UVSQ université PARIS-SACLAY | ISTY Institut des Sciences et Techniques des Yvelines
CAMPUS DE MANTES EN YVELINES
CAMPUS DE SAINT-QUENTIN-EN-YVELINES

Architecture des ordinateurs – Bases

Problème des Ncorps

2021-2022

Encadrant :
JALBY William
BOLLORE Hugo
IBNABAR Mohamed-Salah

Réalisé par :
KHIARI Slim

Sommaire

Introduction 3
Baseline – AoS 4
Gcc 4
Icc 5
Icx 6
Les optimisations apportées à la baseline 7
1ere optimisation : SoA 7
2eme optimisation : remplacement des divisions par des multiplications 10
3eme optimisation : amélioration de la fonction “pow” à l’aide d’uneconstante 12
4eme optimisation : code motion 14
Les tentatives d’optimisations 16
1ere optimisation : suppression d’une boucle for 16
2eme optimisations : remplacement de constantes par leurs valeursdirectement (softening et dt) ... 16
3eme optimisation : déroulage de boucle 16
L’évolution des performances et du speed-up en fonction ducompilateur 17
Conclusion 18
Bibliographie 19

Introduction

Il existe plusieurs techniques d'optimisations d'un code, comme par exemple la vectorisation c'est-à-dire, nous allons effectuer plusieurs instructions scalaires en une seule fois à l'aide d'une instruction vectorielle, ou alors le déroulage afin de réduire les itérations de boucles. Le problème des Ncorps (Nbody) consiste à résoudre les équations de Newton pour N corps interagissant gravitationnellement, afin de prédire le mouvement des corps simulés. Dans le cadre de ce projet, nous allons appliquer un ensemble de techniques d'optimisations à nbody.c, sur les nœuds de calcul KNL du cluster OB-1 mis à disposition, sur 50000 particules. Ainsi, à chaque technique appliquée, nous allons utiliser l'outil perf afin de connaitre les points d'étranglements. D'abord, nous allons mesurer les performances de la version fournie afin d'obtenir une baseline qui sera utilisée pour effectuer des comparaisons entre les différentes versions. Par la suite, nous allons profiler les performances en utilisant le profileur perf, puis expliquer les résultats obtenus. Une fois les goulots d'étranglement identifiés nous allons donc proposer des optimisations, puis comparer les performances des versions optimisées à la version de base. Enfin, nous allons évaluer le speed-up par rapport à la baseline. Les compilateurs choisis sont gcc, icc et icx.

Baseline – AoS

En premier lieu, il faudra mesurer les performances de la version fournie afin d'obtenir une baseline qui sera utilisée pour effectuer des comparaisons entre les différentes versions. Nous allons améliorer les performances en utilisant les 3 compilateurs gcc, icc et icx. Nous allons utiliser perf afin de mesurer les performances à chaque amélioration du code.

Gcc

```
user4425@knl03:~/nbody3D$ perf stat taskset -c 10 ./nbody.g 50000

Total memory size: 1200000 B, 1171 KiB, 1 MiB

 Step     Time, s Interact/s  GFLOP/s
    0  9.233e+01  2.707e+07      0.6 *
    1  9.382e+01  2.665e+07      0.6 *
    2  9.367e+01  2.669e+07      0.6 *
    3  9.410e+01  2.657e+07      0.6
    4  9.490e+01  2.634e+07      0.6
    5  9.307e+01  2.686e+07      0.6
    6  9.216e+01  2.713e+07      0.6
    7  9.216e+01  2.712e+07      0.6
    8  9.218e+01  2.712e+07      0.6
    9  9.218e+01  2.712e+07      0.6
-----------------------------------------------------
Average performance:          0.6 +- 0.0 GFLOP/s
-----------------------------------------------------

 Performance counter stats for 'taskset -c 10 ./nbody.g 50000':

        930,219.01 msec task-clock                #    1.000 CPUs utilized
             2,141      context-switches          #    0.002 K/sec
                 1      cpu-migrations            #    0.000 K/sec
               440      page-faults               #    0.000 K/sec
 1,366,776,599,405      cycles                    #    1.469 GHz                      (49.99%)
   651,132,883,876      instructions              #    0.48  insn per cycle           (74.99%)
    25,245,485,926      branches                  #   27.139 M/sec                    (74.99%)
        26,110,360      branch-misses             #    0.10% of all branches          (75.02%)

     930.616834247 seconds time elapsed

     930.205524000 seconds user
       0.003998000 seconds sys
```

```
 97.31%  nbody.g  nbody.g           [.] move_particles
```

En utilisant perf report, nous constatons que la fonction dominante est la fonction "move_particles", elle occupe prés du 97.31% du temps passé, et plus précisément la majorité du temps est occupé par cettes instructions :

```
64.39 |       vdivss      %xmm1,%xmm12,%xmm1

      |     //Net force
      |     fx += dx / d_3_over_2; //13
11.82 |       vfmadd231ss %xmm1,%xmm4,%xmm5
```

Nous allons donc améliorer la fonction « move_particles » en se concentrant à chaque fois sur les instructions rouges gourmandes en terme de temps.

Nous refaisons les mêmes mesures pour les compilateurs restants à tester.

Icc

```
user4425@knl03:~/nbody3D$ perf stat taskset -c 11 ./nbody.icc 50000

Total memory size: 1200000 B, 1171 KiB, 1 MiB

 Step    Time, s Interact/s  GFLOP/s
    0  9.384e+00  2.664e+08      6.1 *
    1  9.381e+00  2.665e+08      6.1 *
    2  9.424e+00  2.653e+08      6.1 *
    3  9.420e+00  2.654e+08      6.1
    4  9.375e+00  2.667e+08      6.1
    5  9.374e+00  2.667e+08      6.1
    6  9.377e+00  2.666e+08      6.1
    7  9.371e+00  2.668e+08      6.1
    8  9.372e+00  2.668e+08      6.1
    9  9.377e+00  2.666e+08      6.1
-----------------------------------------------------
Average performance:        6.1 +- 0.0 GFLOP/s
-----------------------------------------------------

 Performance counter stats for 'taskset -c 11 ./nbody.icc 50000':

        93,933.38 msec task-clock                #    1.000 CPUs utilized
               70      context-switches          #    0.001 K/sec
                1      cpu-migrations            #    0.000 K/sec
            2,590      page-faults               #    0.028 K/sec
  136,058,725,333      cycles                    #    1.448 GHz                      (50.00%)
   62,772,476,378      instructions              #    0.46  insn per cycle           (75.00%)
      820,547,258      branches                  #    8.735 M/sec                    (75.00%)
        3,863,630      branch-misses             #    0.47% of all branches          (75.00%)

      93.957044165 seconds time elapsed

      93.878939000 seconds user
       0.055994000 seconds sys
```

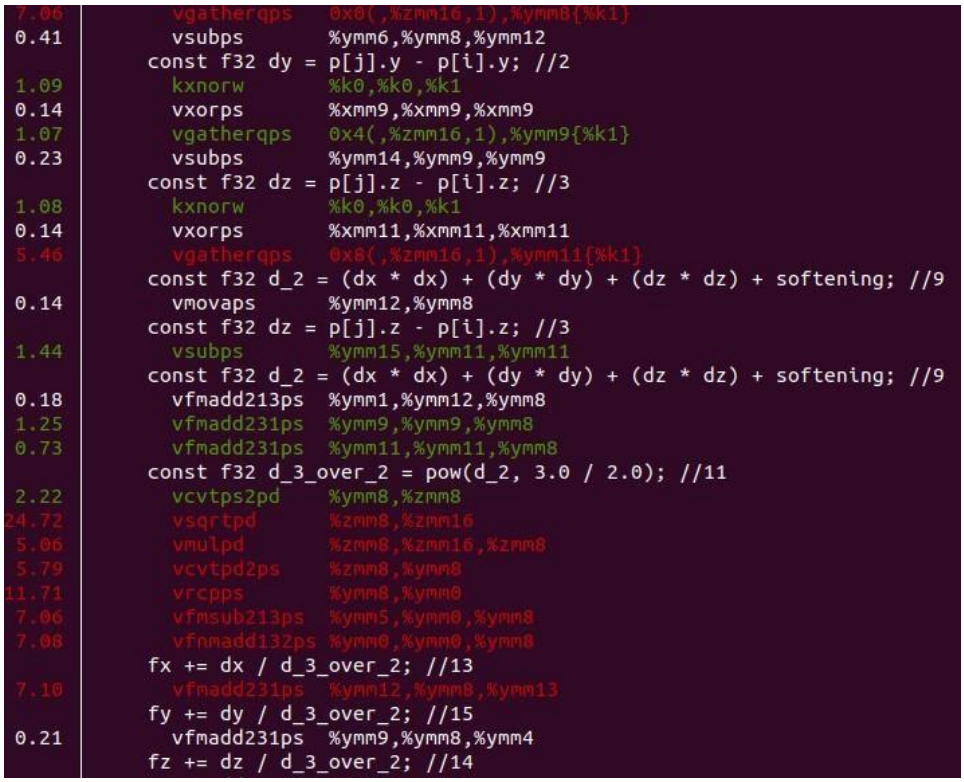
```
97.43%  nbody.icc  nbody.icc          [.] main
```

```
 23.24          vgatherdps    (%r10,%zmm10,8),%zmm29{%k3}
  0.55          vgatherdps    0x4(%r10,%zmm10,8),%zmm27{%k2}
  0.81          vsubps        %zmm7,%zmm29,%zmm28
                const f32 dy = p[j].y - p[i].y; //2
  0.43          vsubps        %zmm6,%zmm27,%zmm26
                const f32 dx = p[j].x - p[i].x; //1
  9.00          vgatherdps    0x8(%r10,%zmm10,8),%zmm25{%k1}
```

```
                fx += dx / d_3_over_2; //13
  6.25          vrcp28ps      %zmm17,%zmm17
```

Nous constatons cette fois ci que la fonction « «main » consomme plus de temps, mais cela revient à optimiser la fonction « move_particles ». En effet, elle occupe prés de 97.4% du temps, plus précisément, nous allons nous concentrer sur l'optimisation des instructions ci-dessus.

Icx




```
user4425@knl03:~/nbody3D$ perf stat taskset -c 11 ./nbody.icx 50000

Total memory size: 1200000 B, 1171 KiB, 1 MiB

 Step    Time, s Interact/s  GFLOP/s
    0  1.865e+01  1.341e+08      3.1 *
    1  1.866e+01  1.340e+08      3.1 *
    2  1.865e+01  1.340e+08      3.1 *
    3  1.865e+01  1.340e+08      3.1
    4  1.865e+01  1.340e+08      3.1
    5  1.864e+01  1.341e+08      3.1
    6  1.865e+01  1.340e+08      3.1
    7  1.866e+01  1.340e+08      3.1
    8  1.866e+01  1.340e+08      3.1
    9  1.866e+01  1.339e+08      3.1
-----------------------------------------------------
Average performance:          3.1 +- 0.0 GFLOP/s
-----------------------------------------------------

 Performance counter stats for 'taskset -c 11 ./nbody.icx 50000':

        186,619.41 msec task-clock                #    1.000 CPUs utilized
                95      context-switches          #    0.001 K/sec
                 1      cpu-migrations            #    0.000 K/sec
             2,588      page-faults               #    0.014 K/sec
   271,078,688,808      cycles                    #    1.453 GHz                      (50.00%)
   115,889,522,891      instructions              #    0.43  insn per cycle           (75.00%)
     3,194,043,858      branches                  #   17.115 M/sec                    (75.00%)
         6,632,359      branch-misses             #    0.21% of all branches         (75.01%)

     186.645583368 seconds time elapsed

     186.568333000 seconds user
       0.051995000 seconds sys
```

Concernant ce compilateur, nous observons une baisse de 50% en moyenne au niveau de la performance mais qui reste supérieur à gcc. Nous constatons que la fonction dominante est la fonction “move_particles”, elle occupe prés du 98.36% du temps passé. Plus plus détails, parmi les instructions qui consomment la majorité du temps, nous avons la fonction « pow ». Nous allons donc essayer de l’optimiser. De plus, les instructions les plus gourmandes en temps sont des instructions manipulant les flottants d’où notre objectif d’optimiser les opérations flottantes.

Voici un tableau résumant les performances de la baseline en utilisant les 3 compilateurs :

	gcc	icc	icx
Instructions par cycle	0.48	0.46	0.43
Nombre de cylces	1,366,776,599,405	136,058,725,333	271,078,688,808
Nombre d'instructions	651,132,883,876	62,772,476,378	115,889,522,891
Temps écoulé	930.616834247	93.957044165	186.645583368
Performances moyennes (GFLOP/s)	0.6	6.1	3.1

Les optimisations apportées à la baseline

Dans cette partie, nous allons présenter les différentes optimisations qui ont clairement amélioré le code. Voici donc dans l'ordre les 3 principales optimisations :

1ere optimisation : SoA

Cette technique d'optimisation permet de décomposer une séquence d'objets composites en un ensemble de séquences des éléments qui les composent. Elle tend à mener, d'une part, à des séquences plus compactes, donc à une meilleure utilisation de la mémoire vive de l'ordinateur, et d'autre part une meilleure utilisation du Cache, donc à une exécution plus rapide.

Voici les résultats des performances moyennes en utilisant cette technique avec les 3 compilateurs :

gcc

```
user4425@knl03:~/nbody3D$ perf stat -d ./nbody_aos.gcc 50000

Total memory size: 1200048 B, 1171 KiB, 1 MiB

 Step    Time, s Interact/s  GFLOP/s
    0  8.258e+00  3.027e+08      7.0 *
    1  8.142e+00  3.070e+08      7.1 *
    2  7.937e+00  3.150e+08      7.2 *
    3  7.776e+00  3.215e+08      7.4
    4  7.775e+00  3.215e+08      7.4
    5  7.777e+00  3.215e+08      7.4
    6  7.776e+00  3.215e+08      7.4
    7  7.776e+00  3.215e+08      7.4
    8  7.777e+00  3.214e+08      7.4
    9  7.780e+00  3.213e+08      7.4
-----------------------------------------------------
Average performance:          7.4 +- 0.0 GFLOP/s
-----------------------------------------------------

 Performance counter stats for './nbody_aos.gcc 50000':

         78,796.41 msec task-clock                #    1.000 CPUs utilized
                92      context-switches          #    0.001 K/sec
                 0      cpu-migrations            #    0.000 K/sec
               376      page-faults               #    0.005 K/sec
   115,380,636,625      cycles                    #    1.464 GHz                      (33.33%)
    42,513,794,683      instructions              #    0.37  insn per cycle           (49.99%)
     1,600,365,064      branches                  #   20.310 M/sec                    (50.00%)
         2,839,928      branch-misses             #    0.18% of all branches          (50.00%)
   <not supported>      L1-dcache-loads
       237,320,124      L1-dcache-load-misses                                         (33.34%)
     4,707,652,735      LLC-loads                 #   59.745 M/sec                    (33.33%)
   <not supported>      LLC-load-misses

     78.815836182 seconds time elapsed

     78.794002000 seconds user
      0.003999000 seconds sys
```

Les principales instructions qui consomment la majorité du temps pour gcc sont :

```
29.43       vsqrtpd      %zmm25,%zmm2
13.33       vsqrtpd      %zmm1,%zmm24
 0.71       vmulpd       %zmm25,%zmm2,%zmm2
 6.90       vmulpd       %zmm1,%zmm24,%zmm1
 1.45       vcvtpd2ps    %zmm2,%ymm2
 6.67       vcvtpd2ps    %zmm1,%ymm1
 4.11       vinsertf64x4 $0x1,%ymm1,%zmm2,%zmm2
13.65       vrcp28ps     %zmm2,%zmm2
          fx += dx / d_3_over_2; //13
 8.24       vfmadd231ps  %zmm2,%zmm20,%zmm7
          fy += dy / d_3_over_2; //15
 1.32       vfmadd231ps  %zmm19,%zmm2,%zmm8
          fz += dz / d_3_over_2; //17
 1.28       vfmadd231ps  %zmm18,%zmm2,%zmm9
```

Icc

```
Total memory size: 1200048 B, 1171 KiB, 1 MiB

 Step    Time, s Interact/s  GFLOP/s
    0  5.632e+00  4.439e+08     10.2 *
    1  5.549e+00  4.505e+08     10.4 *
    2  5.534e+00  4.517e+08     10.4 *
    3  5.524e+00  4.526e+08     10.4
    4  5.541e+00  4.511e+08     10.4
    5  5.526e+00  4.524e+08     10.4
    6  5.533e+00  4.518e+08     10.4
    7  5.536e+00  4.516e+08     10.4
    8  5.527e+00  4.523e+08     10.4
    9  5.541e+00  4.512e+08     10.4
-----------------------------------------------------
Average performance:          10.4 +- 0.0 GFLOP/s
-----------------------------------------------------

 Performance counter stats for './nbody_aos.icc 50000':

        55,505.58 msec task-clock                #    0.999 CPUs utilized
              117      context-switches          #    0.002 K/sec
                0      cpu-migrations            #    0.000 K/sec
            2,531      page-faults               #    0.046 K/sec
   82,309,320,611      cycles                    #    1.483 GHz                      (33.34%)
   61,143,859,236      instructions              #    0.74  insn per cycle           (50.00%)
    1,597,720,529      branches                  #   28.785 M/sec                    (50.01%)
        2,657,144      branch-misses             #    0.17% of all branches          (49.99%)
  <not supported>      L1-dcache-loads
      240,498,499      L1-dcache-load-misses                                         (33.33%)
    5,391,291,247      LLC-loads                 #   97.131 M/sec                    (33.33%)
  <not supported>      LLC-load-misses

      55.547286046 seconds time elapsed

      55.476051000 seconds user
       0.031995000 seconds sys
```

Les principales instructions qui consomment la majorité du temps pour icc sont :

```
 8.95       vcvtpd2ps    %zmm27,%ymm27
 1.00       vfmadd213pd  %zmm31,%zmm30,%zmm29
 1.00       vfixupimmpd  $0x70,0xf8a7(%rip){1to8},%zmm4,%zmm29        # 4121e8 <_IO_stdin_used+0x1e8>
 2.80       vmulpd       %zmm4,%zmm29,%zmm28
10.22       vcvtpd2ps    %zmm28,%ymm5
 5.85       vinsertf64x4 $0x1,%ymm5,%zmm27,%zmm27
          fx += dx / d_3_over_2; //13
19.97       vrcp28ps     %zmm27,%zmm27
11.69       vfmadd231ps  %zmm23,%zmm27,%zmm3
```

Icx

```
user4425@knl03:~/nbody3D$ perf stat -d ./nbody_aos.icx 50000

Total memory size: 1200048 B, 1171 KiB, 1 MiB

 Step    Time, s Interact/s  GFLOP/s
    0  1.047e+01  2.388e+08      5.5 *
    1  1.049e+01  2.384e+08      5.5 *
    2  1.050e+01  2.382e+08      5.5 *
    3  1.079e+01  2.317e+08      5.3
    4  1.056e+01  2.367e+08      5.4
    5  1.057e+01  2.366e+08      5.4
    6  1.101e+01  2.271e+08      5.2
    7  1.109e+01  2.253e+08      5.2
    8  1.110e+01  2.252e+08      5.2
    9  1.110e+01  2.252e+08      5.2
-----------------------------------------------------
Average performance:          5.3 +- 0.1 GFLOP/s
-----------------------------------------------------

 Performance counter stats for './nbody_aos.icx 50000':

        107,700.70 msec task-clock                #    1.000 CPUs utilized
               232      context-switches          #    0.002 K/sec
                 0      cpu-migrations            #    0.000 K/sec
             2,526      page-faults               #    0.023 K/sec
   154,892,061,181      cycles                    #    1.438 GHz                      (33.34%)
    72,112,086,300      instructions              #    0.47  insn per cycle           (50.00%)
     3,171,090,078      branches                  #   29.444 M/sec                    (50.01%)
         3,899,210      branch-misses             #    0.12% of all branches          (49.99%)
   <not supported>      L1-dcache-loads
       275,092,852      L1-dcache-load-misses                                       (33.33%)
     5,574,009,892      LLC-loads                 #   51.755 M/sec                    (33.33%)
   <not supported>      LLC-load-misses

     107.751589682 seconds time elapsed

     107.670845000 seconds user
       0.031987000 seconds sys
```

Les principales instructions qui consomment la majorité du temps pour icx sont

```
 20.99        vsqrtpd      %zmm14,%zmm18
  5.26        vmulpd       %zmm14,%zmm18,%zmm14
  6.61        vcvtpd2ps    %zmm14,%ymm14
 12.46        vrcpps       %ymm14,%ymm10
 10.96        vfmsub213ps  %ymm4,%ymm10,%ymm14
 12.44        vfnmadd132ps %ymm10,%ymm10,%ymm14
                    fx += dx / d_3_over_2; //13
 12.41        vfmadd231ps  %ymm9,%ymm14,%ymm1
```

Résumons maintenant des performances moyennes dans un tableau.

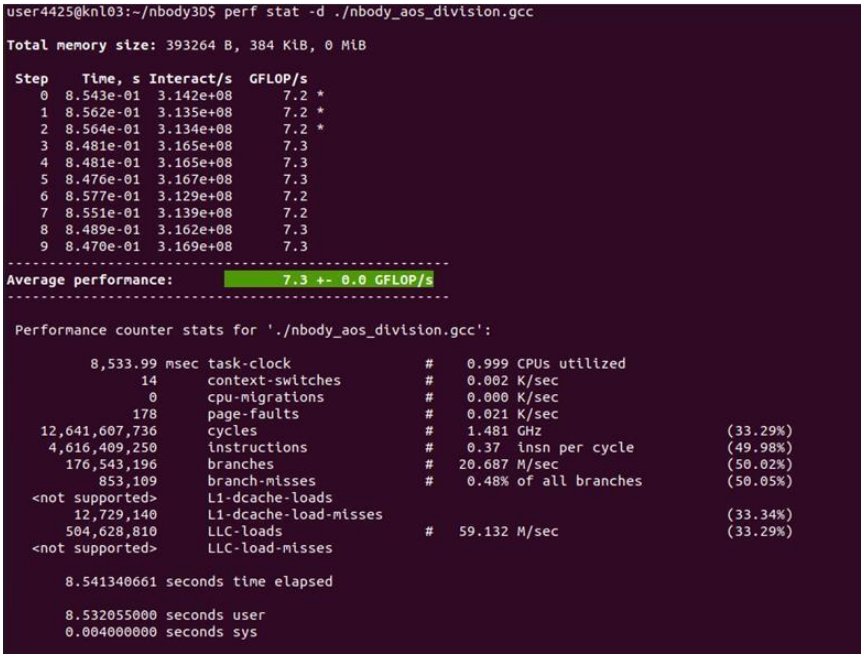
	gcc	icc	icx
Instructions par cycle	0.37	0.74	0.47
Nombre de cylces	115,380,636,625	82,309,320,611	154,892,061,181
Nombre d'instructions	42,513,794,683	61,143,859,236	72,112,086,300
Temps écoulé	78.815836182	55.547286046	107,75
Performances moyennes (GFLOP/s)	7.4	10.4	5.3
Speed-up	7.4/0.6=12.33	10.4/6.1=1.7	5.3/3.1=1.7

Ainsi, nous remarquons une amélioration non négligeable des performances moyennes donc du speed-up.

2eme optimisation : remplacement des divisions par des multiplications de l'inverse

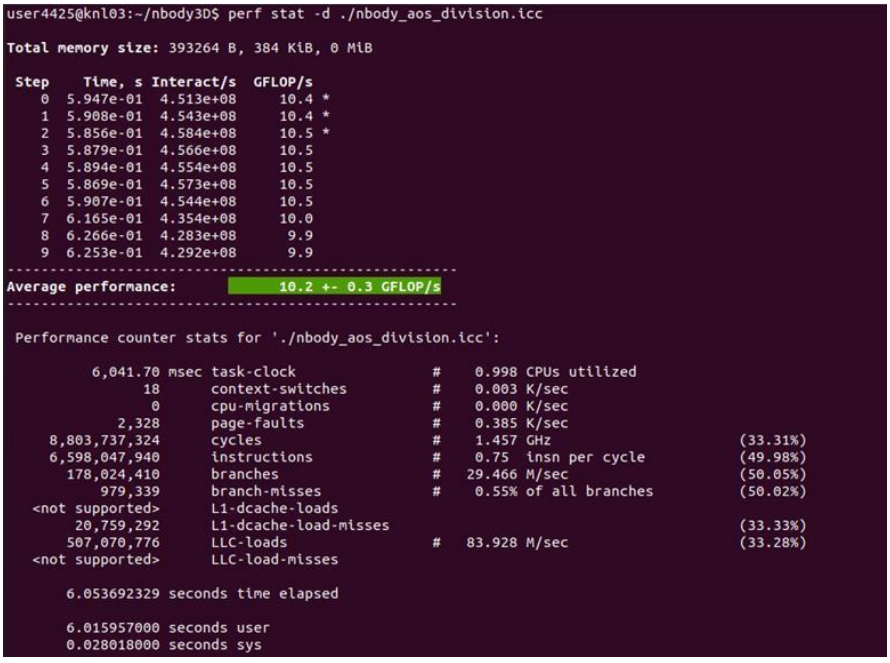
Multiplier par l'inverse du diviseur est une bonne optimisation parce qu'une division en virgule flottante coûte plus chère qu'une multiplication en virgule flottante. Nous remarquons une amélioration au niveau du compilateur icx, mais pour gcc et icc une très légère baisse voir une stabilité au niveau des performances. Voici donc les performances moyennes mesurées, ainsi que les instructions qui coûtent la majorité du temps.

Gcc



```
user4425@knl03:~/nbody3D$ perf stat -d ./nbody_aos_division.gcc

Total memory size: 393264 B, 384 KiB, 0 MiB

 Step     Time, s Interact/s  GFLOP/s
    0  8.543e-01  3.142e+08      7.2 *
    1  8.562e-01  3.135e+08      7.2 *
    2  8.564e-01  3.134e+08      7.2 *
    3  8.481e-01  3.165e+08      7.3
    4  8.481e-01  3.165e+08      7.3
    5  8.476e-01  3.167e+08      7.3
    6  8.577e-01  3.129e+08      7.2
    7  8.551e-01  3.139e+08      7.2
    8  8.489e-01  3.162e+08      7.3
    9  8.470e-01  3.169e+08      7.3
-----------------------------------------------------
Average performance:          7.3 +- 0.0 GFLOP/s
-----------------------------------------------------

 Performance counter stats for './nbody_aos_division.gcc':

          8,533.99 msec task-clock                #    0.999 CPUs utilized
                14      context-switches          #    0.002 K/sec
                 0      cpu-migrations            #    0.000 K/sec
               178      page-faults               #    0.021 K/sec
    12,641,607,736      cycles                    #    1.481 GHz                      (33.29%)
     4,616,409,250      instructions              #    0.37  insn per cycle           (49.98%)
       176,543,196      branches                  #   20.687 M/sec                    (50.02%)
           853,109      branch-misses             #    0.48% of all branches          (50.05%)
   <not supported>      L1-dcache-loads
        12,729,140      L1-dcache-load-misses                                       (33.34%)
       504,628,810      LLC-loads                 #   59.132 M/sec                    (33.29%)
   <not supported>      LLC-load-misses

       8.541340661 seconds time elapsed

       8.532055000 seconds user
       0.004000000 seconds sys
```

Icc



```
user4425@knl03:~/nbody3D$ perf stat -d ./nbody_aos_division.icc

Total memory size: 393264 B, 384 KiB, 0 MiB

 Step     Time, s Interact/s  GFLOP/s
    0  5.947e-01  4.513e+08     10.4 *
    1  5.908e-01  4.543e+08     10.4 *
    2  5.856e-01  4.584e+08     10.5 *
    3  5.879e-01  4.566e+08     10.5
    4  5.894e-01  4.554e+08     10.5
    5  5.869e-01  4.573e+08     10.5
    6  5.907e-01  4.544e+08     10.5
    7  6.165e-01  4.354e+08     10.0
    8  6.266e-01  4.283e+08      9.9
    9  6.253e-01  4.292e+08      9.9
-----------------------------------------------------
Average performance:         10.2 +- 0.3 GFLOP/s
-----------------------------------------------------

 Performance counter stats for './nbody_aos_division.icc':

          6,041.70 msec task-clock                #    0.998 CPUs utilized
                18      context-switches          #    0.003 K/sec
                 0      cpu-migrations            #    0.000 K/sec
             2,328      page-faults               #    0.385 K/sec
     8,803,737,324      cycles                    #    1.457 GHz                      (33.31%)
     6,598,047,940      instructions              #    0.75  insn per cycle           (49.98%)
       178,024,410      branches                  #   29.466 M/sec                    (50.05%)
           979,339      branch-misses             #    0.55% of all branches          (50.02%)
   <not supported>      L1-dcache-loads
        20,759,292      L1-dcache-load-misses                                       (33.33%)
       507,070,776      LLC-loads                 #   83.928 M/sec                    (33.28%)
   <not supported>      LLC-load-misses

       6.053692329 seconds time elapsed

       6.015957000 seconds user
       0.028018000 seconds sys
```

Icx

```
user4425@knl03:~/nbody3D$ perf stat -d ./nbody_aos_division.icx

Total memory size: 393264 B, 384 KiB, 0 MiB

 Step    Time, s Interact/s  GFLOP/s
    0  9.277e-01  2.893e+08      6.7 *
    1  9.273e-01  2.895e+08      6.7 *
    2  9.281e-01  2.892e+08      6.7 *
    3  9.288e-01  2.890e+08      6.6
    4  9.285e-01  2.891e+08      6.6
    5  9.286e-01  2.891e+08      6.6
    6  9.282e-01  2.892e+08      6.7
    7  9.291e-01  2.889e+08      6.6
    8  9.282e-01  2.892e+08      6.7
    9  9.283e-01  2.891e+08      6.7
-----------------------------------------------------
Average performance:          6.6 +- 0.0 GFLOP/s
-----------------------------------------------------

 Performance counter stats for './nbody_aos_division.icx':

          9,329.31 msec task-clock                #    0.999 CPUs utilized
                23      context-switches          #    0.002 K/sec
                 0      cpu-migrations            #    0.000 K/sec
             2,328      page-faults               #    0.250 K/sec
    12,993,984,307      cycles                    #    1.393 GHz                      (33.27%)
     4,918,277,898      instructions              #    0.38  insn per cycle           (49.96%)
       178,507,431      branches                  #   19.134 M/sec                    (50.03%)
         1,046,701      branch-misses             #    0.59% of all branches          (50.06%)
   <not supported>      L1-dcache-loads
        13,176,245      L1-dcache-load-misses                                       (33.34%)
       499,554,738      LLC-loads                 #   53.547 M/sec                    (33.30%)
   <not supported>      LLC-load-misses

       9.342755639 seconds time elapsed

       9.291470000 seconds user
       0.039963000 seconds sys
```

Voici donc les instructions en particulier qui consomment plus de temps :

```
16.04        vsqrtpd      %zmm14,%zmm19
 1.27        vcvtps2pd    %ymm8,%zmm8
14.03        vsqrtpd      %zmm8,%zmm20
 6.02        vmulpd       %zmm8,%zmm20,%zmm8
 0.66        vmulpd       %zmm14,%zmm19,%zmm14
 1.28        vcvtpd2ps    %zmm14,%ymm14
 5.63        vcvtpd2ps    %zmm8,%ymm8
 3.81        vinsertf64x4 $0x1,%ymm8,%zmm14,%zmm8
12.60        vrcp14ps     %zmm8,%zmm14
 7.70        vfmsub213ps  %zmm21,%zmm14,%zmm8
 7.68        vfnmadd132ps %zmm14,%zmm14,%zmm8
                    fx += dx *(1/d_3_over_2); //13
 7.57        vfmadd231ps  %zmm17,%zmm8,%zmm1
                    fy += dy *(1/d_3_over_2); //15
```

Nous avons essayé d'améliorer encore plus le fusedmultiply-add avec l'option de compilation –mfma. La seule option trouvée a été d'ajouter -mfma lors de la compilation. Malheureusement, cette option est incluse avec -march=native, donc il n'est pas possible d'améliorer encore plus cettes instructions.

Résumons les différentes mesures avec les 3 compilateurs d'un tableau :

	gcc	icc	icx
Instructions par cycle	0.37	0.75	0.37
Nombre de cylces	12,641,607,736	8,803,737,324	12,993,964,307
Nombre d'instructions	4,616,409,250	6,598,047,940	4,918,277,898
Temps écoulé	8,541	6.053692329	9.34
Performances moyennes (GFLOP/s)	7.3	10.2	6.6
Speed-up	7.3/0.6=12.16	10.2/6.1=1.67	6.6/3.3=2

3eme optimisation : amélioration de la fonction “pow” à l’aide d’une constante

Grace à perf report, nous avons remarqué que “pow” consomme une partie importante du temps. Nous allons donc optimiser cette fonction. Nous savons que x à la puissance 1.5 est égale à x à la puissance 1 multiplié par x à la puissance 0.5 c’est-à-dire la racine de x. Or, multiplier x par la racine de x prend plus de temps que multiplier x par une constante. Alors, nous pouvons écrire d_3_over_2 de cette manière :

```
const f32 racine_carre_d_2 = sqrt(d_2); //11
const f32 d_3_over_2 = d_2*racine_carre_d_2;
```

gcc

```
user4425@knl03:~/nbody3D$ perf stat -d ./nbody_aos_division.gcc

Total memory size: 393264 B, 384 KiB, 0 MiB

 Step    Time, s Interact/s  GFLOP/s
    0  2.556e-01  1.050e+09     24.2 *
    1  2.545e-01  1.055e+09     24.3 *
    2  2.549e-01  1.053e+09     24.2 *
    3  2.547e-01  1.054e+09     24.2
    4  2.545e-01  1.055e+09     24.3
    5  2.550e-01  1.053e+09     24.2
    6  2.549e-01  1.053e+09     24.2
    7  2.571e-01  1.044e+09     24.0
    8  2.579e-01  1.041e+09     23.9
    9  2.582e-01  1.039e+09     23.9
-----------------------------------------------------
Average performance:        24.1 +- 0.1 GFLOP/s
-----------------------------------------------------

 Performance counter stats for './nbody_aos_division.gcc':

          2,572.96 msec task-clock                #    0.999 CPUs utilized
                 4      context-switches          #    0.002 K/sec
                 0      cpu-migrations            #    0.000 K/sec
               177      page-faults               #    0.069 K/sec
     3,810,305,625      cycles                    #    1.481 GHz                      (33.27%)
     3,601,614,486      instructions              #    0.95  insn per cycle           (50.06%)
       174,515,878      branches                  #   67.827 M/sec                    (50.10%)
           704,286      branch-misses             #    0.40% of all branches          (50.10%)
   <not supported>      L1-dcache-loads
        13,011,356      L1-dcache-load-misses                                       (33.27%)
       506,454,377      LLC-loads                 #  196.837 M/sec                    (33.27%)
   <not supported>      LLC-load-misses

       2.576020832 seconds time elapsed

       2.566375000 seconds user
       0.008007000 seconds sys
```

```
  7.24          vmulps      %zmm2,%zmm1,%zmm1
                fx += dx *(1/d_3_over_2); //13
 22.32          vrcp28ps    %zmm1,%zmm1
 22.65          vfmadd231ps %zmm1,%zmm20,%zmm8
```

Icc

```
user4425@knl03:~/nbody3D$ perf stat -d ./nbody_aos_division.icc

Total memory size: 393264 B, 384 KiB, 0 MiB

 Step    Time, s Interact/s  GFLOP/s
    0  2.236e-01  1.200e+09     27.6 *
    1  2.225e-01  1.207e+09     27.8 *
    2  2.218e-01  1.210e+09     27.8 *
    3  2.192e-01  1.225e+09     28.2
    4  2.200e-01  1.220e+09     28.1
    5  2.198e-01  1.221e+09     28.1
    6  2.200e-01  1.220e+09     28.1
    7  2.194e-01  1.223e+09     28.1
    8  2.195e-01  1.223e+09     28.1
    9  2.191e-01  1.225e+09     28.2
-----------------------------------------------------
Average performance:        28.1 +- 0.0 GFLOP/s
-----------------------------------------------------

 Performance counter stats for './nbody_aos_division.icc':

          2,251.33 msec task-clock                #    0.998 CPUs utilized
                 9      context-switches          #    0.004 K/sec
                 0      cpu-migrations            #    0.000 K/sec
             2,330      page-faults               #    0.001 M/sec
     3,333,008,625      cycles                    #    1.480 GHz                      (33.21%)
     3,411,285,693      instructions              #    1.02  insn per cycle           (49.90%)
       177,184,954      branches                  #   78.703 M/sec                    (50.07%)
           959,593      branch-misses             #    0.54% of all branches          (50.09%)
   <not supported>      L1-dcache-loads
        13,307,673      L1-dcache-load-misses                                       (33.40%)
       507,987,967      LLC-loads                 #  225.640 M/sec                    (33.23%)
   <not supported>      LLC-load-misses

       2.255401747 seconds time elapsed

       2.228825000 seconds user
       0.024008000 seconds sys
```

```
        const f32 dx = p->x[j] - p->x[i]; //1
 5.80 580:   vmovups       (%r11,%rsi,4),%zmm24
        const f32 dy = p->y[j] - p->y[i]; //2
 0.04       vmovups       (%r10,%rsi,4),%zmm25
        const f32 d_2 = (dx * dx) + (dy * dy) + (dz * dz) + softening; //9
 5.55       vmovaps       %zmm19,%zmm28
        const f32 dx = p->x[j] - p->x[i]; //1
 0.03       vsubps        %zmm23,%zmm24,%zmm31
        const f32 dy = p->y[j] - p->y[i]; //2
 5.46       vsubps        %zmm22,%zmm25,%zmm24
        const f32 dz = p->z[j] - p->z[i]; //3
 0.33       vmovups       (%rdx,%rsi,4),%zmm26
        for (u64 j = 0; j < n; j++)   {
 5.48       add           $0x10,%rsi
        const f32 d_2 = (dx * dx) + (dy * dy) + (dz * dz) + softening; //9
 0.01       vfmadd231ps   %zmm31,%zmm31,%zmm28
        const f32 dz = p->z[j] - p->z[i]; //3
 5.31       vsubps        %zmm16,%zmm26,%zmm25
        const f32 d_2 = (dx * dx) + (dy * dy) + (dz * dz) + softening; //9
 0.03       vfmadd231ps   %zmm24,%zmm24,%zmm28
 5.35       vfmadd231ps   %zmm25,%zmm25,%zmm28
        const f32 racine_carre_d_2 = sqrt(d_2); //11
 0.14       vrsqrt28ps    {sae},%zmm28,%zmm27
 6.03       vrcp28ps      {sae},%zmm27,%zmm29
        const f32 d_3_over_2 = d_2*racine_carre_d_2;
 3.85       vmulps        %zmm29,%zmm28,%zmm30
        fx += dx *(1/d_3_over_2); //13
18.85       vrcp28ps      %zmm30,%zmm27
24.50       vfmadd231ps   %zmm31,%zmm27,%zmm3
        fy += dy *(1/d_3_over_2); //15
 5.38       vfmadd231ps   %zmm24,%zmm27,%zmm1
        fz += dz *(1/d_3_over_2); //17
 1.83       vfmadd231ps   %zmm25,%zmm27,%zmm0
        for (u64 j = 0; j < n; j++)   {
 5.45       cmp           %r14,%rsi
```

Icx

```
Total memory size: 393264 B, 384 KiB, 0 MiB

 Step    Time, s Interact/s  GFLOP/s
    0  2.744e-01  9.782e+08     22.5 *
    1  2.734e-01  9.817e+08     22.6 *
    2  2.736e-01  9.812e+08     22.6 *
    3  2.734e-01  9.818e+08     22.6
    4  2.703e-01  9.932e+08     22.8
    5  2.704e-01  9.925e+08     22.8
    6  2.696e-01  9.957e+08     22.9
    7  2.710e-01  9.907e+08     22.8
    8  2.695e-01  9.961e+08     22.9
    9  2.703e-01  9.931e+08     22.8
-----------------------------------------------------
Average performance:          22.8 +- 0.1 GFLOP/s
-----------------------------------------------------

 Performance counter stats for './nbody_aos_division.icx':

          2,762.58 msec task-clock                #    0.999 CPUs utilized
                 9      context-switches          #    0.003 K/sec
                 0      cpu-migrations            #    0.000 K/sec
             2,330      page-faults               #    0.843 K/sec
     4,086,780,244      cycles                    #    1.479 GHz                      (33.31%)
     3,907,508,135      instructions              #    0.96  insn per cycle           (50.05%)
       176,202,882      branches                  #   63.782 M/sec                    (50.05%)
           768,076      branch-misses             #    0.44% of all branches         (50.04%)
   <not supported>      L1-dcache-loads
        12,653,532      L1-dcache-load-misses                                       (33.30%)
       501,954,813      LLC-loads                 #  181.698 M/sec                  (33.30%)
   <not supported>      LLC-load-misses

       2.766394059 seconds time elapsed

       2.739979000 seconds user
       0.023999000 seconds sys
```

```
  6.37 |       vrsqrt14ps    %zmm19,%zmm20
  6.46 |       vmulps        %zmm20,%zmm19,%zmm19
 10.01 |       vfmadd213ps   %zmm12,%zmm20,%zmm19
  2.05 |       vmulps        %zmm21,%zmm20,%zmm20
 12.77 |       vmulps        %zmm19,%zmm20,%zmm19
 19.66 |       vfmadd231ps   %zmm16,%zmm19,%zmm3
```

Ainsi, nous constatons qu'après les 3 optimisations, l'ordre de la performance des compilateurs est toujours le même, nous avons icc en premier, ensuite gcc et enfin icx.

Voici le tableau résumant les résultats ci-dessous :

	gcc	icc	icx
Instructions par cycle	0.95	1.02	0.96
Nombre de cylces	3,810,305,625	3,333,008,625	4,086,780,244
Nombre d'instructions	3,601,614,486	3,411,285,493	3,907,508,135
Temps écoulé	2.57	2.25	2.76
Performances moyennes (GFLOP/s)	24.1	28.1	22.8
Speed-up	24.1/0.6=40.16	28.1/6.1=4.6	22.8/3.3=6.9

4eme optimisation : code motion

Il s'agit d'un déplacement particulier du code afin d'augmenter les performances. Cette action est effectuée par la majorité des compilateurs. En effet, nous avons simplement remplacé « p->x[i] » par une constante au-dessus de la boucle j « pxi », et de même pour y et z. Bien évidement, il ne s'agit pas d'une solution de l'optimisation des instructions en rouges des parties précédentes.

Gcc

```
user4425@knl03:~/nbody3D$ perf stat -d ./nbody_aos_division_puissance_code_motion.gcc 50000

Total memory size: 1200048 B, 1171 KiB, 1 MiB

 Step     Time, s Interact/s  GFLOP/s
    0  2.254e+00  1.109e+09     25.5 *
    1  2.236e+00  1.118e+09     25.7 *
    2  2.249e+00  1.111e+09     25.6 *
    3  2.244e+00  1.114e+09     25.6
    4  2.251e+00  1.111e+09     25.5
    5  2.260e+00  1.106e+09     25.4
    6  2.261e+00  1.106e+09     25.4
    7  2.252e+00  1.110e+09     25.5
    8  2.241e+00  1.115e+09     25.7
    9  2.258e+00  1.107e+09     25.5
-----------------------------------------------------
Average performance:          25.5 +- 0.1 GFLOP/s
-----------------------------------------------------

 Performance counter stats for './nbody_aos_division_puissance_code_motion.gcc 50000':

         22,531.57 msec task-clock                #    0.999 CPUs utilized
                55      context-switches          #    0.002 K/sec
                 0      cpu-migrations            #    0.000 K/sec
               376      page-faults               #    0.017 K/sec
    33,342,889,932      cycles                    #    1.480 GHz                      (33.35%)
    33,056,266,820      instructions              #    0.99  insn per cycle           (50.02%)
     1,586,922,634      branches                  #   70.431 M/sec                    (50.01%)
         1,909,047      branch-misses             #    0.12% of all branches          (50.00%)
   <not supported>      L1-dcache-loads
       229,149,132      L1-dcache-load-misses                                       (33.33%)
     4,720,343,515      LLC-loads                 #  209.499 M/sec                    (33.33%)
   <not supported>      LLC-load-misses

      22.544306369 seconds time elapsed

      22.528975000 seconds user
       0.003998000 seconds sys
```

Icc

```
user4425@knl03:~/nbody3D$ perf stat -d ./nbody_aos_division_puissance_code_motion.icc 50000

Total memory size: 1200048 B, 1171 KiB, 1 MiB

 Step    Time, s Interact/s  GFLOP/s
    0  2.014e+00  1.241e+09     28.6 *
    1  1.961e+00  1.275e+09     29.3 *
    2  1.961e+00  1.275e+09     29.3 *
    3  1.956e+00  1.278e+09     29.4
    4  1.962e+00  1.274e+09     29.3
    5  1.953e+00  1.280e+09     29.4
    6  1.962e+00  1.274e+09     29.3
    7  1.956e+00  1.278e+09     29.4
    8  1.964e+00  1.273e+09     29.3
    9  1.957e+00  1.277e+09     29.4
-----------------------------------------------------
Average performance:          29.4 +- 0.1 GFLOP/s
-----------------------------------------------------

 Performance counter stats for './nbody_aos_division_puissance_code_motion.icc 50000':

         19,713.75 msec task-clock                #    0.999 CPUs utilized
                71      context-switches          #    0.004 K/sec
                 0      cpu-migrations            #    0.000 K/sec
             2,529      page-faults               #    0.128 K/sec
    29,191,275,734      cycles                    #    1.481 GHz                      (33.33%)
    31,393,896,413      instructions              #    1.08  insn per cycle           (49.99%)
     1,586,306,995      branches                  #   80.467 M/sec                    (50.00%)
         2,095,888      branch-misses             #    0.13% of all branches         (49.99%)
   <not supported>      L1-dcache-loads
       231,649,669      L1-dcache-load-misses                                       (33.34%)
     4,817,477,985      LLC-loads                 #  244.371 M/sec                    (33.33%)
   <not supported>      LLC-load-misses

      19.737574911 seconds time elapsed

      19.672290000 seconds user
       0.043973000 seconds sys
```

Icx

```
user4425@knl03:~/nbody3D$ perf stat -d ./nbody_aos_division_puissance_code_motion.icx 50000

Total memory size: 1200048 B, 1171 KiB, 1 MiB

 Step    Time, s Interact/s  GFLOP/s
    0  2.656e+00  9.412e+08     21.6 *
    1  2.651e+00  9.429e+08     21.7 *
    2  2.553e+00  9.791e+08     22.5 *
    3  2.569e+00  9.732e+08     22.4
    4  2.566e+00  9.744e+08     22.4
    5  2.555e+00  9.785e+08     22.5
    6  2.566e+00  9.741e+08     22.4
    7  2.565e+00  9.747e+08     22.4
    8  2.555e+00  9.783e+08     22.5
    9  2.566e+00  9.743e+08     22.4
-----------------------------------------------------
Average performance:          22.4 +- 0.0 GFLOP/s
-----------------------------------------------------

 Performance counter stats for './nbody_aos_division_puissance_code_motion.icx 50000':

         25,861.50 msec task-clock                #    1.000 CPUs utilized
                43      context-switches          #    0.002 K/sec
                 0      cpu-migrations            #    0.000 K/sec
             2,525      page-faults               #    0.098 K/sec
    38,301,798,879      cycles                    #    1.481 GHz                      (33.32%)
    36,067,213,084      instructions              #    0.94  insn per cycle           (49.99%)
     1,582,941,292      branches                  #   61.208 M/sec                    (49.99%)
         2,169,768      branch-misses             #    0.14% of all branches         (50.01%)
   <not supported>      L1-dcache-loads
       153,055,304      L1-dcache-load-misses                                       (33.34%)
     4,850,749,021      LLC-loads                 #  187.566 M/sec                    (33.34%)
   <not supported>      LLC-load-misses

      25.869436053 seconds time elapsed

      25.826834000 seconds user
       0.035987000 seconds sys
```

Nous remarquons une amélioration sur gcc et icc sauf icx qui présente une stabilité au niveau des performances.

	gcc	icc	icx
Instructions par cycle	0.99	1.08	0.94
Nombre de cylces	33,342,889,932	29,191,275,734	38,301,798,879
Nombre d'instructions	33,056,266,820	31,393,896,413	36,067,213,084
Temps écoulé	22.54	19.73	25.86
Performances moyennes (GFLOP/s)	25.5	29.4	22.4
Speed-up	25.5/0.6=42.5	29.4/6.1=4.82	22.4/3.3=6.78

Les tentatives d'optimisations

Nous avons essayé d'optimiser encore plus mais malheureusement nous observons une stabilité ou voir une dégradation de la performance moyenne. Nous allons présenter dans cette partie les principales optimisations que nous avons essayées. L'ensemble des techniques d'optimisations testées ci-dessus ne sont pas retenues parce qu'elles ne présentent aucun avantage par rapport à la dernière version optimisée de nbody.c.

1ere optimisation : suppression d'une boucle for

Nous avons constaté qu'il y a une redondance de calcul, la dernière boucle for dans la fonction à optimiser, « move _particles ». En utilisant la technique d'optimisation de la fusion des boucles, c'est-à-dire en supprimant la dernière boucle for de la fonction "move_particles" et mettre son corps dans la grande boucle, nous constatons qu'il n y a aucune amélioration des performances, c'est-à-dire que les performances restent stables pour les 3 compilateurs gcc, icc, et icx.

2eme optimisation : remplacement de constantes par leurs valeurs directement (softening et dt)

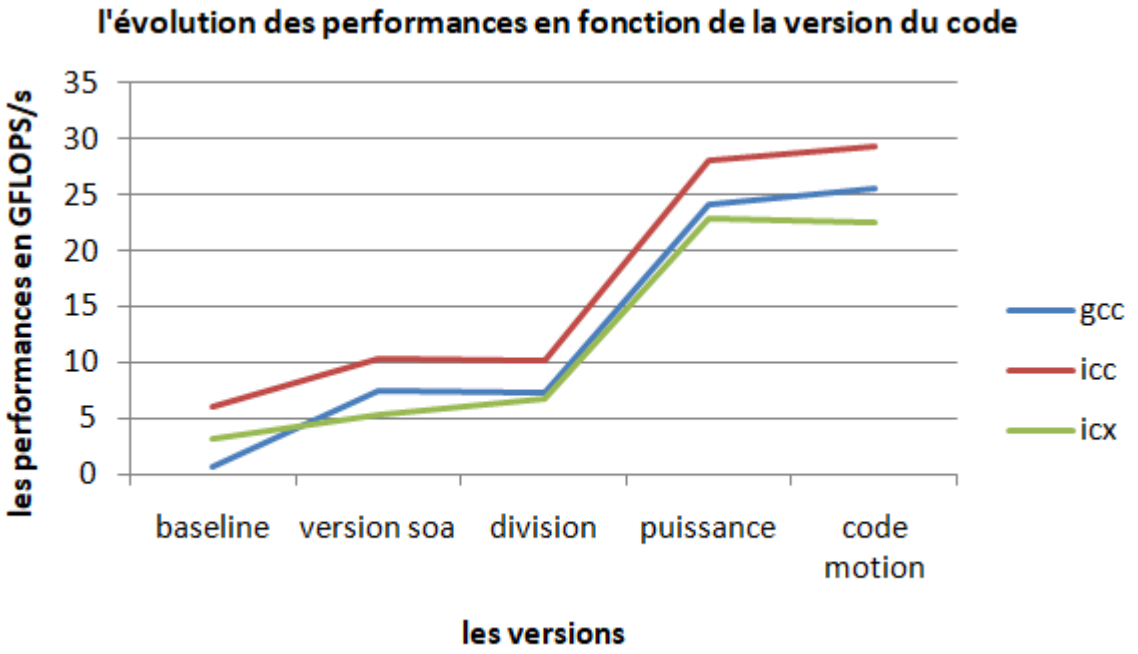
La prochaine technique que nous avons testée est la propagation des constantes. En effet, nous avons remplacé les noms de variables "softening" et "dt", dont la valeur est constante, par la constanteelle-même, respectivement 1e-20 et 0.01. Nous remarquons qu'il s'agit d'une dégradation des performances forte, par rapport à la 1ère optimisation, pour le compilateur icc, et plus ou moins légère pour les compilateurs gcc et icx.

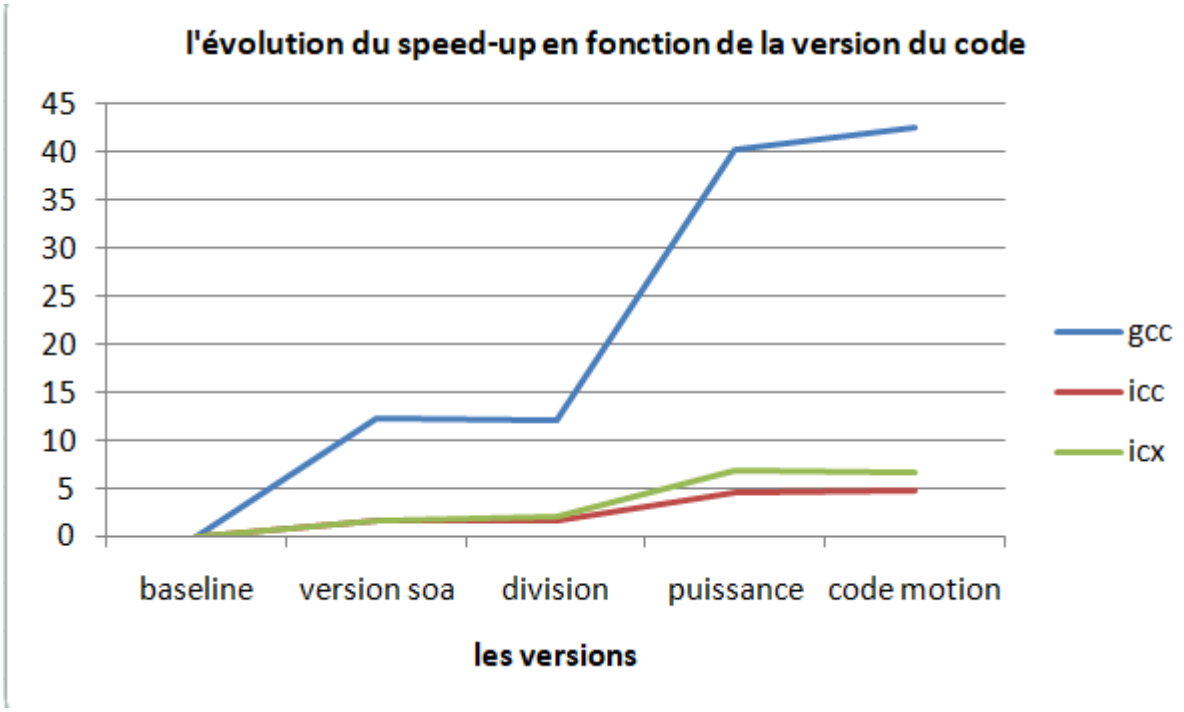
3eme optimisation : déroulage de boucle

L'optimisation suivante est le déroulage de boucles c'est-à-dire effectué plusieurs itérations de boucle en une seule fois en recopiant le corps de la boucle en plusieurs fois. Du point de vue théorique, cette technique d'optimisation est très efficace pour diminuer le nombre de branchements exécutés sachant qu'on exécute les branchements à chaque itération, baisser le nombre d'itérations permet d'en diminuer le nombre également. Nous avons observé une dégradation des performances lors du déroulage non seulement de la boucle j de la fonction "move_particles", mais aussi de la boucle i, c'est-à-dire plus nous déroulons (augmenter le nombre de pas), plus les performances moyennes baissent. Nous avons aussi effectué un déroulage directement via les options de compilation « funroll-loops », rien n'a été amélioré.

L'évolution des performances et du speed-up en fonction du compilateur

Ainsi, nous pouvons conclure que le compilateur icx est largement plus efficace en termes de performance par rapport à gcc et icx.





Conclusion

En définitive, l'intérêt de notre analyse repose sur 3 optimisations majeures ; la transformation en aos, l'amélioration de la multiplication et l'amélioration de la puissance. En effet, nous constatons que les compilateurs prennent un temps différent pour un même calcul. Afin de gagner en temps d'exécution, il est primordial de réfléchir à comment est construit notre code pour éliminer le code inutile, ou faciliter certaines instructions en se basant sur des formules mathématiques. De plus, certaines instructions ne peuvent pas être remplacées sans perdre en précision.

Bibliographie

https://godbolt.org/
https://www.agner.org/optimize/
https://perf.wiki.kernel.org/index.php/Main_Page
https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html#
https://en.wikipedia.org/wiki/AoS_and_SoA