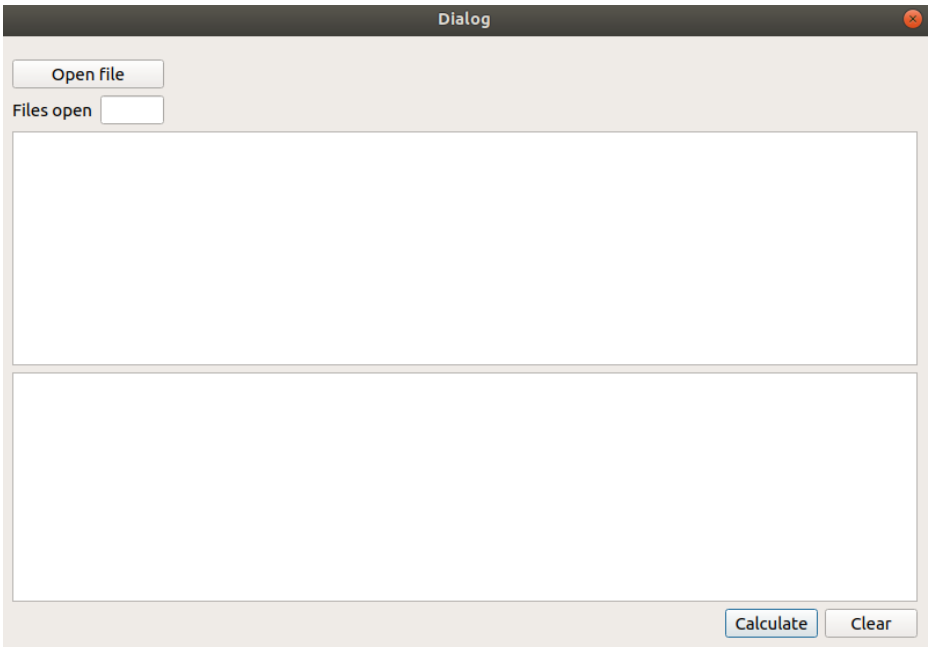
The “to file” button opens a dialog window with which you can open the .txt file to view the result in the plot.

The “save plot” button is if you want saved some result in a specific place in your computer. You can choose if you want to save only file(s) or a directory. To note, after each execution of an application, the results are put in a directory in the same place with the date and the hour of the ending execution. If you select the file option, the date and the hour of the ending execution are not lost because they are put in the name of the saved file.

The “analysis” button opens a dialog window where is located Mr. Jubeau’s part.

Assembling part

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 Mr. Jubeau’s functions and the GUI in relation to them. Once the class was made, I had to connect the "Analysis" button of my graphic interface to this class. For this, I created a function that when you click on this button, a dialogue window is displayed with Mr. Jubeau’s interface.



10: Interface of analysis part

Then after that, the user only has to use the interface as shown before. When the application is launched, the user can also access the dialog part, except when the "Run" function is running. This is a choice between him and me as a matter of practice.

Projection

The creation of an icon to start the interface, with this the user will not open the Qt application and will use it more simply.

Redo the plot part differently, being able to choose which of the counters must be displayed and not, only the last one currently.

ANNEX

Github : https://github.com/TomLaurendeau/Intership_Summer_2022.git

paper : automatic Characterization and Mitigation of Shared-Resource Contention in Multi-core System.

Reference : Qt documentation https://doc.qt.io/