2. Profilage

Analyse de profilage de l'exécution du ASCQ_ME sur les fichiers de configuration

Étapes pour obtenir une analyse de profilage sur le fichier de configuration default_configuration.config :

1. Nous avons modifié le fichier Makefile existant de l'application pour avoir l'option d'exécution -pg. Dans le Makefile, nous avons supprimé les commentaires, en laissant la commande :

```
#CFLAGS += -pg
#CFLAGS += -O3
```

Comme résultat, on a :

```
#Makefile by J.C. BOISSON (2005)
CC = g++
CFLAGS = -W -Wall -Werror -ansi -pedantic
# Je sais, c'est n'importe quoi ...
CFLAGS = -Wno-error-use-after-free
CFLAGS += -g
CFLAGS += -O3
CFLAGS += -pg
SRCDIR = ./src/
```

Après, nous avons relancé le makefile en exécutant la commande <<make>> dans le terminal pour faire la compilation.

1. Nous avons appelé l'exécutable avec le fichier de configuration choisit , avec la commande. :

./ascq_me _configuration/default_configuration.config

1. Comme nous avons maintenat un fichier gmon.out, nous avons fait appel à gprof pour sortir une analyse complète de la trace d'exécution, avec la commande:

gprof ./ascq_me gmon.out > rapport_profilage.txt

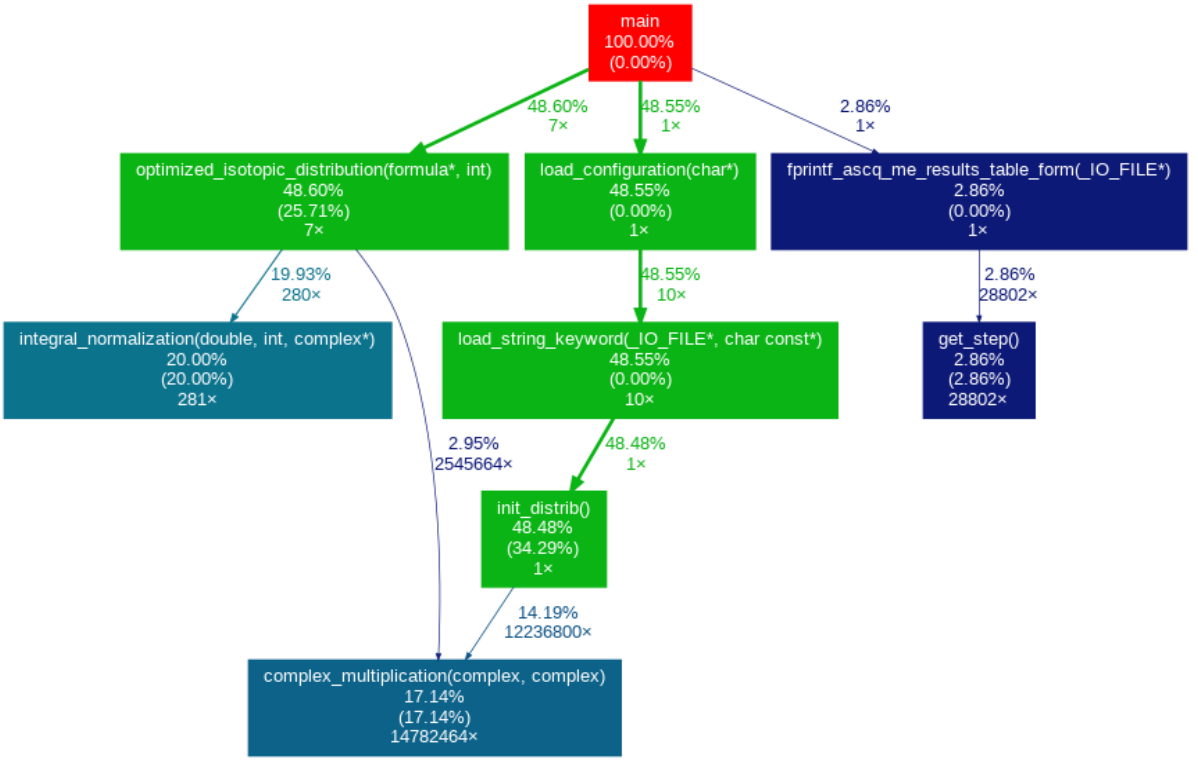
1. Pour pouvoir générer une vue graphique de la sortie on va utiliser l'outil gprof2dot en tapant la commande suivante :

gprof2dot rapport_profilage.txt > rapport_profilage.dot

1. Pour transfomer le fichier en fichier image (.PNG), nous avons utilisé l'outil dot, avec la commande :

dot -Tpng rapport_profilage.dot > rapport_profillage.png

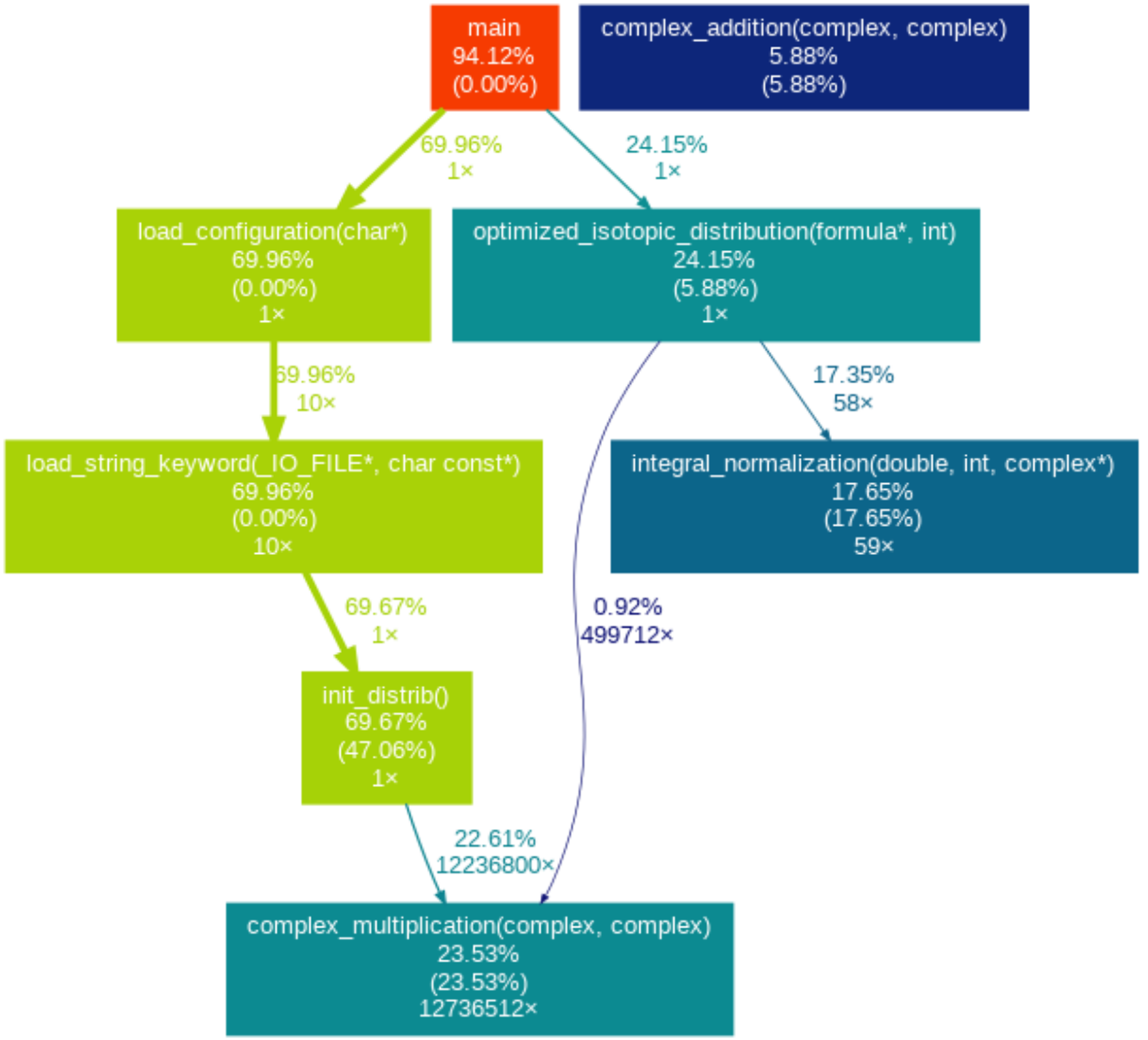
1. Analyser le résultat de profilage rendu sous forme graphique lors de l'image .png :



En analysant le résultat de profilage effectué, on peut remarquer que l'exécution de ce fichier en therme de temps CPU est divisé par 2 grandes parties de codes qui vont utiliser le plus de temps CPU : 48,60% pour optimized_isotopic_distribution(formula *, int) et 48,55% pour load_configuration (char *). On a également une partie de code moins couteuse en therme de temps mémoire, 2,86% pour fprint_ascq_me_results_table_form(_IO_FILE*).
La fonction **init_distrib()**, appelée une fois, est la plus consommatrice en temps, avec 48,48% , qui représente 34,29% du temps total d'exécution.
La fonction **optimized_isotopic_distribution(formula *, int)** est la deuxième plus consommatrice en temps, représentant 25,71% du temps total d'exécution, elle est appelée 7 fois.
La fonction appelée plusieurs, **get_step()**, avec un nombre de 28802 appels et qui prend 2,86% du temp total d'exécution a un temps propre de 0,01 secondes.

La fonction **complex_multiplication(complex, complex)** , a été appelée un très grand nombre de fois, 14782464 et elle a un temps propre très faible de 0,06 sec et représente 17,14% du temps total CPU.
Cette analyse suggère que dans le code les plus couteuse partie en therme mémoire se trouvent dans les fonctions principales, en particulier `init_distrib()`, `optimized_isotopic_distribution()` et `load_configuration(char *)`, `load_string_keyword(_IO_FILE *, char const *)`.
Pour optimiser le programme, il faut optimiser ces fonctions et cela peut conduire à des améliorations significatives des performances globales de l'application.

Étapes pour obtenir une analyse de profilage sur le fichier de configuration one_miss_cleavage_apo_mascot.config :
Nous allons utiliser les memes commandes vues précédemment, en changent le nom de fichier et on va obtenir le résultat graphique suivant :



En analysant le résultat de profilage effectué, on peut remarquer que l'exécution de ce fichier en therme de temps CPU est divisé par une grande partie de code qui va utiliser le plus de temps CPU - la fonction load_configuration(char *) avec 69,96% et deux autres parties de codes moins couteuse en therme de temps mémoire : 24,15% pour optimized_isotopic_distribution(formula *, int) et 5,88% pour la fonction complex_addition(complex, complex).
La fonction **init_distrib()** est la plus consommatrice en temps, représentant 47,06% du temps total d'exécution, elle a été appelée une seule fois et elle a un temps propre de 0,08 sec.
La fonction **complex_multiplication(complex, complex)** est la deuxième plus consommatrice en temps, avec 23,53% du temps total CPU, elle a été appelée 12 736 512 fois et elle a un temps propre de 0,04 sec.
La fonction **integral_normalization(double, int, complex *)** prend 17,65% du temps total, avec un temps propre de 0,03 sec et étant appelée 59 fois.
La fonction **optimized_isotopic_distribution(formula *, int)** prend 5,88% du temps total d'exécution et elle est appelée une fois, ayant un temps propre de 0,01

sec.
Le reste des fonctions a des pourcentages plus faibles du temps total d'exécution, avec un temps propre de 0,00 seconde, ce qui indique qu'elles contribuent moins de manière significative au temps d'exécution global du programme.
Cette analyse montre que la majeure partie du temps d'exécution est consacrée à quatre fonctions principales : `init_distrib()`, `complex_multiplication()`, `integral_normalization()` et `optimized_isotopic_distribution()`.
Pour optimiser le programme, il faut améliorer les performances globales de ces fonctions pour avoir des meilleurs performances de l'application.