Projecte Optimització FTDock

Albert Segarra Roca

Programació Conscient de l'Arquitectura

FIB • UPC

27 de juny de 2016

Resum

En aquest document s'expliquen i s'analitzen les optimitzacions aplicades a un programa de docking de proteines, `ftdock`, tenint en compte l'arquitectura d'un computador `Intel Core i5-3317U`, i aplicant la metodologia explicada en l'assignatura de `Programació Conscient de l'Arquitectura` de la `Facultat d'Informàtica de Barcelona`.

1 Arquitectura del computador

L'arquitectura del computador en el que s'han realitzat i evaluat les optimitzaciones és la següent:

- **Processador**: Intel Core i5-3317U
- **Instruction Set**: 64 Bits
- **Instruction Set Extensions**: AVX
- **Freqüència Base**: 1.7 GHz
- **Freqüència Turbo**: 2.6 GHz
- **Cores**: 2
- **Threads**: 4
- **L3 Cache**: 3 MB
- **Memòria RAM**: 8 GB

2 Metodologia

La metodologia que s'ha seguit durant el desenvolupament de les optimitzacions del codi, està basada en la que s'ha explicat a les classes de teoria de l'assignatura.

Primer de tot, es realitza una avaluació de les parts del codi que influeixen més en el temps d'execució, per tal de saber quines funcions és més convenient optimitzar per tal d'obtenir un major speedup, d'acord amb la llei d'Ahmdal.

Per això s'utilitza l'script `profile`, que realitza profiling de l'última versió optimitzada del codi mitjançant `oprofile`, i que permet veure quin és el pes en cicles d'execució de cada línia del codi.

Un cop identificats els punts calents del codi, es mesuren els speedups potencials mitjançant la llei d'Ahmdal, i si es veu que l'speedup potencial és suficient com per que valgui la pena fer l'optimització, es procedeix a intentar optimitzar el codi.

Per fer-ho, primer s'avaluen possibles optimitzacions, i se n'escull la més prometedora, tenint en compte abans de comprovar mitjançant l'script `assembler` que el compilador no realitzi ja la optimització que anem a fer. En cada un d'aquests passos de la metodologia es fa una única optimització, per tal de poder avaluar-les individualment.

Un cop s'ha realitzat l'optimització, es comprova que el resultat del programa segueixi sent correcte, i, en cas afirmatiu , es calcula l'speedup aconseguit. Per tal de fer això s'utilitza l'script `cs`, que realitza les dues operacions alhora.

Si es troba que les sortides no són correctes, es procedeix a debugar el codi i arreglar-lo, fins que la sortida sigui correcta. Un cop la sortida és correcta, es calcula l'speedup respecte la última optimització realitzant

diverses execucions i fent la mitjana dels resultats. Mitjançant un test estadístic (que el script `cs` ja realitza), es determina si realment hi ha hagut un speedup positiu. En cas afirmatiu, la optimització es dona per bona i es repeteix el procés a partir del codi optimitzat, mitjançant el script `new`, que crea una nova versió del codi a partir de l'actual.

En cas negatiu, la optimització es marca com a *no optimització*, i es repeteix el procés desde la versió anterior i provant altres optimitzacions.

3 Optimitzacions

3.1 Actualització llibreria FFTW 3.3.4 (`001-library`)

En aquest cas, la optimització simplement ha consistit en actualitzar la versió de la llibreria de FFTW de 2.1.3 a 3.3.4. Per fer-ho, s'han seguit les instruccions de la web oficial de la llibreria (cal fer modificacions al codi ja que l'interfície de la llibreria no es exactament la mateixa), i s'ha baixat i instal·lat la nova llibreria.

Aquí tenim una imatge de l'execució de l'script speedup que mostra l'speedup aconseguit amb els diferents tests:

```
      Is faster? | CPU Speedup | CPU       | Elp Speedup | Elapsed   | CPU%   | Memory    | Reps
INFO  : ftdock.3 -static ../inputs/2pka.parsed -mobile ../inputs/5pti.parsed:
            N/A | N/A         | 32.696s   | N/A         | 32.707s   | 99.97% | 60.152MB  | 1
INFO  : ftdock-opt001-library.3 -static ../inputs/2pka.parsed -mobile ../inputs/5pti.parsed:
         UNKNOWN | 1.359       | 24.052s   | 1.359       | 24.06s    | 99.97% | 60.564MB  | 1
      Is faster? | CPU Speedup | CPU       | Elp Speedup | Elapsed   | CPU%   | Memory    | Reps
INFO  : ftdock.3 -static ../inputs/1hba.parsed -mobile ../inputs/5pti.parsed:
            N/A | N/A         | 74.73s    | N/A         | 74.753s   | 99.97% | 93.292MB  | 1
INFO  : ftdock-opt001-library.3 -static ../inputs/1hba.parsed -mobile ../inputs/5pti.parsed:
         UNKNOWN | 1.012       | 73.87s    | 1.012       | 73.894s   | 99.97% | 93.88MB   | 1
      Is faster? | CPU Speedup | CPU       | Elp Speedup | Elapsed   | CPU%   | Memory    | Reps
INFO  : ftdock.3 -static ../inputs/4hhb.parsed -mobile ../inputs/5pti.parsed:
            N/A | N/A         | 102.674s  | N/A         | 102.706s  | 99.97% | 96.712MB  | 1
INFO  : ftdock-opt001-library.3 -static ../inputs/4hhb.parsed -mobile ../inputs/5pti.parsed:
         UNKNOWN | 1.009       | 101.729s  | 1.009       | 101.761s  | 99.97% | 97.212MB  | 1
      Is faster? | CPU Speedup | CPU       | Elp Speedup | Elapsed   | CPU%   | Memory    | Reps
INFO  : ftdock.3 -static ../inputs/1ACB_rec.parsed -mobile ../inputs/1ACB_lig.parsed:
            N/A | N/A         | 42.667s   | N/A         | 42.681s   | 99.97% | 86.32MB   | 1
INFO  : ftdock-opt001-library.3 -static ../inputs/1ACB_rec.parsed -mobile ../inputs/1ACB_lig.parsed:
         UNKNOWN | 1.359       | 31.391s   | 1.359       | 31.401s   | 99.97% | 86.78MB   | 1
```

Figura 1: Speedups de l'optimització 001 calculats amb l'script `speedup`

Tal com podem veure, tant en el primer test com en el 4t test és perceb un speedup considerable de **1.359**. Pels altres dos, en canvi, no es perceb cap speedup. Això segurament és degut a que per aquests testos la influència del càlcul que realitza la llibreria no és tant gran com en els altres dos, per això el canvi no influeix gairebé en el temps d'execució.

3.2 Inlining pythagoras (002-pythagoras-inline)

Abans de realitzar la primera optimització, evaluem les funcions que tenen un cost més important en l'execució del programa mitjançant `gprof`:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  us/call  us/call  name
 47.36     16.23    16.23                             electric_field
 25.59     25.00     8.77                             apply
 14.42     29.94     4.94 1303909272     0.00     0.00  pythagoras
  3.47     31.13     1.19                             writev
  1.63     31.69     0.56                             __profile_frequency
  1.52     32.21     0.52                             apply_r2hc
  1.49     32.72     0.51                             apply_hc2r
  0.93     33.04     0.32                             t1_2
  0.61     33.25     0.21    23184     9.06     9.06  gord
```

Degut al que s'ha pogut observar en el profiling anterior, les dues funcions que tenen un pes més important (i de fet, les úniques que el tenen de les que podem optimitzar) en el temps d'execució del programa són les funcions `electric_field` i `pythagoras`, per tant ens centrarem en analitzar-les més a fons mitjançant oprofile, per poder veure el pes de cada línia de codi de les funcions:

electric_field:

```
    24  0.0021 :      y_centre  = gcentre( y , grid_span , grid_size ) ;
           :
    73  0.0065 :      for( z = 0 ; z < grid_size ; z ++ ) {
           :
   368  0.0329 :        z_centre  = gcentre( z , grid_span , grid_size ) ;
           :
    28  0.0025 :        phi = 0 ;
           :
  7509  0.6721 :        for( residue = 1 ; residue <= This_Structure.length ; residue ++ ) {
 58404  5.2273 :          for( atom = 1 ; atom <= This_Structure.Residue[residue].size ; atom ++ ) {
           :
108369  9.6993 :            if( This_Structure.Residue[residue].Atom[atom].charge != 0 ) {
           :
 38116  3.4115 :              distance = pythagoras( This_Structure.Residue[residue].Atom[atom].coord[1] ,
           :
 16453  1.4726 :              if( distance < 2.0 ) distance = 2.0 ;
           :
 16582  1.4841 :              if( distance >= 2.0 ) {
           :
 18830  1.6853 :                if( distance >= 8.0 ) {
           :
           :                  epsilon = 80 ;
           :
           :                } else {
           :
   335  0.0300 :                  if( distance <= 6.0 ) {
           :
           :                    epsilon = 4 ;
           :
           :                  } else {
           :
   304  0.0272 :                    epsilon = ( 38 * distance ) - 224 ;
           :
           :                  }
```

```
                :
                :                   }
                :
202651 18.1377  :                   phi += ( This_Structure.Residue[residue].Atom[atom].charge / ( epsilon * d
                :
                :                 }
                :
                :               }
                :
                :             }
                :           }
                :
    48  0.0043  :           grid[gaddress(x,y,z,grid_size)] = (float)phi ;
                :
                :       }
```

pythagoras

```
:float pythagoras( float x1 , float y1 , float z1 , float x2 , float y2 , float z2 ) {
                :
191735 17.1607  :  return sqrt( ( ( x1 - x2 ) * ( x1 - x2 ) ) + ( ( y1 - y2 ) * ( y1 - y2 ) ) + ( ( z1 - z2
                :
  8902  0.7967 :}
```

Tal com podem veure, les dues línies més costoses són la del cos de `pythagoras` i l'increment de `phi`. El fet que aquesta part del programa tingui un pes tant gran en el temps d'execució és que la seva complexitat és, en termes generals $O(n^5)$. Això, sumat al fet que es realitzen operacions de llarga latència (sqrt, divisió) fa que el cost es dispari en les dues línies que hem observat.

Optimitzar aquestes operacions de llarga latència, però, serà difícil, ja que treballem amb reals i la memoització no es possible i no es poden utilitzar bit hacks. A més, no hi ha cap constant en les divisions que ens permeti calcular la divisió mitjançant la multiplicació per l'invers prèviament calculat.

Una possible opció, però, per tal d'optimitzar el cost de la funció `pythagoras`, és realitzar-ne inlining. Notem que el cos de la funció no és molt gran, per tant els beneficis poden ser importants, sobretot si el compilador és capaç de realitzar més optimitzacions un cop veu el codi de `pythagoras` en perspectiva amb el de la resta de `electric_field`. A més, tal com podem veure en el següent anàlisi, el compilador no realitza cap inlining.

Anàlisi del codi assemblador:

```
4054bb: c5 f8 2e 70 24          vucomiss 0x24(%rax),%xmm6
4054c0: 7b de                   jnp     4054a0 <electric_field+0x220>
          distance = pythagoras( This_Structure.Residue[residue].Atom[atom].coord[1] , This_Structure.Resi
.
.
.
4054e8: e8 53 fb ff ff          callq  405040 <pythagoras>
```

Procedim per tant a realitzar inlining de la funció mitjançant una macro.

Els speedups obtinguts són els següents:

Tal com podem veure, els speedups no són molt elevats, però són significatius en tots els casos.

```
       Is faster? | CPU Speedup | CPU       | Elp Speedup | Elapsed   | CPU%   | Memory    | Reps
INFO  : ftdock-opt001-library.3 -static ../inputs/2pka.parsed -mobile ../inputs/5pti.parsed:
             N/A | N/A         | 24.366s   | N/A         | 24.374s   | 99.96% | 60.748MB  | 1
INFO  : ftdock-opt002-pythagoras-inline.3 -static ../inputs/2pka.parsed -mobile ../inputs/5pti.parsed:
          UNKNOWN | 1.061       | 22.96s    | 1.061       | 22.967s   | 99.97% | 60.76MB   | 1
       Is faster? | CPU Speedup | CPU       | Elp Speedup | Elapsed   | CPU%   | Memory    | Reps
INFO  : ftdock-opt001-library.3 -static ../inputs/1hba.parsed -mobile ../inputs/5pti.parsed:
             N/A | N/A         | 74.981s   | N/A         | 75.005s   | 99.97% | 93.78MB   | 1
INFO  : ftdock-opt002-pythagoras-inline.3 -static ../inputs/1hba.parsed -mobile ../inputs/5pti.parsed:
          UNKNOWN | 1.093       | 68.582s   | 1.093       | 68.604s   | 99.97% | 93.732MB  | 1
       Is faster? | CPU Speedup | CPU       | Elp Speedup | Elapsed   | CPU%   | Memory    | Reps
INFO  : ftdock-opt001-library.3 -static ../inputs/4hhb.parsed -mobile ../inputs/5pti.parsed:
             N/A | N/A         | 86.519s   | N/A         | 86.546s   | 99.97% | 97.3MB    | 1
INFO  : ftdock-opt002-pythagoras-inline.3 -static ../inputs/4hhb.parsed -mobile ../inputs/5pti.parsed:
          UNKNOWN | 1.066       | 81.155s   | 1.066       | 81.181s   | 99.97% | 97.196MB  | 1
       Is faster? | CPU Speedup | CPU       | Elp Speedup | Elapsed   | CPU%   | Memory    | Reps
INFO  : ftdock-opt001-library.3 -static ../inputs/1ACB_rec.parsed -mobile ../inputs/1ACB_lig.parsed:
             N/A | N/A         | 30.938s   | N/A         | 30.948s   | 99.97% | 86.744MB  | 1
INFO  : ftdock-opt002-pythagoras-inline.3 -static ../inputs/1ACB_rec.parsed -mobile ../inputs/1ACB_lig.parsed:
          UNKNOWN | 1.083       | 28.563s   | 1.083       | 28.572s   | 99.97% | 86.744MB  | 1
```

Figura 2: Speedups de l'optimització 002 calculats amb l'script `speedup`

3.3 Loop interchange (`003-loop-interchange`)

Després de realitzar l'anterior optimització, realitzem profiling per evaluar de nou les línies més costoses:

```
  7181   0.6231 :                 for (residue = 1; residue <= This_Structure.length; residue++) {
 94747   8.2210 :                     for (atom = 1; atom <= This_Structure.Residue[residue].size; atom++) {
107599   9.3362 :                         if (This_Structure.Residue[residue].Atom[atom].charge != 0) {
                :                             // Inlined pythagoras function with a macro to avoid call overh
230779  20.0243 :                             distance = PYTHAGORAS(This_Structure.Residue[residue].Atom[aton
                :                                                   This_Structure.Residue[residue].Atom[aton
                :                                                   This_Structure.Residue[residue].Atom[aton
                :                                                   x_centre,
                :                                                   y_centre,
                :                                                   z_centre);
                :
 12729   1.1045 :                             if (distance < 2.0) distance = 2.0;
                :
 17640   1.5306 :                             if (distance >= 2.0) {
 28902   2.5078 :                                 if (distance >= 8.0) {
                :                                     epsilon = 80;
                :                                 } else {
   322   0.0279 :                                     if (distance <= 6.0) {
                :                                         epsilon = 4;
                :                                     } else {
   303   0.0263 :                                         epsilon = (38 * distance) - 224;
                :                                     }
                :                                 }
                :
154662  13.4198 :                                 phi += (This_Structure.Residue[residue].Atom[atom].charge
                :                             }
                :                         }
                :                     }
                :                 }
```

Ens adonem que una altra font de ineficiència important, amb un pes del 9%, és la línia que comprova que la càrrega no sigui zero. Mitjançant profiling amb oprofile i mesurant l'event `BRANCH_MISP_RETIRED`, veiem que aquesta línia suposa una gran quantitat de fallades del predictor:

```
 4   2.0513 :                for (residue = 1; residue <= This_Structure.length; residue++) {
 1   0.5128 :                    for (atom = 1; atom <= This_Structure.Residue[residue].size; atom++) {
23  11.7949 :                        if (This_Structure.Residue[residue].Atom[atom].charge != 0) {
         :                            // Inlined pythagoras function with a macro to avoid call overhead
76  38.9744 :                            distance = PYTHAGORAS(This_Structure.Residue[residue].Atom[atom].c
         :                                                  This_Structure.Residue[residue].Atom[atom].c
```

Per tal d'optimitzar això, ens adonem que realment l'ordre dels bucles for anidats és intercanviable en alguns casos. Un dels possibles és moure els bucles que recorren els residus i els àtoms del residu a fora del tot, i per cada àtom fer els càlculs amb les coordenades x,y,z, que seràn els bucles interiors. D'aquesta manera, la comprovació de l'if es realitza $O(n^2)$ cops, versus els $O(n^5)$ cops d'abans, i a més en el cas que la condició sigui falsa ens estalviem de recórrer totes les possibles components x, y, z, lo cual suposa un estalvi de recursos important en certs casos.

Un cop optimitzat el codi, obtenim els següents speedups:

```
       Is faster? | CPU Speedup | CPU       | Elp Speedup | Elapsed   | CPU%   | Memory    | Reps
INFO  : ftdock-opt001-library.3 -static ../inputs/2pka.parsed -mobile ../inputs/5pti.parsed:
             N/A | N/A         | 24.366s   | N/A         | 24.374s   | 99.96% | 60.748MB  | 1
INFO  : ftdock-opt002-pythagoras-inline.3 -static ../inputs/2pka.parsed -mobile ../inputs/5pti.parsed:
          UNKNOWN | 1.061       | 22.96s    | 1.061       | 22.967s   | 99.97% | 60.76MB   | 1
       Is faster? | CPU Speedup | CPU       | Elp Speedup | Elapsed   | CPU%   | Memory    | Reps
INFO  : ftdock-opt001-library.3 -static ../inputs/1hba.parsed -mobile ../inputs/5pti.parsed:
             N/A | N/A         | 74.981s   | N/A         | 75.005s   | 99.97% | 93.78MB   | 1
INFO  : ftdock-opt002-pythagoras-inline.3 -static ../inputs/1hba.parsed -mobile ../inputs/5pti.parsed:
          UNKNOWN | 1.093       | 68.582s   | 1.093       | 68.604s   | 99.97% | 93.732MB  | 1
       Is faster? | CPU Speedup | CPU       | Elp Speedup | Elapsed   | CPU%   | Memory    | Reps
INFO  : ftdock-opt001-library.3 -static ../inputs/4hhb.parsed -mobile ../inputs/5pti.parsed:
             N/A | N/A         | 86.519s   | N/A         | 86.546s   | 99.97% | 97.3MB    | 1
INFO  : ftdock-opt002-pythagoras-inline.3 -static ../inputs/4hhb.parsed -mobile ../inputs/5pti.parsed:
          UNKNOWN | 1.066       | 81.155s   | 1.066       | 81.181s   | 99.97% | 97.196MB  | 1
       Is faster? | CPU Speedup | CPU       | Elp Speedup | Elapsed   | CPU%   | Memory    | Reps
INFO  : ftdock-opt001-library.3 -static ../inputs/1ACB_rec.parsed -mobile ../inputs/1ACB_lig.parsed:
             N/A | N/A         | 30.938s   | N/A         | 30.948s   | 99.97% | 86.744MB  | 1
INFO  : ftdock-opt002-pythagoras-inline.3 -static ../inputs/1ACB_rec.parsed -mobile ../inputs/1ACB_lig.parsed:
          UNKNOWN | 1.083       | 28.563s   | 1.083       | 28.572s   | 99.97% | 86.744MB  | 1
```

Figura 3: Speedups de l'optimització 002 calculats amb l'script `speedup`

3.4 Simplificació de codi (`004-code-simplification`)

En aquest punt, i donat el profiling anterior, ens adonem que no té massa sentit optimitzar la resta d'ifs (per exemple, possiblement amb bit hacks), ja que aquests no suposen un nombre significatiu de fallades de predictor. El que sí que podem apreciar, però, és que hi ha una condició innecessària (que el compilador no pot optimitzar, ja que donat que treballem en coma flotant en certs casos podria donar resultats diferents). El següent if:

```
if (distance >= 2.0) {
    ...
```

és totalment innecessari, donat que anteriorment ja ens assegurem que el valor de distance és més gran o igual a 2.0.

Per tant doncs, eliminem la condició.

Els speedups obtinguts són els següents:

```
       Is faster? | CPU Speedup | CPU       | Elp Speedup | Elapsed   | CPU%   | Memory    | Reps
INFO  : ftdock-opt003-loop-interchange.3 -static ../inputs/2pka.parsed -mobile ../inputs/5pti.parsed:
             N/A | N/A         | 20.615s   | N/A         | 20.622s   | 99.97% | 60.632MB  | 1
INFO  : ftdock-opt004-code-simplification.3 -static ../inputs/2pka.parsed -mobile ../inputs/5pti.parsed:
         UNKNOWN | 1.043       | 19.76s    | 1.043       | 19.766s   | 99.97% | 60.684MB  | 1
       Is faster? | CPU Speedup | CPU       | Elp Speedup | Elapsed   | CPU%   | Memory    | Reps
INFO  : ftdock-opt003-loop-interchange.3 -static ../inputs/1hba.parsed -mobile ../inputs/5pti.parsed:
             N/A | N/A         | 60.325s   | N/A         | 60.344s   | 99.97% | 93.888MB  | 1
INFO  : ftdock-opt004-code-simplification.3 -static ../inputs/1hba.parsed -mobile ../inputs/5pti.parsed:
         UNKNOWN | 1.032       | 58.439s   | 1.032       | 58.458s   | 99.97% | 93.896MB  | 1
       Is faster? | CPU Speedup | CPU       | Elp Speedup | Elapsed   | CPU%   | Memory    | Reps
INFO  : ftdock-opt003-loop-interchange.3 -static ../inputs/4hhb.parsed -mobile ../inputs/5pti.parsed:
             N/A | N/A         | 71.379s   | N/A         | 71.401s   | 99.97% | 97.3MB    | 1
INFO  : ftdock-opt004-code-simplification.3 -static ../inputs/4hhb.parsed -mobile ../inputs/5pti.parsed:
         UNKNOWN | 1.061       | 67.304s   | 1.061       | 67.325s   | 99.97% | 97.264MB  | 1
       Is faster? | CPU Speedup | CPU       | Elp Speedup | Elapsed   | CPU%   | Memory    | Reps
INFO  : ftdock-opt003-loop-interchange.3 -static ../inputs/1ACB_rec.parsed -mobile ../inputs/1ACB_lig.parsed:
             N/A | N/A         | 24.571s   | N/A         | 24.578s   | 99.97% | 86.76MB   | 1
INFO  : ftdock-opt004-code-simplification.3 -static ../inputs/1ACB_rec.parsed -mobile ../inputs/1ACB_lig.parsed:
         UNKNOWN | 1.017       | 24.158s   | 1.017       | 24.165s   | 99.97% | 86.868MB  | 1
```

Figura 4: Speedups de l'optimització 004 calculats amb l'script `speedup`

De nou, no són molt significatius, però són rellevants.

3.5 Memorització de gcentre (005-mem-gcentre)

Profiling després de l'optimització anterior:

```
     3  4.2e-04 :            for (x = 0; x < grid_size; x++) {
    19   0.0027 :                x_centre = gcentre(x, grid_span, grid_size);
    20   0.0028 :                for (y = 0; y < grid_size; y++) {
 13787   1.9243 :                    y_centre = gcentre(y, grid_span, grid_size);
 12369   1.7264 :                    for (z = 0; z < grid_size; z++) {
 29530   4.1217 :                        z_centre = gcentre(z, grid_span, grid_size);
                :
                :                        // Inlined pythagoras function with a macro to avoid call overhead
 91647  12.7917 :                        distance = PYTHAGORAS(This_Structure.Residue[residue].Atom[atom].c
                :                                              This_Structure.Residue[residue].Atom[atom].c
                :                                              This_Structure.Residue[residue].Atom[atom].c
                :                                              x_centre,
                :                                              y_centre,
                :                                              z_centre);
                :
```

```
 15534   2.1682 :                        if (distance < 2.0) distance = 2.0;
                :
 14378   2.0068 :                        if (distance >= 8.0)
                :                            epsilon = 80;
   157   0.0219 :                        else if (distance <= 6.0)
                :                            epsilon = 4;
                :                        else
   222   0.0310 :                            epsilon = (38 * distance) - 224;
                :
204218  28.5038 :                        grid[gaddress(x, y, z, grid_size)] += This_Structure.Residue[residu
```

Un cop realitzada la optimització del `loop-interchange`, i donat que les crides a les funcions `gcentre` que es fan als diferents bucles que recorren x, y, z han passat a tenir un cost significatiu pel que podem veure al profiling (i donat que ara estàn en bucles més interns), intentem optimitzar aquesta part del codi. Per fer-ho, ens adonem que en el fons per les tres components estem realitzant els mateixos càlculs de `gcentre` per un rang de valors del paràmetre `oordinate` determinat, que va de 0 a `grid_size` - 1. A més, tant `grid_span` com `grid_size` són valors que es mantenen constants durant tota la iteració.

Per aquesta raó, decidim extreure el càlcul fora dels bucles, i memoritzar-lo en un array que tindrà tamany `grid_size`, i en el que es guardaràn els resultats de `gcentre` per tots els possibles valors de `oordinate`. D'aquesta forma, podem reutilitzar el càlcul en els tres llocs que s'utilitza dels bucles interiors.

Els speedups que obtenim són els següents:

```
       Is faster? | CPU Speedup | CPU       | Elp Speedup | Elapsed   | CPU%   | Memory    | Reps
INFO  : ftdock-opt004-code-simplification.3 -static ../inputs/2pka.parsed -mobile ../inputs/5pti.parsed:
             N/A | N/A         | 21.18s    | N/A         | 21.187s   | 99.97% | 60.632MB  | 1
INFO  : ftdock-opt005-mem-gcentre.3 -static ../inputs/2pka.parsed -mobile ../inputs/5pti.parsed:
         UNKNOWN | 1.424       | 14.874s   | 1.424       | 14.879s   | 99.97% | 60.528MB  | 1
       Is faster? | CPU Speedup | CPU       | Elp Speedup | Elapsed   | CPU%   | Memory    | Reps
INFO  : ftdock-opt004-code-simplification.3 -static ../inputs/1hba.parsed -mobile ../inputs/5pti.parsed:
             N/A | N/A         | 56.023s   | N/A         | 56.04s    | 99.97% | 93.776MB  | 1
INFO  : ftdock-opt005-mem-gcentre.3 -static ../inputs/1hba.parsed -mobile ../inputs/5pti.parsed:
         UNKNOWN | 1.463       | 38.289s   | 1.463       | 38.301s   | 99.97% | 93.776MB  | 1
       Is faster? | CPU Speedup | CPU       | Elp Speedup | Elapsed   | CPU%   | Memory    | Reps
INFO  : ftdock-opt004-code-simplification.3 -static ../inputs/4hhb.parsed -mobile ../inputs/5pti.parsed:
             N/A | N/A         | 67.249s   | N/A         | 67.271s   | 99.97% | 97.4MB    | 1
INFO  : ftdock-opt005-mem-gcentre.3 -static ../inputs/4hhb.parsed -mobile ../inputs/5pti.parsed:
         UNKNOWN | 1.399       | 48.066s   | 1.399       | 48.081s   | 99.97% | 97.144MB  | 1
       Is faster? | CPU Speedup | CPU       | Elp Speedup | Elapsed   | CPU%   | Memory    | Reps
INFO  : ftdock-opt004-code-simplification.3 -static ../inputs/1ACB_rec.parsed -mobile ../inputs/1ACB_lig.parsed:
             N/A | N/A         | 24.117s   | N/A         | 24.125s   | 99.97% | 86.804MB  | 1
INFO  : ftdock-opt005-mem-gcentre.3 -static ../inputs/1ACB_rec.parsed -mobile ../inputs/1ACB_lig.parsed:
         UNKNOWN | 1.378       | 17.497s   | 1.378       | 17.503s   | 99.97% | 86.892MB  | 1
```

Figura 5: Speedups de l'optimització 005 calculats amb l'script `speedup`

Tal com podem veure, aquesta optimització ha sigut molt significativa.

3.6 Inducció amb punters (`006-pointer-induction`)

Profiling després de l'optimització anterior:

```
                :    for (x = 0; x < grid_size; ++x)
                :        mem_gcentre[x] = gcentre(x, grid_span, grid_size);
                :
                :    for (residue = 1; residue <= This_Structure.length; residue++) {
                :        printf(".");
                :        for (atom = 1; atom <= This_Structure.Residue[residue].size; atom++) {
                :            if (This_Structure.Residue[residue].Atom[atom].charge == 0) continue;
                :            for (x = 0; x < grid_size; x++) {
     1 2.6e-04  :                x_centre = mem_gcentre[x];
     4  0.0010  :                for (y = 0; y < grid_size; y++) {
    94  0.0244  :                    y_centre = mem_gcentre[y];
 13337  3.4670  :                    for (z = 0; z < grid_size; z++) {
                :                        z_centre = mem_gcentre[z];
                :
                :                        // Inlined pythagoras function with a macro to avoid call overhead
 62525 16.2534  :                        distance = PYTHAGORAS(This_Structure.Residue[residue].Atom[atom].c
                :                                              This_Structure.Residue[residue].Atom[atom].c
                :                                              This_Structure.Residue[residue].Atom[atom].c
                :                                              x_centre,
                :                                              y_centre,
                :                                              z_centre);
                :
 13647  3.5476  :                        if (distance < 2.0) distance = 2.0;
                :
  6271  1.6302  :                        if (distance >= 8.0) {
                :                            epsilon = 80;
                :                        } else {
   103  0.0268  :                            if (distance <= 6.0) {
                :                                epsilon = 4;
                :                            } else {
 13505  3.5106  :                                epsilon = (38 * distance) - 224;
                :                            }
                :                        }
                :
125641 32.6605  :                        grid[gaddress(x, y, z, grid_size)] += (This_Structure.Residue[resid
```

Mirant el codi que calcula al memorització, sen's passen pel cap possibles optimizacions com vectorització o unrolling, a més de un possible càlcul acumulat dels valors. Tot i així, mirant el profiling, ens adonem que fer aquestes optimitzacions no tindria sentit, ja que el pes del codi que calcula la memorització és negligible.

Una optimització que sí és possible, però, donat el pes de la línia de codi en la que se situa (32%), és el càlcul de `gaddress(x,y,z,grid_size)`. Si analitzem aquesta funció, ens adonem que realment es pot calcular de forma acumulada, mitjançant inducció amb punters.

El fet és que aquesta fórmula:

```
#define gaddress(x,y,z,grid_size) ( (z) + ( 2 * ( (grid_size) / 2 + 1 ) ) * ( (y) + (grid_size) * (x) ) )
```

és simplificable a:

```
#define gaddress(x,y,z,grid_size) ( (z) + ( grid_size + 2 ) * ( (y) + (grid_size) * (x) ) )
```

Donat que per les restriccions de l'entrada que s'imposen en `ftdock.c`, el valor de `grid_size` ha de ser necessàriament múltiple de 2.

Això ens calcular l'adreça incrementant en una unitat en el bucle de les z's, en 2 unitats en el bucle de les y's i en 1 unitat en el bucle de les x's.

Els speedups obtinguts són els següents:

```
INFO  : ftdock-opt005-mem-gcentre.3 -static ../inputs/2pka.parsed -mobile ../inputs/5pti.parsed:
            N/A | N/A         | 14.714s   | N/A         | 14.719s   | 99.97% | 60.48MB   | 1
INFO  : ftdock-opt006-pointer-induction.3 -static ../inputs/2pka.parsed -mobile ../inputs/5pti.parsed:
         UNKNOWN | 0.94        | 15.66s    | 0.94        | 15.665s   | 99.97% | 60.528MB  | 1
       Is faster? | CPU Speedup | CPU       | Elp Speedup | Elapsed   | CPU%   | Memory    | Reps
INFO  : ftdock-opt005-mem-gcentre.3 -static ../inputs/1hba.parsed -mobile ../inputs/5pti.parsed:
            N/A | N/A         | 39.68s    | N/A         | 39.692s   | 99.97% | 93.968MB  | 1
INFO  : ftdock-opt006-pointer-induction.3 -static ../inputs/1hba.parsed -mobile ../inputs/5pti.parsed:
         UNKNOWN | 1.037       | 38.254s   | 1.037       | 38.266s   | 99.97% | 93.884MB  | 1
       Is faster? | CPU Speedup | CPU       | Elp Speedup | Elapsed   | CPU%   | Memory    | Reps
INFO  : ftdock-opt005-mem-gcentre.3 -static ../inputs/4hhb.parsed -mobile ../inputs/5pti.parsed:
            N/A | N/A         | 48.826s   | N/A         | 48.842s   | 99.97% | 97.148MB  | 1
INFO  : ftdock-opt006-pointer-induction.3 -static ../inputs/4hhb.parsed -mobile ../inputs/5pti.parsed:
         UNKNOWN | 0.999       | 48.873s   | 0.999       | 48.889s   | 99.97% | 97.396MB  | 1
       Is faster? | CPU Speedup | CPU       | Elp Speedup | Elapsed   | CPU%   | Memory    | Reps
INFO  : ftdock-opt005-mem-gcentre.3 -static ../inputs/1ACB_rec.parsed -mobile ../inputs/1ACB_lig.parsed:
            N/A | N/A         | 18.831s   | N/A         | 18.837s   | 99.97% | 86.904MB  | 1
INFO  : ftdock-opt006-pointer-induction.3 -static ../inputs/1ACB_rec.parsed -mobile ../inputs/1ACB_lig.parsed:
         UNKNOWN | 1.048       | 17.973s   | 1.048       | 17.979s   | 99.97% | 86.996MB  | 1
```

Figura 6: Speedups de l'optimització 006 calculats amb l'script `speedup`

Com veiem, els speedups només són significatius per alguns testos concrets.

4 Altres intents i idees d'optimització

4.1 Vectorització

Mirant el codi, veiem que seria possible realitzar una vectorització dels càlculs dels bucles interiors, ja que tenim disponibles intrínseques per calcular arrels quadrades, a part de la resta de càlculs. Tot i així, si analitzem el codi assemblador ens adonem que el compilador ja està realitzant vectorització al codi:

```
        for (x = 0; x < grid_size; x++) {
4053f2: 45 85 ff                test   %r15d,%r15d
4053f5: 0f 8e d4 01 00 00       jle    4055cf <electric_field+0x34f>
4053fb: c4 c1 52 2a ef          vcvtsi2ss %r15d,%xmm5,%xmm5
405400: c5 fa 10 7c 24 38       vmovss 0x38(%rsp),%xmm7
405406: 31 ff                   xor    %edi,%edi
405408: c5 fb 10 1d 20 00 0b    vmovsd 0xb0020(%rip),%xmm3        # 4b5430 <_IO_stdin_used+0x1050>
40540f: 00
405410: 31 c9                   xor    %ecx,%ecx
405412: c5 c2 59 25 f6 ff 0a    vmulss 0xafff6(%rip),%xmm7,%xmm4        # 4b5410 <_IO_stdin_used+0x1030>
405419: 00
40541a: c5 7a 10 05 f6 ff 0a    vmovss 0xafff6(%rip),%xmm8        # 4b5418 <_IO_stdin_used+0x1038>
405421: 00
405422: c5 c2 5e ed             vdivss %xmm5,%xmm7,%xmm5

                    if (distance >= 2.0) {
                        if (distance >= 8.0) {
                            epsilon = 80;
```

```
                        } else {
                            if (distance <= 6.0) {
405426: c5 7a 10 15 f2 ff 0a     vmovss 0xafff2(%rip),%xmm10        # 4b5420 <_IO_stdin_used+0x1040>
40542d: 00
40542e: c5 d8 14 e4              vunpcklps %xmm4,%xmm4,%xmm4
405432: c5 f8 5a e4              vcvtps2pd %xmm4,%xmm4
405436: c5 d0 14 ed              vunpcklps %xmm5,%xmm5,%xmm5
40543a: c5 f8 5a ed              vcvtps2pd %xmm5,%xmm5
```

Per tant desestimem l'optimització, ja que tot i que probablement podríem obtenir un speedup major que el del compilador mitjançant una millor vectorització i unrolling, aquest molt probablement no seria prou significatiu com per justificar l'esforç.

4.2 Extracció d'invariants (`n004-invariant-extraction`)

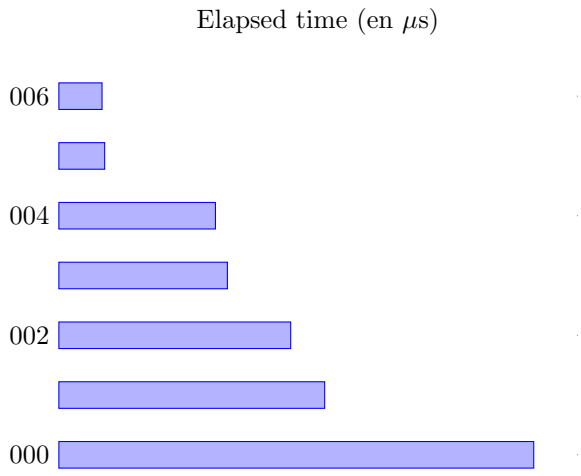
Una optimització que se'm va ocórrer, però que no va resultar ser bona, és la extracció de les invariants del càlcul de la distància en els bucles x, y, z. El fet és que la component $(atom.x - x)^2$ del càlcul de la distància es pot calcular en el bucle que recórre la x, i aprofitar el càlcul pels bucles interiors, i el mateix pel bucle y, en comptes de calcular-lo cada cop en les iteracions interiors.

El resultat de l'optimització, però, va ser un speedup negatiu de **0.8**. No vaig acabar d'aconseguir entendre quina era la raó exacta d'això, tot i que vaig analitzar rigurosament els codis ensamblador. Donada la seva complexitat, no vaig aconseguir esbrinar-ho, però intueixo que o bé la optimització no és compatible amb una altra optimització que ja realitzaba el compilador, i que és més important, fet que provoca que aquesta última quedi anulada, o bé el compilador ja realitzava aquesta optimització i el fet de fer-la explícita li ha donat menys llibertat per altres optimitzacions.

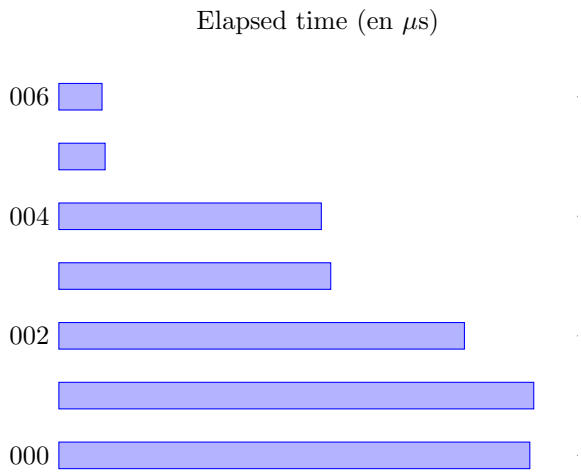
5 Anàlisi de rendiment

En aquest apartat es mostra l'evolució de l'elapsed time en microsegons les diferents optimitzacions. Tots els resultats s'han mesurat fent la mitjana de tres execucions.

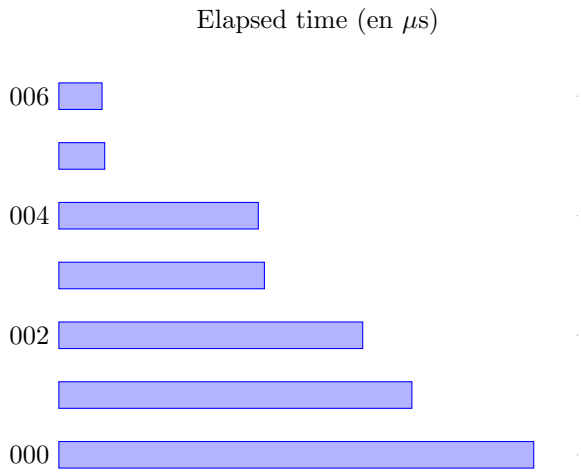
5.1 Test 1



5.2 Test 2



5.3 Test 3



5.4 Test 4