



Rapport de stage

Stage réalisé par Régis JUBEAU à l'université de Mälardalen en Suède.
(pour la 4^{ième} année de Microélectronique et Automatique de l'école d'ingénieur
Polytech à l'université des sciences et des lettres de Montpellier)



Remerciements

Je tiens à remercier tout particulièrement Mme. Cristina SECELEANU, ma tutrice au sein de cette université pour son accueil.

Je souhaite également remercier M. Tiberiu SECELEANU pour son accompagnement et ses renseignements.

Je voudrais aussi remercier M. Jakob DANILESSON pour son sujet de projet, son soutien, son aide et ses conseils.

Table des matières

Remerciements	2
Introduction	4
1 – Déroulement de mon projet	5
1.1 - Introduction du projet de stage	5
1.2 - Réalisation de l'interface graphique	7
1.2.1 - Lecture des fichiers .txt sélectionnés par l'utilisateur	7
1.2.2 - Analyse des caches mises en fonction des fichiers	8
1.2.3 - Affichage des partitions et des graphiques en fonction de l'application	11
1.2.4 - Le reste de l'interface graphique	12
1.3 - Conclusion pour l'interface graphique	13
2 – Assemblage des projets	14
3 – Conclusion	15
4 – Annexes	16
Annexe 1 :	16
Annexe 2 :	17
Annexe 3 :	18
Annexe 5 :	19
Annexe 6 :	20
Annexe 7 :	20
Annexe 8 :	21
Annexe 9 :	22
Annexe 10 :	23
Annexe 11 :	23
Annexe 12 :	24
Annexe 13 :	24
Annexe 14 :	25
Annexe 15 :	26
Annexe 16 :	27
Annexe 17 :	28
Annexe 18 :	29
Annexe 19 :	30

Introduction

Mälardalen Höskola est une université suédoise dont un campus est situé à Västerås. Elle compte six spécialisations de recherche différentes : sciences de l'éducation et mathématiques, systèmes embarqués, énergie du futur, santé et bien-être, économie et gestion industries, et innovation et réalisation de produits. L'université de Mälardalen est organisée en quatre écoles dont l'école d'innovation, de design et d'ingénierie où j'effectue mon stage.

Mon stage avait pour principal objectif de créer une interface graphique sur le framework Qt codé en C++ qui serait additionnée à une autre interface graphique (réalisée par M. LAURENDEAU) pour compléter le projet global. L'interface graphique globale doit rendre meilleure l'exécution, l'affichage et la comparaison des caches mises par rapport au différents événements de PAPI (Performance Application Programming Interface) que vous découvrirez au fur et à mesure de ce rapport.

Vous suivrez également le chemin de ces trois mois passés auprès de chercheurs et thésards.

Vous découvrirez tous les problèmes et résolutions prises pour finir cette interface graphique et l'assemblage.

1 – Déroulement de mon projet

1.1 - Introduction du projet de stage

Pour introduire mon sujet de stage, voici une mise en situation. Les systèmes multicœurs sont en train de devenir la norme de fait dans les domaines de l'informatique commercial et de l'informatique embarquée. Les processeurs multicœurs présentent une plus grande capacité de calcul tout en offrant une diminution de la taille et du poids par rapport à leurs prédécesseurs à simple cœur. Les processeurs multicœurs permettent la parallélisation au niveau des applications et du système et l'exécution simultanée de différentes applications sur différents cœurs. Les systèmes multicœurs mettent souvent en œuvre une structure interne de ressources de déchiquetage pour augmenter la vitesse de communication entre les cœurs. Parmi les exemples de ressources partagées, citons les Translation lookaside Buffers (TLB), le Last Level Cache (LLC), le bus mémoire et le périphérique d'E/S à usage général. Voici un de structure de ressources partagées typique dans la figure ci-dessous :

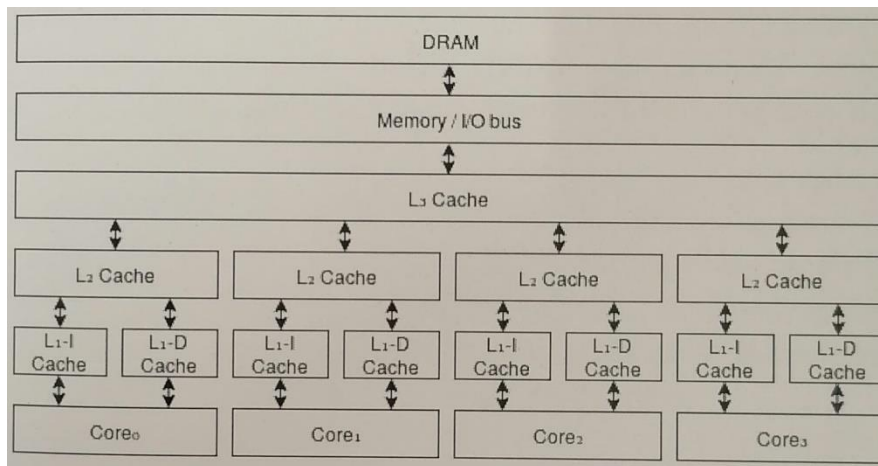


Figure 1: Exemple d'un système à 4 cœurs

Mon sujet de stage a été proposé par M. Danielsson, un doctorant de l'université de Mälardalen depuis Août 2016. Sa thèse, « [Automatic Characterization and Mitigation of Shared-Resource Contention in Multi-core Systems](#) », se concentrait sur les systèmes multi-cœurs homogènes qui utilisent deux ou plusieurs cœurs pour exécuter des applications. Elle ciblait le Last-level cache (LLC), un terme englobant tous les caches situés en dernier dans la hiérarchie de la mémoire. Le LLC est le plus souvent physiquement partagé entre les différents cœurs d'un processeur et relie la connexion entre le cache local d'un processeur et le bus mémoire. Il est particulièrement sujet à la contestation de ressources partagées puisqu'il est partagé entre différents cœurs. Lors de sa thèse, sa recherche visait principalement la contestation des ressources partagées, qui se produit en raison de l'utilisation simultanée de plusieurs cœurs.

Lors de sa thèse, il a réalisé un code où il a utilisé PAPI (Performance Application Programming Interface) qui est une interface de programmation portable permettant d'accéder aux compteurs matériels spécifiques aux microprocesseurs modernes. PAPI est utilisé pour collecter des informations de bas niveau, telles que le nombre d'opérations en virgule flottante par seconde, le nombre de cache

misses durant l'exécution d'un code, etc. Cette méthodologie possède une liste d'évènements et en voici un exemple que j'ai pu rencontrer lors de mon stage :

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 shutdowns
PAPI_CSR_FAL	0x8000001f	No	No	Failed store conditional instructions

Figure 2: Exemple d'évènement de PAPI

Lors de l'exécution de ce code, en fonction des counters choisis, un fichier .txt était retourné, composé de 3 colonnes : un index, le nombre d'opérations par seconde et le nombre de cache misses sous cette forme :

0	8029715.000000	0.000000
1	8741410.000000	109730.000000
2	12417641.000000	208470.000000
3	12583279.000000	32157.000000
4	12419414.000000	477.000000
5	12587119.000000	473.000000
6	12484801.000000	649.000000
7	12581100.000000	989.000000
8	12513776.000000	884.000000
9	12583567.000000	1322.000000
10	12516176.000000	1363.000000
11	12588374.000000	1822.000000
12	12487712.000000	1833.000000
13	12166951.000000	3023.000000
14	12164830.000000	3011.000000
15	12321748.000000	1507.000000
16	12465443.000000	1501.000000
17	12323771.000000	1031.000000
18	12334702.000000	1026.000000
19	12480070.000000	9062.000000
20	12489973.000000	15780.000000
21	12489848.000000	16820.000000
22	12481974.000000	22990.000000
23	12487526.000000	23777.000000
24	12444505.000000	21740.000000
25	12481961.000000	13801.000000
26	12394663.000000	15515.000000
27	12450511.000000	15164.000000
28	12393687.000000	39461.000000
29	12427847.000000	41661.000000
30	12387581.000000	49983.000000
31	12393631.000000	11118.000000
32	12347384.000000	11471.000000
33	12396831.000000	48061.000000
34	12251438.000000	17342.000000
35	12411602.000000	11717.000000
36	12480050.000000	11899.000000
37	12373809.000000	15064.000000
38	12480081.000000	61893.000000
39	12384544.000000	64093.000000
40	12289590.000000	64402.000000
41	12164919.000000	66216.000000
42	12161511.000000	17600.000000
43	12185112.000000	69712.000000
44	12153711.000000	68812.000000
45	12164730.000000	69810.000000
46	12164768.000000	7874.000000
47	12242969.000000	75121.000000
48	12172819.000000	71400.000000
49	12177070.000000	71204.000000
50	12156440.000000	83050.000000

Figure 3: Exemple de fichier .txt

Donc, l'objectif de mon stage a été de réaliser une interface graphique sur le framework Qt codée en C++. Cette interface graphique devait lire des fichiers .txt, et les analyser. Le but étant d'avoir des fichiers du même événement mais réalisé avec des applications différentes. Et affecter à chacune des applications, une nombre de partition en fonction du nombre de caches misses.

1.2 - Réalisation de l'interface graphique

Une image de l'interface graphique sera mise à la fin de cette partie (figure 18).

Pour les exemples, ce sera toujours le même évènement « PAPI_L2_DCA » avec les mêmes applications (« FAST », « SIFT », « SUSAN ») pour voir l'évolution au fur et à mesure de l'explication.

1.2.1 - Lecture des fichiers .txt sélectionnés par l'utilisateur

Pour le projet, il fallait que ce soit l'utilisateur qui ait le choix des fichiers .txt qu'il voulait analyser. Pour cela, j'ai créé un bouton qui nous amène dans les fichiers de l'ordinateur. Ensuite, l'utilisateur n'a plus qu'à sélectionner ce qu'il souhaite.

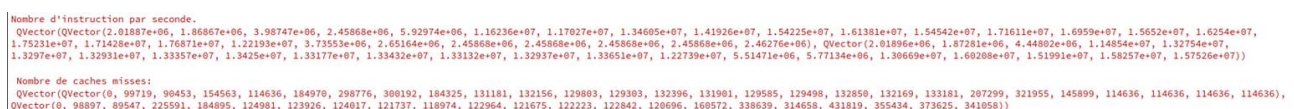


```
void MainWindow::on_btn_select_file_clicked()
{
    if (numberCore < QThread::idealThreadCount())
    {
        file_name = QFileDialog::getOpenFileName(this, "Open files",
        "/home/mdh/Desktop/Summerwork2022/new_small_tests");
        file_path.append(file_name);
        numberFile++;
        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: Fonction du bouton "OPEN"

On peut voir sur cette image, le code pour le bouton. Grâce à Qt et ses fonctions, dès que l'utilisateur cliquera sur le bouton, il rentrera dans cette fonction. Cette fonction récupère le nom du fichier sélectionné qu'elle enregistrera en l'ajoutant dans le vecteur « file_path ». Cette fonction vérifiera aussi le nombre de fichier ouvert par rapport aux nombres de core que possède l'ordinateur d'où l'indicateur de fichier ouvert. En cas d'un nombre supérieur, une alerte s'ouvrira.

Pour la partie lecture du fichier, c'est la fonction « create_vector » (Annexe 1) qui s'en occupe. À savoir, les fichiers .txt sont écrits sous forme de colonnes et elles sont toutes séparées par le caractère « tabulation ». La fonction récupère le vecteur contenant le nom de chacun des fichiers sélectionné et les ouvre un par un. Une fois ouvert, la fonction lit chaque valeur (double) et l'ajoute dans un vecteur qui lui correspond. Puis ses vecteurs sont eux-même ajoutés dans un autre vecteur ce qui me permet d'avoir dans un seul vecteur toutes les valeurs du nombre d'instructions ou encore le nombre de caches misses. Voici le résultat de cette fonction en 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, 2.46276e+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, 1.38669e+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, 380192, 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: Résultat de la fonction "create_vector"

En n'ouvrant que trois fichiers, on voit qu'un vecteur en comprend trois.

1.2.2 - Analyse des caches misses en fonction des fichiers

Pour cette partie, je devais partager un nombre de partitions (16 partitions) en fonction du nombre de caches misses de chaque application faite sur un évènement. Pour cela, j'ai procédé par étape.

Tout d'abord, je devais réaliser une fonction qui trouvait les segmentations de mes vecteurs. Voici un schéma approximatif et représentatif de ma fonction :

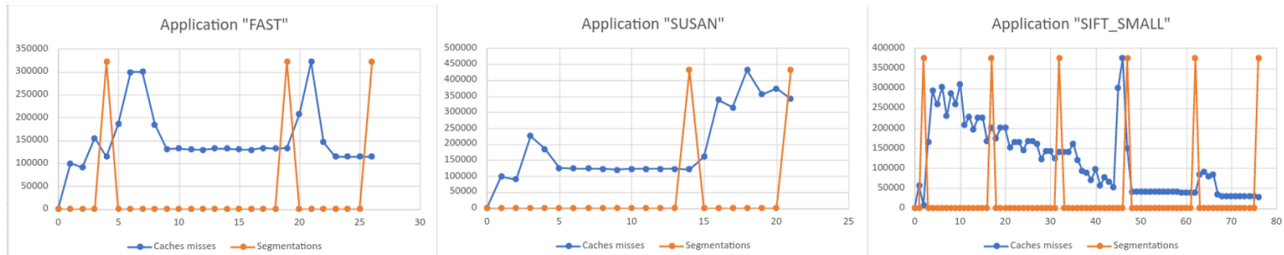


Figure 6: Schéma de la segmentation des fichiers

Pour la créer, j'ai créé une autre fonction qui me renvoyait la moyenne globale du vecteur que je mettais en entrée, et j'analysais par la suite, chaque valeur du segment par rapport à celle-ci. Si une des valeurs était inférieure à 80 % de la moyenne, alors la fonction crée une segmentation. Cette fonction est nommée « find_segmentation » (Annexe 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: Fonction "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: Résultat de la fonction "find_segmentation"

D'après les résultats, on peut voir qu'il y a des segmentations consécutives trop proches que nous allons supprimer par la suite.

Pour chaque vecteur de segmentations, je vérifiais que les segmentations n'étaient pas trop proches ni trop éloignées pour éviter d'avoir un nombre trop important ou peu important de segmentations.

Pour cela, je vérifiais si la différence de chaque valeur consécutive était supérieur à 10 ou supérieure à 20. Dans ces cas là, soit je supprimais la valeur inférieure pour ne former plus qu'une seule segmentation, soit j'ajoutais une segmentation à environ +15 par rapport à la plus petite valeur.

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

Figure 9: Résultat de la fonction "correction_seg"

D'après le résultat, nous sommes passés pour le premier vecteur de 9 segmentations à 3, pour le deuxième de 15 segmentations à 2 et pour le troisième de 40 à 6. Dans ce résultat, on peut voir pour le premier vecteur, qu'il y a la segmentation « 19 » qui n'apparaissait pas de base. Elle a été ajoutée car la différence entre 27 et 4 était supérieure à 20. Ce qui permet aux vecteurs de segmentations d'être plus homogène.

Ensuite, ces fichiers .txt doivent être comparés entre eux, pour cela, j'ai décidé de ne créer qu'un seul vecteur de segmentations réunissant tous les autres comme le montre l'exemple suivant :

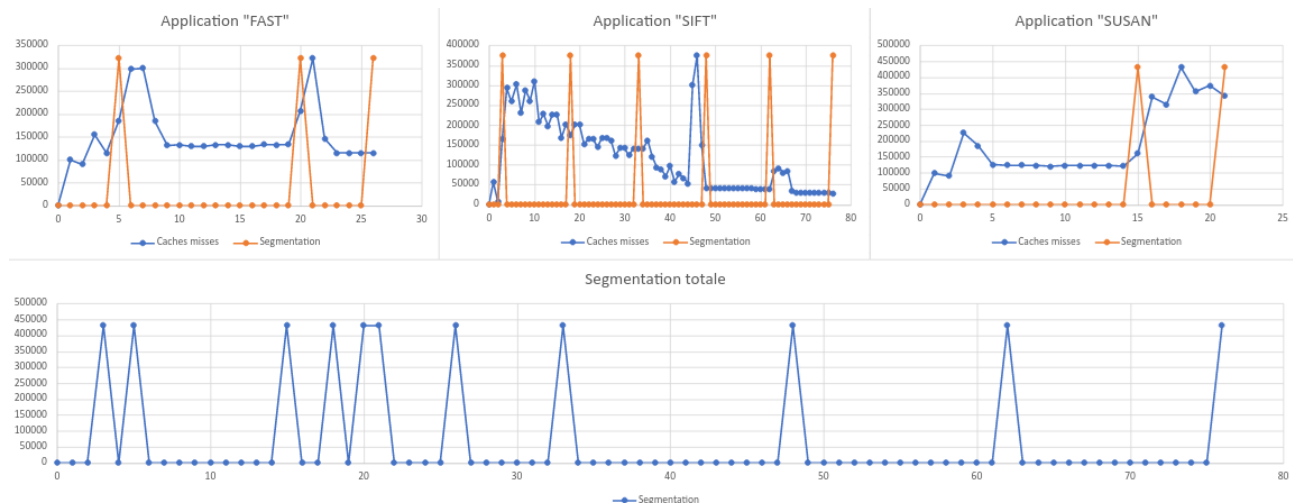


Figure 10: Schéma du rassemblement des différentes segmentations en fonction des fichiers

Sur le schéma du dessous, toutes les segmentations ont été ajoutées pour ne former qu'un vecteur. Bien sûr, pour l'application « FAST » et l'application « SUSAN », l'index ne va pas à plus de 30, mais ici ce sont des graphiques d'application réalisés auparavant par Mr. Danielsson qui n'ont pas la même durée.

Pour créer ce vecteur, j'ai réalisé une fonction, qui ajoute les segmentations de tous les autres vecteurs à un seul vecteur, nommée « create_segmentation_and_avg_total » (Annexe 14). De plus, elle le trie et supprime les doublons comme le résultat suivant :

```
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: Résultat dde la première partie de la fonction "create_segmentation_and_avg_total"

On voit donc que le dernier vecteur est la somme des trois autres vecteurs.

Cette fonction a aussi une autre utilité. Elle calcule la moyenne du nombre de caches misses entre chaque segmentation de ce nouveau vecteur pour pouvoir les analyser après. Voici le résultat :

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, 399984, 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: Résultat de la deuxième partie de la fonction "create_segmentation_and_avg_total"

Dans ce résultat, on peut voir que certains vecteurs de moyenne possèdent des « 0 ». Ceci est dû à la différence de taille des fichiers et donc le programme adapte les vecteurs. En effet, si une application finit plus tôt que les autres, il est logique que son nombre de caches misses soit à 0 et donc sa moyenne aussi.

Enfin, toutes ces étapes intermédiaires réalisées, j'ai pu commencer à créer une fonction qui calculait le nombre de partition en fonction du nombre de caches misses. Pour illustrer cela, voici un schéma :

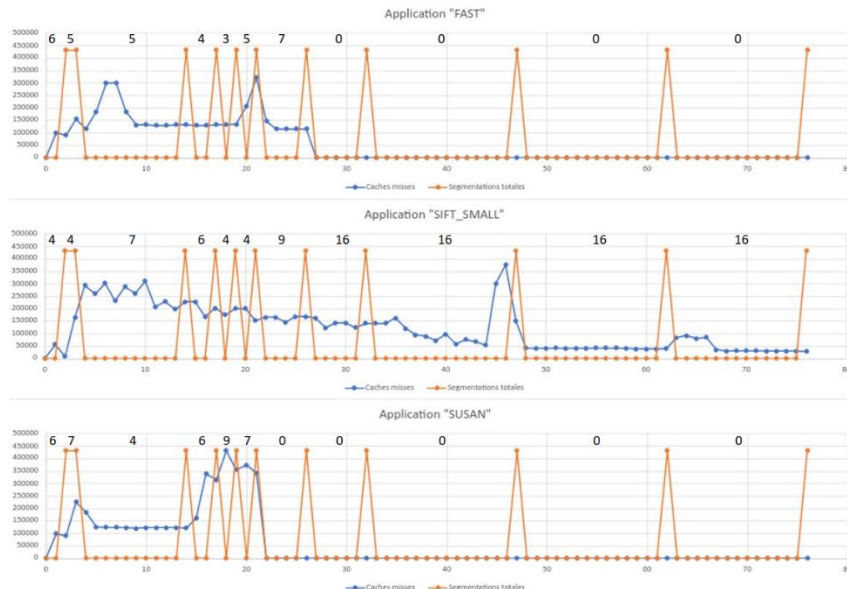


Figure 13: Schéma du partitionnement des applications en fonction de leur nombre de caches misses respectif

Les nombres situés en haut de chaque courbe représentent le nombre de partitions que l'on pourrait accorder à ces différentes applications selon le nombre de caches misses. Mais pour réaliser ceci, j'ai créé une fonction nommée « create_partition » (Annexe 15). Cette fonction fait une somme de toutes les moyennes entre deux segmentations, puis réalise une règle de trois et multiplie par le nombre de partitions disponibles (ici 16 partitions). Ceci réalisé, les valeurs de partitions sont mises dans leur vecteur respectif, et ses vecteurs sont mis dans le vecteur des partitions comme le montre le résultat suivant :

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: Résultat de la fonction "create_partition"

On remarque donc que le partage correspond à peu près avec les moyennes obtenues avant. De plus, les valeurs sont entières, ceci est dû à une fonction que j'ai créée car une partition est forcément entière.

Pour être sûr que mes résultats soient corrects, j'ai fait les mêmes calculs avec les mêmes fichiers sur l'application « excel » et en voici les résultats :

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: Résultat du partitionnage des applications en fonction de leur nombre de caches misses respectif sur "Excel"

On remarque que les résultats sont pour la plupart les mêmes entre « excel » et mon programme. J'en ai donc déduit que mes calculs étaient bons.

1.2.3 - Affichage des partitions et des graphiques en fonction de l'application

Pour la partie affichage qui est la partie la plus importante de mon interface graphique, j'ai procédé de la même manière que ce soit pour l'affichage des partitions que pour l'affichage des graphiques.

Tout d'abord, pour afficher les résultats sur l'interface graphique, j'avais pensé à faire comme sur la figure 13, c'est-à-dire afficher les valeur au-dessus de leur courbe respective. Ni parvenant pas, j'ai eu l'idée de créer un tableau où j'afficherai toutes les partitions en fonction de leur application. Pour cela, j'ai créé un tableau où la première ligne correspondait aux segmentations, et les autres lignes se créaient en fonction du nombre de fichiers ouverts. Ensuite, je n'avais plus qu'à ajouter les valeurs à l'intérieur de celui-ci, comme le montre le résultat en dessous :

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

Figure 16: Affichage du partitionnage sur l'interface graphique

Ensuite, pour afficher les graphiques sur l'interface graphique, j'ai également créé un tableau qui ajoute autant de graphique que de fichiers ouverts. Étant donné qu'ils sont dans un tableau, il suffit juste de scroller. Il me suffisait juste de régler les paramètres de Qt pour la partie « plot » et de lui envoyer les valeurs voulues pour obtenir ce résultat :

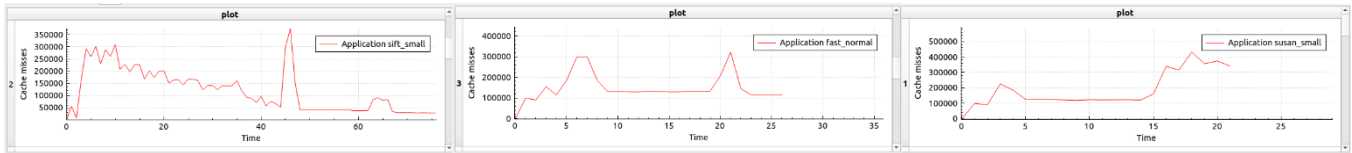


Figure 17: Affichage des graphiques des applications "SIFT_SMALL", "FAST" et "SUSAN" sur l'interface graphique

Nous pouvons zoomer et déplacer le graphique en fonction de ce que l'on veut voir comme j'ai pu le faire pour certains de ces graphiques.

1.2.4 - Le reste de l'interface graphique

En plus de toutes ces fonctionnalités, j'ai deux boutons qui me servent, un à lancer le programme une fois que l'utilisateur a sélectionné ses fichiers qu'il veut les analyser, et l'autre, pour tout nettoyer et sélectionner de nouveau fichier pour éviter de devoir fermer l'application et la relancer à chaque fois.

1.3 - Conclusion pour l'interface graphique

Pour conclure, l'interface graphique est finie et j'ai atteint les objectifs mis en place avec ma tutrice Mme. Secleanu et le doctorant M. Danielsson. Plusieurs problèmes sont survenus lors du projet mais ont été assez vite réglés grâce au débogage de celui-ci. Pour voir le résultat final, je vous mets en-dessous des images de l'interface graphique :

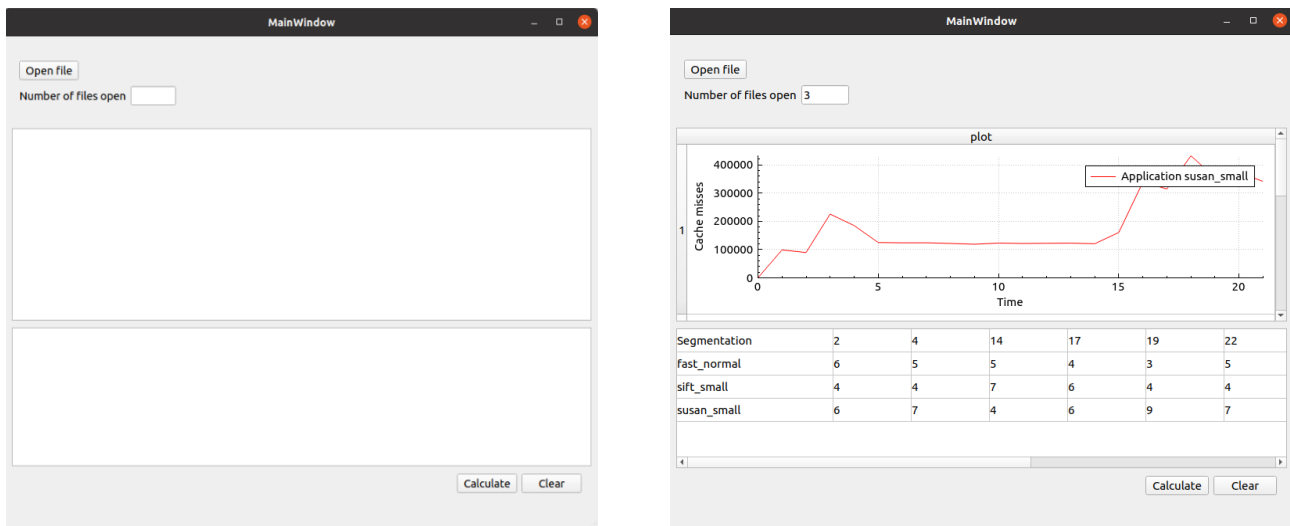


Figure 18: Interface graphique "non utilisée" et "utilisée"

2 – Assemblage des projets

Après la réalisation de nos deux projets respectifs, nous avons dû les rassembler pour ne former qu'une seule interface graphique. Pour cet assemblage, j'ai dû créer une classe avec toutes mes fonctions et l'interface graphique par rapport à celle-ci. Une fois la classe réalisée, il fallait que je relie le bouton « Analysis » de l'interface graphique de M. LAURENDEAU à cette classe. Pour cela, j'ai créé une fonction qui lorsque l'on clique sur ce bouton, mon interface graphique s'affiche sur une nouvelle fenêtre. Puis après, l'utilisateur n'a plus qu'à utiliser l'interface comme indiqué auparavant. Lorsque l'application est lancée, nous pouvons accéder à mon interface graphique en permanence sauf quand la fonction « Run » de M. LAURENDEAU est en marche. Ceci est un choix entre lui et moi pour une question de pratique par rapport à l'interface graphique. La fonction « Run » permet d'afficher en temps réel, les valeurs renvoyées par les événements PAPI et donc par sécurité, aucune autre fonction ne peut être réalisée lors de l'exécution de celle-ci jusqu'à temps qu'elle se termine, ou si vous appuyez sur le bouton « Stop ».

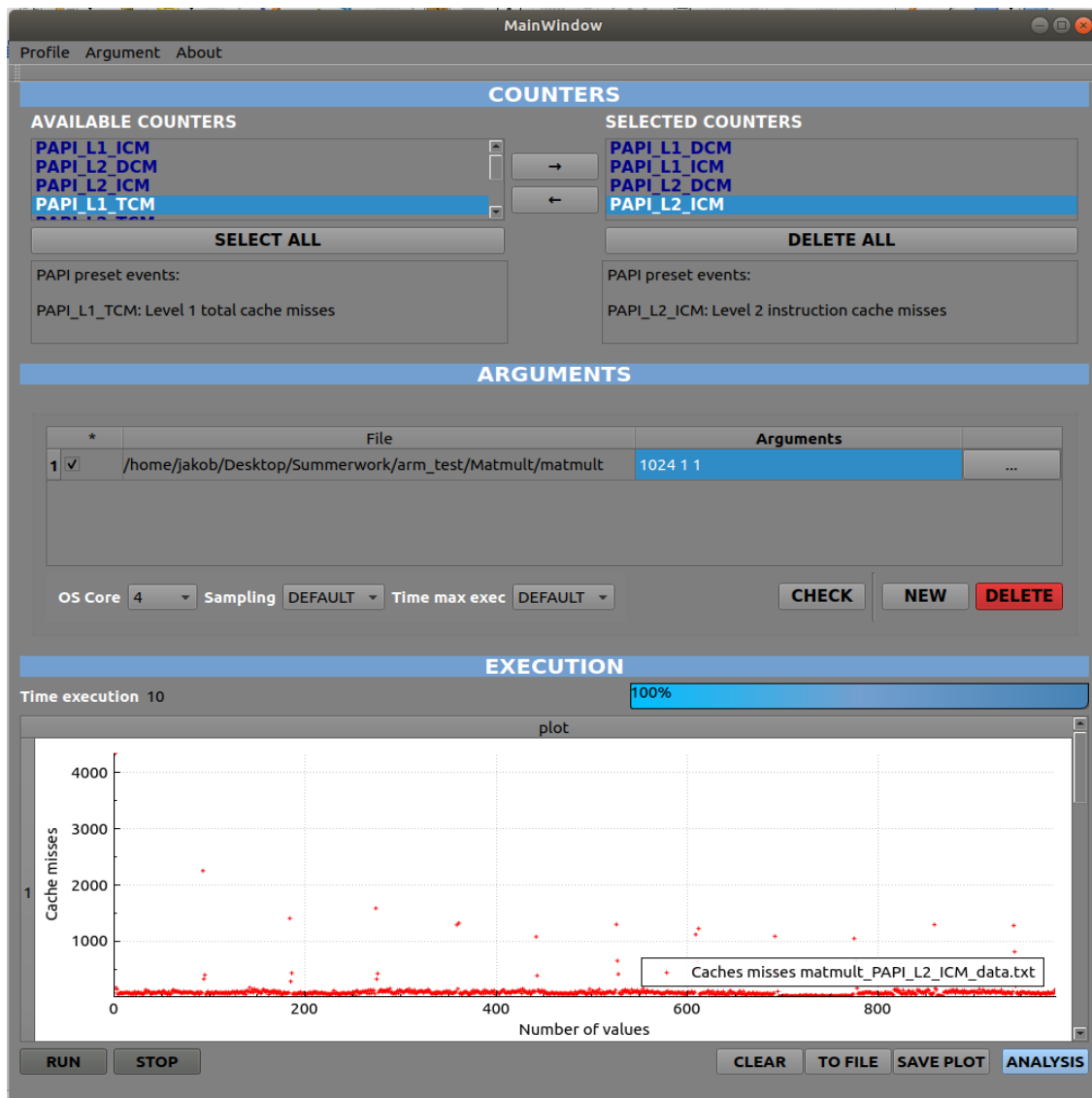


Figure 19: Interface graphique complète

Et lorsque l'on clique sur « Analysis », nous avons cette fenêtre qui s'ouvre :

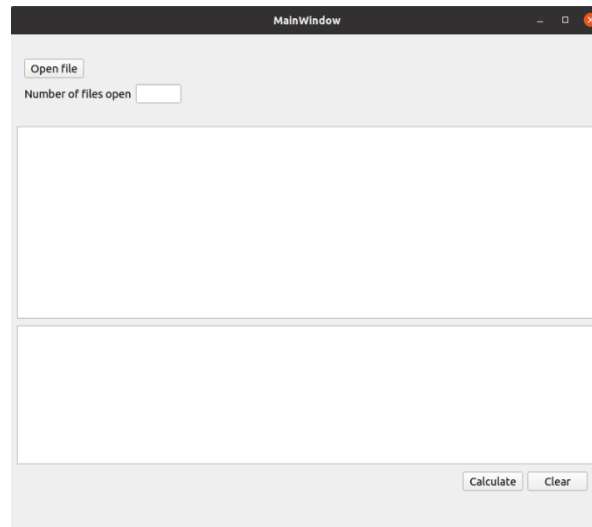


Figure 20: Partie "Analysis" de l'interface graphique

3 – Conclusion

Ce stage, avec le fait de voyager, d'apprendre et de se perfectionner en anglais, fut une excellente expérience. Le début fut compliqué entre le sujet, la compréhension et la mise en route du programme. Mais au final, j'ai su contrer chaque difficulté pour aujourd'hui rendre un bon résultat qui a été accepté par ma tutrice Mme. Seceleanu et le doctorant M. Danielsson. Ce projet va permettre à notre doctorant et nos tuteurs de visualiser et de comparer en direct les caches mises de l'ordinateur qui seront utilisés au cours des applications.

4 – Annexes

Annexe 1 :

```

/*****
 * Name : create_vector
 *
 * Input variable: QVector<QString> *file_name
 *                 QVector<QVector<double>> *colonne0
 *                 QVector<QVector<double>> *colonne1
 *                 QVector<QVector<double>> *colonne2
 *
 * Output variable: -
 *
 * Summary of function: This function allows you to read text files that you
 *                      have previously selected. While reading the files,
 *                      it adds each value to its corresponding vector.
 *****/
void MainWindow::create_vector(QVector<QString> *file_name, QVector<QVector<double>> *colonne0,
                              QVector<QVector<double>> *colonne1, QVector<QVector<double>> *colonne2)
{
    QVector<double> index;
    QVector<double> instr;
    QVector<double> cache_mises;
    bool ok;
    for (int j = 0; j < file_name->size(); j++)
    {
        QFile file(file_name->at(j));
        if(!file.open(QIODevice::ReadOnly))
        {
            QMessageBox::information(0, "error", file.errorString());
        }

        QTextStream in(&file);

        instr.clear();
        cache_mises.clear();

        int i = 0;
        while(!in.atEnd())
        {
            QString line = in.readLine();
            QStringList words = line.split("\t");
            foreach(QString word, words)
            {
                if (i == 0)
                {
                    index.append(word.toDouble(&ok));
                    i++;
                }
                else if (i == 1)
                {
                    instr.append(word.toDouble(&ok));
                    i++;
                }
                else if (i == 2)
                {
                    cache_mises.append(word.toDouble(&ok));
                    i = 0;
                }
            }
        }
        colonne0->append(index);
        colonne1->append(instr);
        colonne2->append(cache_mises);
        file.close();
    }
}

```

Annexe 2 :

```

/*****
 * Name : find_segmentation
 *
 * Input variable: QVector<QVector<double>> *tb
 *                 QVector<QVector<double>> *segm
 *
 * Output variable: -
 *
 * Summary of function: This function allows to find the segmentations from
 *                       the global average of a vector. To do this, it takes
 *                       each vector, calculates its average and as soon as a
 *                       value is less than 80% of the average, it creates a
 *                       segmentation.
 *****/
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();
    }
}

```

Annexe 3 :

```

/*****
 * Name : avg_by_seg
 *
 * Input variable: QVector<double> *segm
 *                 QVector<double> *colonne2
 *
 * Output variable: QVector<double>
 *
 * Summary of function: This function calculates the average of
 *                      the vector between each segmentation of it.
 *****/
QVector<double> MainWindow::avg_by_seg(QVector<double> *segm, QVector<double> *colonne2)
{
    double sum;
    QVector<double> avg_tb;
    avg_tb.clear();
    int seg = 0;

    for(int i = 0; i < segm->size(); i++)
    {
        sum = 0;
        for (int l = seg; l < segm->at(i); l++)
        {
            sum += colonne2->at(l);
        }
        seg = segm->at(i);
        avg_tb.append(sum);
    }
    return avg_tb;
}

```

Annexe 4 :

```

/*****
 * Name : DoubleToInt
 *
 * Input variable: double a
 *
 * Output variable: int b
 *
 * Summary of function: This function converts a double to an int.
 *****/
int MainWindow::DoubleToInt(double a)
{
    int b = a;
    int reste = (a-b)*10;

    if (reste >= 5)
        return (b+1);
    else
        return b;
}

```

Annexe 5 :

```

/*****
 * Name : correction_seg
 *
 * Input variable: -
 *
 * Output variable: -
 *
 * Summary of function: This function checks if the values of the segmentations are not
 *                      too close on 2 consecutive values ( < 10). If this is the case,
 *                      then the lower segmentation is deleted.
 *                      Furthermore, if the difference between each segmentation is too
 *                      great ( > 20) then it creates a new segmentation between the two
 *                      at +15 compared to the first one.
 *****/
void MainWindow::correction_seg()
{
    for (int t = 0; t < segm.size(); t++)
    {
        for (int i = 0; i < segm[t].size()-2; i++)
        {
            if ((segm[t].at(i+1)-segm[t].at(i)) < 10)
            {
                if ((segm[t].at(i+2)-segm[t].at(i+1)) < 10)
                {
                    segm[t].remove(i+1);
                }
                segm[t].remove(i);
                i--;
            }
        }
    }

    for (int t = 0; t < segm.size(); t++)
    {
        for (int i = 0; i < segm[t].size()-1; i++)
        {
            if ((segm[t].at(i+1) - segm[t].at(i)) > 20)
                segm[t].insert(i+1, (segm[t].at(i)+15));
        }
    }
}
```

Annexe 6 :

```

/*****
* Name : avg_by_seg
*
* Input variable: QVector<double> *segm
*                 QVector<double> *colonne2
*
* Output variable: QVector<double>
*
* Summary of function: This function calculates the average of
*                      the vector between each segmentation of it.
*****/
QVector<double> MainWindow::avg_by_seg(QVector<double> *segm, QVector<double> *colonne2)
{
    double sum;
    QVector<double> avg_tb;
    avg_tb.clear();
    int seg = 0;

    for(int i = 0; i < segm->size(); i++)
    {
        sum = 0;
        for (int l = seg; l < segm->at(i); l++)
        {
            sum += colonne2->at(l);
        }
        seg = segm->at(i);
        avg_tb.append(sum);
    }
    return avg_tb;
}

```

Annexe 7 :

```

/*****
* Name : creation_table_to_comparison
*
* Input variable: QVector<double> *segm
*                 QVector<double> *segmentation
*
* Output variable: -
*
* Summary of function: This function creates a new vector of all the combined
*                      segmentations of the different selected text files
*****/
void MainWindow::creation_table_to_comparison(QVector<QVector<double>> *segm, QVector<double> *segmentation)
{
    for(int i = 0; i < segm->size(); i++)
    {
        for (int j = 0; j < segm[i].size(); j++)
        {
            segmentation->append(segm[i].at(j));
        }
    }
}

```

Annexe 8 :

```

/*****
 * Name : tri_and_double
 *
 * Input variable: QVector<double> *segmentation
 *
 * Output variable: -
 *
 * Summary of function: This function sorts the vector and removes duplicates
 *****/
void MainWindow::tri_and_double(QVector<double> *segmentation)
{
    int i,j,k;
    double c;
    QVector<double> essaie;

    for (int i = 0; i < segmentation->size(); i++)
        essaie.append(segmentation->at(i));

    for(i=1;i<essaie.size();i++)
    {
        if ( essaie.at(i) < essaie.at(i-1) )
        {
            j = 0;
            while ( essaie.at(j) < essaie.at(i) )
            {
                j++;
            }
            c = essaie.at(i);
            for( k = i-1 ; k >= j ; k-- )
            {
                essaie[k+1] = essaie[k];
            }
            essaie[j] = c;
        }
    }

    segmentation->clear();
    for (int i = 0; i < essaie.size(); i++)
        segmentation->append(essaie.at(i));

    for(int i = 0; i < (segmentation->size()-1); i++)
    {
        if (segmentation->at(i) == segmentation->at(i+1))
        {
            segmentation->remove(i);
            i--;
        }
    }
}

```

Annexe 9 :

```

/*****
 * Name : plot
 *
 * Input variable: QVector<double> colonne0
 *                 QVector<double> colonne2
 *                 QString file_name
 *
 * Output variable: -
 *
 * Summary of function: This function creates an array that displays as
 *                       many graphs as we select files.
 *                       After being created, this function displays the graphs
 *                       according to the values read in the files (here we just
 *                       take the index and the caches put)
 *****/
void MainWindow::plot(QVector<double> colonne0, QVector<double> colonne2, QString file_name)
{
    ui->tableWidget->insertRow(0);
    ui->tableWidget->setColumnCount(1);
    ui->tableWidget->setShowGrid(true);
    ui->tableWidget->setSelectionMode(QAbstractItemView::SingleSelection);
    ui->tableWidget->setHorizontalHeaderLabels(QStringList() << "plot" );
    ui->tableWidget->horizontalHeader()->setStretchLastSection(true);
    ui->tableWidget->setRowHeight(0, 220);

    QCustomPlot *customPlot1 = new QCustomPlot(QCUSTOMPLOT_H);

    customPlot1->clearGraphs();

    // create graph and assign data to it:
    customPlot1->addGraph();

    customPlot1->graph(0)->setData(colonne0, colonne2);

    // let the ranges scale themselves so graph 0 fits perfectly in the visible area:
    customPlot1->graph(0)->rescaleAxes();

    // give the axes some labels
    customPlot1->xAxis->setLabel("Time");
    customPlot1->yAxis->setLabel("Cache misses");

    //legend
    QFont legendFont = font(); // start out with MainWindow's font..
    customPlot1->legend->setVisible(true);
    customPlot1->legend->setFont(legendFont);

    customPlot1->graph(0)->setLineStyle(QCPGraph::lsNone);
    customPlot1->graph(0)->setScatterStyle(QCPScatterStyle(QCPScatterStyle::ssPlus, 3));
    // by default, the legend is in the inset layout of the main axis rect. So this is how we access it to change legend placement:
    customPlot1->axisRect()->insetLayout()->setInsetAlignment(0, Qt::AlignBottom|Qt::AlignRight);

    // add two new graphs and set their look:
    customPlot1->graph(0)->setPen(QPen(Qt::red)); // line color blue for first graph
    customPlot1->graph(0)->setName("Caches misses " + file_name.section("/",-1,-1));

    // Allow user to drag axis ranges with mouse, zoom with mouse wheel and select graphs by clicking:
    customPlot1->setInteractions(QCP::iRangeDrag | QCP::iRangeZoom | QCP::iSelectPlottables);
    customPlot1->replot();

    ui->tableWidget->setCellWidget(0, 0, customPlot1);
}

```

Annexe 10 :

```

Name : create_first_row
Input variable: int column
                double value
Output variable: -
Summary of function: This function writes the double value passed
                     to it according to the column given on the first line
void MainWindow::create_first_row(int column, double value)
{
    QLabel *in_case = new QLabel();
    in_case->setText(QString::number(value));
    ui->partition->setCellWidget(0, column, in_case);
}

```

Annexe 11 :

```

Name : create_first_column
Input variable: int row
                QString name
Output variable: -
Summary of function: This function writes the string that is passed
                     to it according to the given row in the first column
void MainWindow::create_first_column(int row, QString name)
{
    QLabel *in_case = new QLabel();
    in_case->setText(name);
    ui->partition->setCellWidget(row, 0, in_case);
}

```


Annexe 12 :

```

Name : put_value_table_partition
Input variable: double value
                int row
                int column
Output variable: -
Summary of function: This function writes the double given to it
                    according to the given column and row also
void MainWindow::put_value_table_partition(double value, int row, int column)
{
    QLabel *in_case = new QLabel();
    in_case->setText(QString::number(value));
    ui->partition->setCellWidget(row, column, in_case);
}

```

Annexe 13 :

```

Name : create_base_table_partition
Input variable: -
Output variable: -
Summary of function: This function creates the basis
                    of the table for the partitions
void MainWindow::create_base_table_partition()
{
    ui->partition->setRowCount(segm.size()+1);
    ui->partition->setColumnCount(segmentation.size()+1);
    ui->partition->setShowGrid(true);
    ui->partition->horizontalHeader()->setVisible(false);
    ui->partition->verticalHeader()->setVisible(false);
    ui->partition->setColumnWidth(0, 200);
}

```

Annexe 14 :

```

/*****
 * Name : create_segmentation_and_avg_total
 *
 * Input variable: -
 *
 * Output variable: -
 *
 * Summary of function: This function creates a new vector of all the combined
 *                      segmentations of the different selected text files and
 *                      sorts them.
 *                      In addition, it creates the new average vectors according
 *                      to the segmentations of each one which it puts in a single
 *                      vector
 *****/
void MainWindow::create_segmentation_and_avg_total()
{
    for(int i = 0; i < segm.size(); i++)
    {
        for (int j = 0; j < segm[i].size(); j++)
        {
            segmentation.append(segm[i].at(j));
        }
    }

    for (int i = 0; i < segm.size(); i++)
    {
        avg.append(avg_by_seg(&segm[i], &colonne2[i]));
    }

    tri_and_double(&segmentation);
}

```

Annexe 15 :

```

/*****
 * Name : create_partition
 *
 * Input variable: -
 *
 * Output variable: -
 *
 * Summary of function: This function creates the partitions according to each
 *                      segmentations and each average corresponding to these
 *                      segmentations.
 *                      Then, it adds these partitions into different vectors,
 *                      which are added into a vector of vectors.
 *****/
void MainWindow::create_partition()
{
    QVector<int> variable;
    QVector<int> part_tb;
    double sum;
    QVector<double> init;
    int l = 0;

    for (int i = 0; i < segm.size(); i++)
    {
        variable.append(0);
    }

    for (int i = 0; i < segmentation.size(); i++)
    {
        sum = 0;
        for (int j = 0; j < segm.size(); j++)
        {
            if(variable.at(j) == -1)
            {}
            else
                sum += avg[j].at(variable.at(j));
        }

        for(int k = 0; k < segm.size(); k++)
        {
            if (variable.at(k) == -1)
                part_tb.append(0);
            else if (avg[k].at(variable.at(k)) == 0)
                part_tb.append(0);
            else
                part_tb.append(DoubleToInt(16*(avg[k].at(variable.at(k))/sum)));
        }

        for(int row = 0; row < segm.size(); row++)
        {
            for(int column = 0; column < segm[row].size(); column++)
            {
                if (segm[row][column] == segmentation[i])
                {
                    if (variable[row] == (avg[row].size()-1))
                        variable[row] = -1;
                    else
                        variable[row] += 1;
                }
            }
        }

        init.append(0);
        for (int i = 0; i < segm.size(); i++)
        {
            part.append(init);
        }

        for (int i = 0; i < (part_tb.size()/segm.size()); i++)
        {
            for(int j = 0; j < segm.size(); j++)
            {
                part[j].append(part_tb.at(i+j+l));
            }
            l += (segm.size()-1);
        }

        for (int i = 0; i < part.size(); i++)
        {
            part[i].remove(0);
        }
    }
}

```

Annexe 16 :

```

/*****
 * Name : create_table_partition
 *
 * Input variable: -
 *
 * Output variable: -
 *
 * Summary of function: This function creates the partition table by entering
 *                      the file names in the first column, then the values of
 *                      each partition according to the files.
 *****/
void MainWindow::create_table_partition()
{
    create_base_table_partition();

    QLabel *title = new QLabel();
    title->setText("Segmentation");
    ui->partition->setCellWidget(0, 0, title);

    for(int i = 1; i < (file_path.size()+1); i++)
    {
        create_first_column(i, file_path[i-1].section("/",-1,-1));
    }

    for(int column = 1; column < (segmentation.size()+1); column++)
    {
        create_first_row(column, segmentation.at(column-1));
    }

    for(int row = 1; row < (part.size()+1); row++)
    {
        for(int column = 1; column < (part[row-1].size()+1); column++)
        {
            put_value_table_partition(part[row-1].at(column-1), row, column);
        }
    }
}

```

Annexe 17 :

```

/*****
 * Name : on_btn_select_file_clicked
 *
 * Input variable: A button called "OPEN" is connected
 *                to this slot via a signal*
 *
 * Output variable: -
 *
 * Summary of function: This button allows you to
 *                      retrieve the paths to the various files.
 *****/
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));
    }
}

```

Annexe 18 :

```

/*****
 * Name : on_pushButton_clicked
 *
 * Input variable: A button called "CALCULATE" is connected
 *                 to this slot via a signal
 *
 * Output variable: -
 *
 * Summary of function: This button starts the calculation
 *                      of the partitions of the selected
 *                      files and their displays.
 *****/
void MainWindow::on_pushButton_clicked()
{
    create_vector(&file_path, &colonne0, &colonne1, &colonne2);
    find_segmentation(&colonne2, &segm);
    correction_segm();
    create_segmentation_and_avg_total();

    create_partition();

    create_table_partition();

    for (int i = 0; i < segm.size(); i++)
    {
        plot(colonne0[i], colonne2[i], file_path[i]);
    }
}
```

Annexe 19 :

```

/*****
 * Name : on_clear_clicked
 *
 * Input variable: A button called "CLEAR" is connected
 *                  to this slot via a signal
 *
 * Output variable: -
 *
 * Summary of function: This button clears all global variables
 *                      so that the entire program can be
 *                      restarted without restarting the software.
 *****/
void MainWindow::on_clear_clicked()
{
    colonne0.clear();
    colonne1.clear();
    colonne2.clear();
    segm.clear();
    avg.clear();
    part.clear();
    file_path.clear();
    numberCore = 0;
    numberFile = 0;
    ui->tableWidget->clear();
    ui->lineEdit->setText(QString::number(numberFile));
}

```