

A thick dark blue vertical bar runs along the left edge of the page. A blue arrow-shaped banner points to the right from this bar, containing the text '2020-2021'. In the bottom-left corner, several thin, curved lines in dark blue and light grey sweep upwards and to the right.

2020-2021

# Etude de cas

INFO0504

TONNELLE Nathan  
L3 GROUPE S507A

## Table des matières

Corrections.....	2
Exemple.....	2
Corrections effectuées.....	2
Profilage .....	4
Annexe .....	6
gprof_ker.txt (début) .....	6
profilage_ker.png.....	7

## Corrections

### Exemple

"fichier" :

```
ligne(s)
    ligne(s) originel(s)

=> ligne(s) modifiée(s)
```

### Corrections effectuées

"isodistrib.cpp" :

```
l.1531
    t_exp_spec[i+1].lm=0.0;

=> t_exp_spec[i].lm=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 coûteuses (>10%) sont :

- <b>integral_normalization</b> (double, int, complex*)	<b>40.07</b>	<b>%</b>
- <b>optimized_isotopic_distribution</b> (formula*, int)	<b>18.80</b>	<b>%</b>
- <b>add_element</b> (formula*, composition*)	<b>12.41</b>	<b>%</b>

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

- <b>complex_multiplication</b> (complex, complex)	<b>7.45</b>	<b>%</b>
- <b>compute_equivalent_peptides</b> ()	<b>4.97</b>	<b>%</b>

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

- <b>add_element</b> (formula*, composition*)	<b>81270441</b>
- <b>complex_multiplication</b> (complex, complex)	<b>50929664</b>
- <b>integral_normalization</b> (double, int, complex*)	<b>4401</b>
- <b>optimized_isotopic_distribution</b> (formula*, int)	<b>50</b>
- <b>compute_equivalent_peptides</b> ()	<b>50</b>

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 parcourt 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 passeront pas par l'exécution du if, mais par l'exécution du else if, si la case est plus grande que le max.

```
void
integral_normalization(double integral_value,int size,complex_t *t_complex)
{
    int j;
    double sum,coef,max;
    /* integral calculation for normalization*/
    sum=0.0;
    max=0.0;

    for(j=0;j<size;j++)
    {
        if(t_complex[j].Re<0.0)
        {
            t_complex[j].Re=0.0;
        }
        sum+=t_complex[j].Re;
        if(t_complex[j].Re>max)
        {
            max=t_complex[j].Re;
        }
    }

    /*normalization for integral=integral_value*/
    if (max < 1e-18)
    {
        max=1e-18;
    }

    coef=integral_value/max;
    for(j=0;j<size;j++)
    {
        t_complex[j].Re*=coef;
    }
}
```

Ainsi, avec le code transformé, cela donne :

```
void
integral_normalization(double integral_value,int size,complex_t *t_complex)
{
    int j;
    double sum,coef,max;
    /* integral calculation for normalization*/
    sum=0.0;
    max=0.0;

    for(j=1;j<size;j++)                //for(j=0;j<size;j++)
    {
        if(t_complex[j].Re<0.0)        //if(t_complex[j].Re<0.0)
        {
            t_complex[j].Re=0.0;        //t_complex[j].Re=0.0;
        }
        else if (t_complex[j].Re>max)    //sum+=t_complex[j].Re;
        {                                //if(t_complex[j].Re>max)
            max=t_complex[j].Re;        //{
            //max=t_complex[j].Re;
            sum+=t_complex[j].Re;        //}
        }
    }

    /*normalization for integral=integral_value*/
    if (max < 1e-18)
    {
        max=1e-18;
    }

    coef=integral_value/max;
    for(j=0;j<size;j++)
    {
        t_complex[j].Re*=coef;
    }
}
```

## Annexe

## gprof\_ker.txt (début)

4	%	cumulative	self	self	total	
5	time	seconds	seconds	calls	ms/call	ms/call name
6	40.07	1.13	1.13	4401	0.26	0.26 integral_normalization(double, int, complex*)
7	18.80	1.66	0.53	50	10.60	38.68 optimized_isotopic_distribution(formula*, int)
8	12.41	2.01	0.35	81270441	0.00	0.00 add_element(formula*, composition*)
9	7.45	2.22	0.21	50929664	0.00	0.00 complex_multiplication(complex, complex)
10	4.97	2.36	0.14	50	2.80	3.49 compute_equivalent_peptides()
11	3.19	2.45	0.09	24493450	0.00	0.00 add_formula(formula*, formula*)
12	2.13	2.51	0.06	4357	0.01	0.01 cft1st(int, double*, double*)
13	1.77	2.56	0.05	3615144	0.00	0.00 get_weight(formula*)
14	1.77	2.61	0.05	1	50.01	140.65 init_distrib()
15	1.42	2.65	0.04	185152	0.00	0.00 get_peptide_sequence(int)
16	1.42	2.69	0.04	1	40.00	40.00 __gnu_cxx::__enable_if<std::__is_integer<int>::__value, double>::__type std::sqrt
17	0.71	2.71	0.02	806110	0.00	0.00 copy_one_element(element*)
18	0.71	2.73	0.02	13071	0.00	0.00 cftmdl(int, int, double*, double*)
19	0.71	2.75	0.02	4350	0.00	0.02 cftbsub(int, double*, double*)
20	0.71	2.77	0.02	2276	0.01	0.03 new_peptides_with_missed_cleavages(int, int*)
21	0.35	2.78	0.01	14103564	0.00	0.00 get_util_formula(char const*)
22	0.35	2.79	0.01	4350	0.00	0.00 bitrv2conj(int, int*, double*)
23	0.35	2.80	0.01	1	10.00	10.00 get_element_table(char const*, int*)
24	0.35	2.81	0.01	1	10.00	10.17 fprintf_ascq_me_results_table_form(_IO_FILE*)
25	0.35	2.82	0.01			get_element(char*, element*, int)
26	0.00	2.82	0.00	12184947	0.00	0.00 get_amino_acid(char)
27	0.00	2.82	0.00	1858867	0.00	0.00 fusion_peptide(peptide*, peptide*)
28	0.00	2.82	0.00	1752319	0.00	0.00 peptide_charge_weight()
29	0.00	2.82	0.00	848528	0.00	0.00 compute_correlation(int, complex*)
30	0.00	2.82	0.00	806110	0.00	0.00 copy_isotop(isotop*, int)
31	0.00	2.82	0.00	806110	0.00	0.00 free_element(element*)
32	0.00	2.82	0.00	167913	0.00	0.00 free_peptide(peptide*)
33	0.00	2.82	0.00	165836	0.00	0.00 free_composition(composition*, int)
34	0.00	2.82	0.00	165741	0.00	0.00 copy_composition(composition*, int)
35	0.00	2.82	0.00	163311	0.00	0.00 copy_peptide(peptide*)
36	0.00	2.82	0.00	109518	0.00	0.00 copy_formula(formula*)
37	0.00	2.82	0.00	83130	0.00	0.00 get_peptide_charge_formula()
38	0.00	2.82	0.00	56763	0.00	0.00 is_equivalent_to_another(int)
39	0.00	2.82	0.00	28802	0.00	0.00 get_step()
40	0.00	2.82	0.00	26366	0.00	0.01 get_peptide_formula_from_index(int)
41	0.00	2.82	0.00	21361	0.00	0.00 cpl_isCleaveage2(cleavePointList*, char, char, char)
42	0.00	2.82	0.00	18899	0.00	0.00 get_atom_quantity(char const*)
43	0.00	2.82	0.00	18893	0.00	0.00 get_X_atom(char*)
44	0.00	2.82	0.00	4357	0.00	0.03 cdft(int, int, double*, int*, double*)
45	0.00	2.82	0.00	3646	0.00	0.00 read_until(_IO_FILE*, char)
46	0.00	2.82	0.00	3646	0.00	0.00 jump_blanks(_IO_FILE*)
47	0.00	2.82	0.00	2559	0.00	0.00 jump_commentaries(_IO_FILE*, char)
48	0.00	2.82	0.00	2297	0.00	0.00 jump_a_line(_IO_FILE*)
49	0.00	2.82	0.00	719	0.00	0.00 read_line(_IO_FILE*)
50	0.00	2.82	0.00	382	0.00	0.00 get_peptide_score_threshold()
51	0.00	2.82	0.00	367	0.00	0.00 is_verbose_mode_activated()
52	0.00	2.82	0.00	245	0.00	0.00 affect_atom(char*, int, int, composition*, element*, int)
53	0.00	2.82	0.00	139	0.00	0.00 read_formula(char const*, element*, int)
54	0.00	2.82	0.00	96	0.00	0.00 free_n_formula(formula*, int)
55	0.00	2.82	0.00	71	0.00	0.00 get_nb_displayed_proteins()
56	0.00	2.82	0.00	64	0.00	0.00 get_element_table()
57	0.00	2.82	0.00	64	0.00	0.00 get_element_table_size()
58	0.00	2.82	0.00	62	0.00	0.00 free_formula(formula*)
59	0.00	2.82	0.00	53	0.00	0.00 is_decoy_mode_activated()
60	0.00	2.82	0.00	51	0.00	0.00 get_number_of_activated_modifications()
61	0.00	2.82	0.00	51	0.00	0.00 nextEntry(long*)
62	0.00	2.82	0.00	50	0.00	3.17 add_protein(char*, char*, complex*, double)
63	0.00	2.82	0.00	50	0.00	4.89 real_digest(char*)
64	0.00	2.82	0.00	50	0.00	0.00 get_theo_spectrum()
65	0.00	2.82	0.00	50	0.00	0.00 is_a_wanted_protein()
66	0.00	2.82	0.00	50	0.00	6.23 get_formulae_of_peptides(int*)
67	0.00	2.82	0.00	50	0.00	0.00 save_partial_peptide_scoring(double*, int)
68	0.00	2.82	0.00	50	0.00	0.00 get_current_peptide_score_save()
69	0.00	2.82	0.00	50	0.00	0.00 get_current_peptide_score_size()

