2020-2021

TONNELLE Nathan

L3 groupe s5o7a

Etude de cas

INFO0504

Table des matières

[Corrections 2](#_Toc55758880)

[Exemple 2](#_Toc55758881)

[Corrections effectuées 2](#_Toc55758882)

[Profilage 4](#_Toc55758883)

[Annexe 6](#_Toc55758884)

[gprof\_ker.txt (début) 6](#_Toc55758885)

[profilage\_ker.png 7](#_Toc55758886)

# Corrections

## Exemple

"fichier" :

ligne(s)

ligne(s) originel(s)

=> ligne(s) modifiée(s)

## Corrections effectuées

"isodistrib.cpp" :

l.1531

t\_exp\_spec[i+1].Im=0.0;

=> t\_exp\_spec[i].Im=0.0;

"digest.cpp" :

l.126 - l.127

Peptides = (peptide\_t \*) malloc (32 \* sizeof (char));

//Peptides = (peptide\_t \*) malloc (NbPeptides \* sizeof (peptide\_t));

=> //Peptides = (peptide\_t \*) malloc (32 \* sizeof (char));

Peptides = (peptide\_t \*) malloc (NbPeptides \* sizeof (peptide\_t));

"ascq\_me\_configuration.cpp" :

l.377

//free(unlimited\_buffer);

=> free(unlimited\_buffer);

"formula.cpp" :

l.559

free(copy);

=> //free(copy);

"seqio.cpp" :

l.273 -> l.276

/\*if(SeqBuffer!=NULL)

{

free(SeqBuffer);

}\*/

=> if(SeqBuffer!=NULL)

{

free(SeqBuffer);

}

"digest.cpp" :

l.573

=> peptide\_t\* Peptides;

"digest.cpp" :

l.430

if((first->weight==0)&&(second->weight==0))

=> if((first->weight==0) || (second->weight==0))

# Profilage

Nous pouvons voir dans le fichier **"gprof\_ker.txt"** ou le fichier graphique **"profilage\_ker.png"**, que les fonctions très couteuses (>10%) sont :

- integral\_normalization (double, int, complex\*) **40.07 %**

- optimized\_isotopic\_distribution (formula\*, int) **18.80 %**

- add\_element (formula\*, composition\*) **12.41 %**

Ensuite vient les partie faiblement coûteuses (environ, ou supérieur à 5%) :

- complex\_multiplication (complex, complex) **7.45 %**

- compute\_equivalent\_peptides () **4.97 %**

Dans ces différentes fonctions coûteuses, le nombre d'appel diffère, voici un tri du plus grand au plus petit :

- add\_element (formula\*, composition\*) **81270441**

- complex\_multiplication (complex, complex) **50929664**

- integral\_normalization (double, int, complex\*) **4401**

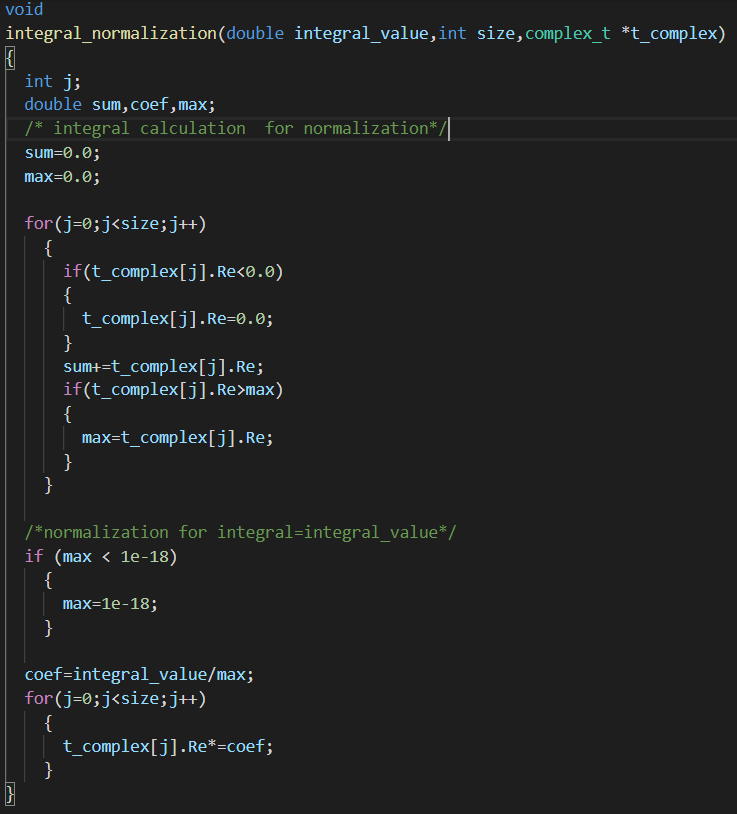
- optimized\_isotopic\_distribution (formula\*, int) **50**

- compute\_equivalent\_peptides () **50**

Nous pouvons donc constater que ce n'est pas à cause de leurs nombres d'appel que les fonctions ont un temps plus long, mais bien à cause de ce qu'elles font.

Par exemple, la fonction optimized\_isotopic\_distribution (formula\*, int) est appelée **50 fois**, mais coûte **18.80%** du temps total du programme.

Si nous allons voir la fonction qui prend le plus de temps dans le programme, dans **"isodistrib.cpp"**, integral\_normalization (double, int, complex\*), appelée **4401** fois et **40.07%** du temps d'exécution, cette fonction est composée de 29 lignes de code, ce qui ne veut rien en dire. Mais dans celle-ci, nous pouvons voir qu'il y a 2 boucles for, avec dans le premier 2 tests et dans la seconde, une multiplication de variables.

Nous pouvons donc en dire que cette fonction prend du temps car elle parcourt 2 fois la totalité des variables contenues dans \*t\_complex, de plus dans la première boucle parcours le tableau t\_complex et y réalise 2 tests sur chacune des cases, ce qui alourdi la fonction.

Cependant, nous pouvons améliorer cette fonction :

* Nous pouvons, en testant si t\_complex[0].Re est supérieur à 0, initialiser max à t\_complex[0].Re, sinon max prendra la valeur 0.0

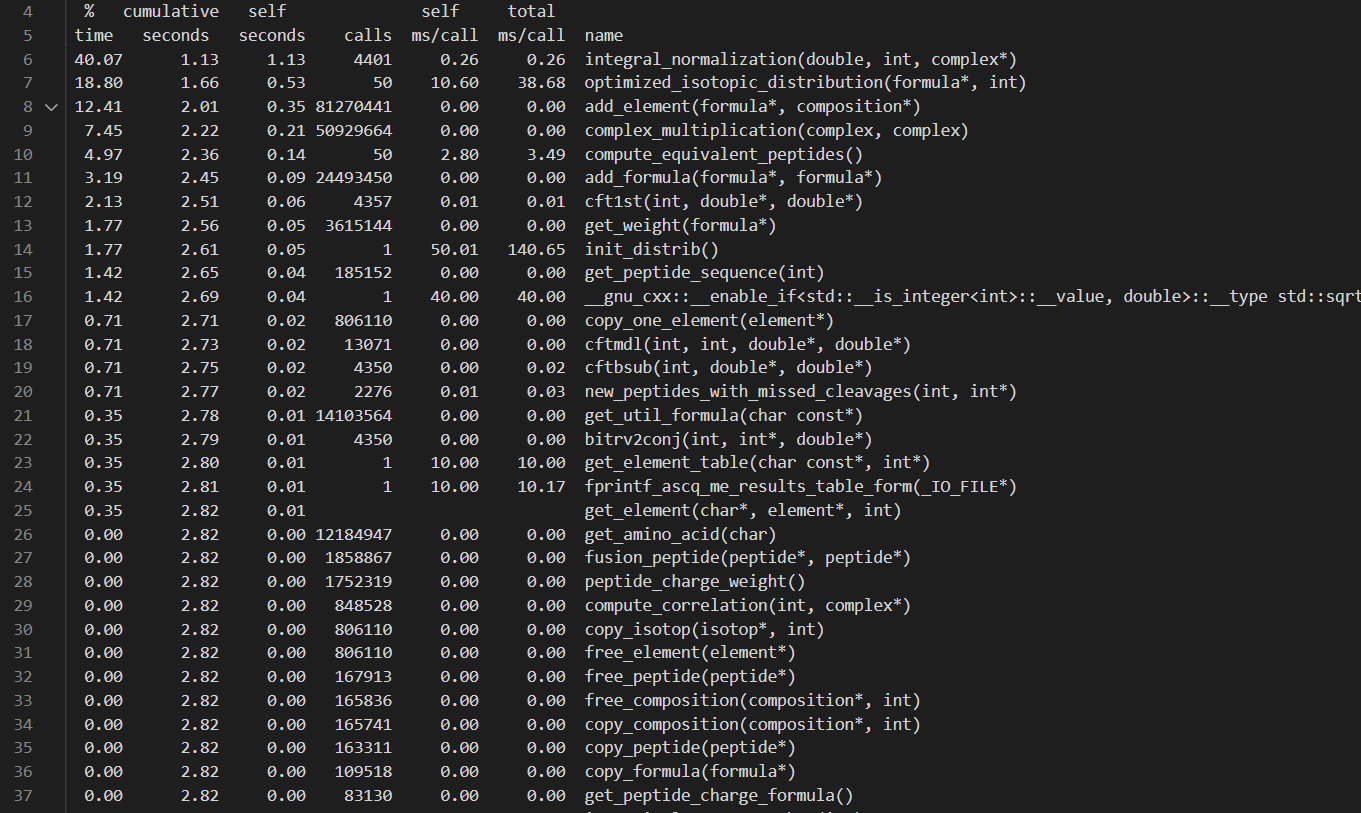
Ainsi nous faisons 1 test au lieu de 1 tour de boucle et 2 tests.

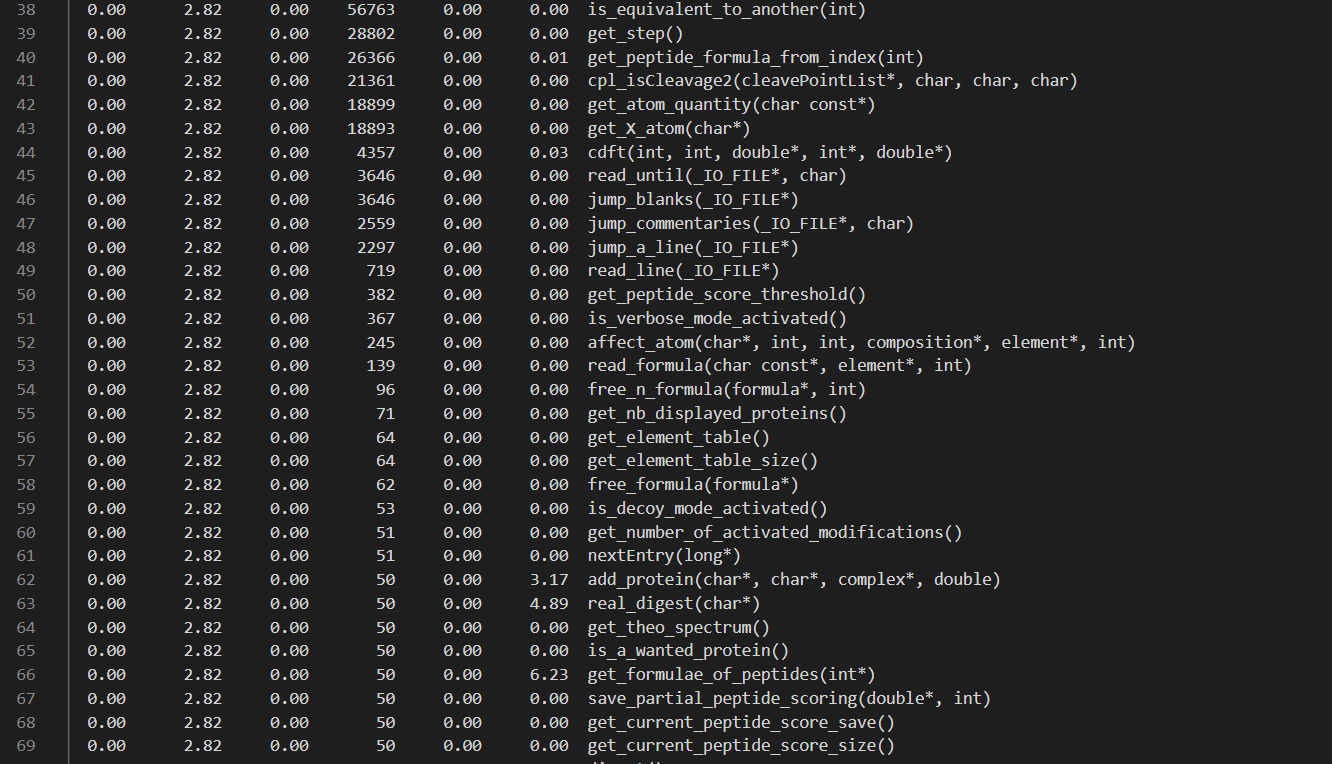
* Nous pouvons également concaténer les deux tests en faisant un if(){}else if(){}, car si le maximum est, dans le pire et le premier des cas égale à 0.0, alors toutes les cases du tableau t\_complex ne peuvent être supérieures à max que si elles sont supérieures ou égales à 0.0. Et donc ne passeraient pas par l'exécution du if, mais par l'exécution du else if, si la case est plus grande que le max.

Ainsi, avec le code transformé, cela donne :

# Annexe

## gprof\_ker.txt (début)





## profilage\_ker.png