Université de Versailles - Saint-Quentin

COMPTE RENDU

nbody_3D

BIGAND Guillaume
17 Décembre 2021

Présentation

Le but de cet exercice est d'avoir, à partir d'une version de base d'un code donné, différentes versions de celui-ci en ayant une amélioration des performances entre chaque version.
La machine utilisée pour l'analyse des performances est le kernel07 qui est une architecture knights landing. Pour plus de détails, lire les fichiers cpuinfo.txt et frequencyinfo.txt se trouvant dans le dossier res.

Version 0 :

La première version du code nbody 3D est la version de base. Il n'y a pas eu de changements. Elle permet ainsi d'avoir une base pour comparer les performances de chacune des autres versions implémentées.

```
Average performance:          0.6 +- 0.0 GFLOP/s
Average rdtsc = 13490310493
```

```
Average performance:          8.4 +- 0.0 GFLOP/s
Average rdtsc = 987155349
```

Average Performance et Rdtsc pour la version 0 de nbody3d avec les compilateurs gcc (à gauche) et icc (à droite)

On remarque que pour la version 0, le compilateur icc est nettement plus performant que le compilateur gcc. Cela peut être du au fait que icc réarrange certaines exécutions automatiquement afin de gagner en performance.

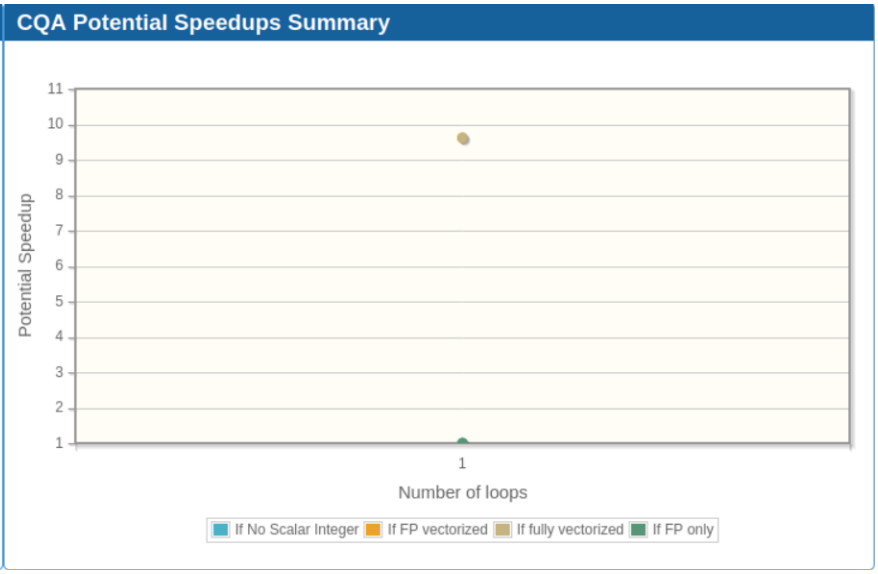
```
 0.38       vsubss      %xmm11,%xmm4,%xmm4
 1.54       cmp         %rax,%rsi
 0.39       vmulss      %xmm3,%xmm3,%xmm1
 1.52       vmovss      %xmm2,%xmm2,%xmm8
 0.38       vfmadd132ss %xmm2,%xmm13,%xmm8
 1.52       vfmadd231ss %xmm4,%xmm4,%xmm1
 0.37       vaddss      %xmm8,%xmm1,%xmm1
 1.53       vcvtss2sd   %xmm1,%xmm1,%xmm1
21.21       vsqrtsd     %xmm1,%xmm1,%xmm8
 6.52       vmulsd      %xmm8,%xmm1,%xmm1
 2.11       vcvtsd2ss   %xmm1,%xmm1,%xmm1
44.94       vdivss      %xmm1,%xmm12,%xmm1
 7.64       vfmadd231ss %xmm1,%xmm4,%xmm5
 2.19       vfmadd231ss %xmm1,%xmm3,%xmm6
 0.38       vfmadd231ss %xmm1,%xmm2,%xmm7
 1.52     ↑ jne         50
 0.01       vfmadd213ss 0xc(%rdx),%xmm0,%xmm5
 0.00       vfmadd213ss 0x10(%rdx),%xmm0,%xmm6
 0.00       vfmadd213ss 0x14(%rdx),%xmm0,%xmm7
 0.00       vmovss      %xmm5,0xc(%rdx)
 0.00       vmovss      %xmm6,0x10(%rdx)
```

```
 0.41       vmulpd      %zmm24,%zmm23,%zmm19
 0.00       vmovaps     %zmm18,%zmm23
 1.97       vcvtpd2ps   %zmm19,%ymm24
 0.44       vrsqrt28pd  {sae},%zmm18,%zmm19
 0.00       vmulpd      {rn-sae},%zmm18,%zmm19,%zmm20
 0.44       vscalefpd   0x1098c(%rip){1to8},%zmm19,%zmm19
 0.00       vfnmadd231pd {rn-sae},%zmm20,%zmm20,%zmm23
 0.34       vfmadd213pd %zmm20,%zmm19,%zmm23
            vfixupimmpd $0x70,0x1097d(%rip){1to8},%zmm18,%zmm
 0.49       vmulpd      %zmm18,%zmm23,%zmm18
 1.16       vcvtpd2ps   %zmm18,%ymm14
 1.25       vinsertf64x4 $0x1,%ymm14,%zmm24,%zmm18
 4.07       vrcp28ps    %zmm18,%zmm19
 1.57       vfmadd231ps %zmm29,%zmm19,%zmm8
 0.84       vfmadd231ps %zmm27,%zmm19,%zmm1
 0.00       vfmadd231ps %zmm25,%zmm19,%zmm0
 0.38       cmp         %r13,%rcx
          ↑ jb          3f1
 0.01       vmovss      0x80(%rsp),%xmm19
```

Perfs pour la version 0 de nbody3d avec les compilateurs gcc (à gauche) et icc (à droite)

Ici on peut voir grâce à l'outil perf que gcc passe beaucoup de temps dans l'instruction vsqrtsd qui permet de calculer une racine carrée, et vdivss qui calcule la division de deux flottants, alors qu'icc a beaucoup moins d'instructions chaudes.

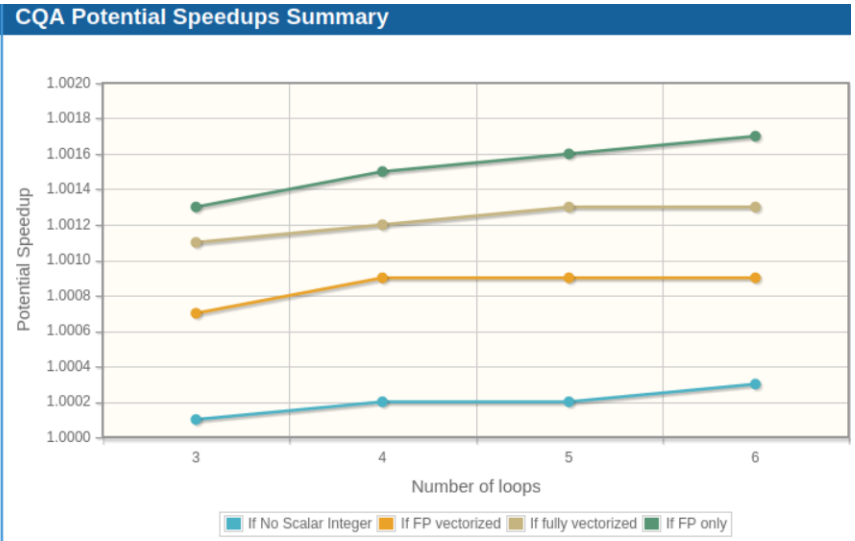
Global Metrics		
Total Time (s)		203.09
Profiled Time (s)		203.09
Time in analyzed loops (%)		100.0
Time in analyzed innermost loops (%)		100.0
Time in user code (%)		100.0
Compilation Options		Not Available
Perfect Flow Complexity		1.00
Array Access Efficiency (%)		50.0
Perfect OpenMP + MPI + Pthread		1.00
Perfect OpenMP + MPI + Pthread + Perfect Load Distribution		1.00
No Scalar Integer	Potential Speedup	1.00
	Nb Loops to get 80%	1
FP Vectorised	Potential Speedup	9.61
	Nb Loops to get 80%	1
Fully Vectorised	Potential Speedup	9.61
	Nb Loops to get 80%	1
FP Arithmetic Only	Potential Speedup	1.00
	Nb Loops to get 80%	1



Rapport Maqao de la version gcc

Le rapport Maqao nous montre que les accès aux tableaux ne sont pas très efficaces (seulement 50%) et que le programme pourrait être mieux vectorisé.

Global Metrics		
Total Time (s)		35.68
Profiled Time (s)		35.68
Time in analyzed loops (%)		47.8
Time in analyzed innermost loops (%)		2.77
Time in user code (%)		99.8
Compilation Options		Not Available
Perfect Flow Complexity		1.00
Array Access Efficiency (%)		75.0
Perfect OpenMP + MPI + Pthread		1.00
Perfect OpenMP + MPI + Pthread + Perfect Load Distribution		1.00
No Scalar Integer	Potential Speedup	1.00
	Nb Loops to get 80%	4
FP Vectorised	Potential Speedup	1.00
	Nb Loops to get 80%	2
Fully Vectorised	Potential Speedup	1.00
	Nb Loops to get 80%	1
FP Arithmetic Only	Potential Speedup	1.00
	Nb Loops to get 80%	2



Rapport Maqao de la version icc

Le rapport Maqao nous montre que les accès aux tableaux sont plutôt efficaces (75%) et que le programme est optimisé avec icc, le speedup qu'on peut avoir est faible.

Version 1 :

La version 1 consiste à transformer la structure de données. La structure actuelle est une Array of Structure (AoS). Dans cette version nous changeons la forme de cette structure en Strucutre of Array (SoA). Cela permet d'avoir une meilleure manipulation du processeur pour l'accès aux données. En effet, le processeur peut avoir accès plus rapidement à la mémoire en SoA[1].

[1]Source : `https://en.wikipedia.org/wiki/Locality_of_reference`

```
Average performance:          7.1 +- 0.0 GFLOP/s
Average rdtsc = 1209531612
```

```
Average performance:          10.4 +- 0.0 GFLOP/s
Average rdtsc = 826236644
```

Average Performance et Rdtsc pour la version 1 de nbody3d avec les compilateurs gcc (à gauche) et icc (à droite)

On remarque que pour la version 1, le compilateur icc reste plus performant que le compilateur gcc. Néanmoins gcc a donné un facteur 11 en GFLOP/s par rapport à la version 0 ce qui est beaucoup, et icc a eu un facteur 1,2.

```
 0.75    vsubps      %zmm21,%zmm2,%zmm18
 0.70    lea         0x40(%rdx),%rdx
 0.65    cmp         %rdx,%rbx
 0.65    vmulps      %zmm19,%zmm19,%zmm1
 0.72    vfmadd231ps %zmm20,%zmm20,%zmm1
 0.73    vaddps      %zmm14,%zmm1,%zmm1
 0.65    vfmadd231ps %zmm18,%zmm18,%zmm1
 1.51    vcvtps2pd   %ymm1,%zmm25
 0.66    vextracti64x4 $0x1,%zmm1,%ymm1
 1.45    vcvtps2pd   %ymm1,%zmm1
29.16    vsqrtpd     %zmm25,%zmm2
13.94    vsqrtpd     %zmm1,%zmm24
 0.66    vmulpd      %zmm25,%zmm2,%zmm2
 6.99    vmulpd      %zmm1,%zmm24,%zmm1
 1.45    vcvtpd2ps   %zmm2,%ymm2
 7.12    vcvtpd2ps   %zmm1,%ymm1
 4.10    vinsertf64x4 $0x1,%ymm1,%zmm2,%zmm2
13.36    vrcp28ps    %zmm2,%zmm2
 5.36    vfmadd231ps %zmm2,%zmm20,%zmm7
 1.35    vfmadd231ps %zmm19,%zmm2,%zmm8
 1.30    vfmadd231ps %zmm18,%zmm2,%zmm9
```

```
 0.47    vfmadd213pd %zmm1,%zmm13,%zmm2
 0.00    vmovaps     %zmm0,%zmm1
 0.52    vfnmadd231pd {rn-sae},%zmm5,%zmm5,%zmm1
 0.00    vfixupimmpd $0x70,0x11450(%rip){1to8},%zmm31,%zmm2
 0.46    vfmadd213pd %zmm5,%zmm4,%zmm1
 0.02    vmulpd      %zmm31,%zmm2,%zmm31
 0.53    vfixupimmpd $0x70,0x11439(%rip){1to8},%zmm0,%zmm1
 0.63    vcvtpd2ps   %zmm31,%ymm31
 0.10    vmulpd      %zmm0,%zmm1,%zmm0
 1.99    vcvtpd2ps   %zmm0,%ymm0
 1.39    vinsertf64x4 $0x1,%ymm0,%zmm31,%zmm31
 5.00    vrcp28ps    %zmm31,%zmm31
 1.95    vfmadd231ps %zmm23,%zmm31,%zmm12
 1.03    vfmadd231ps %zmm16,%zmm31,%zmm7
 0.50    vfmadd231ps %zmm24,%zmm31,%zmm8
 0.01    cmp         %r9,%rcx
 0.49  ↑ jb          922
```

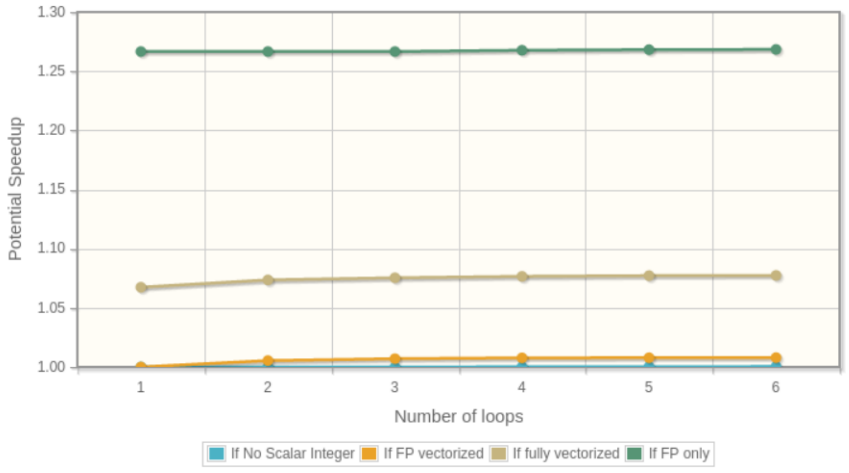
Perfs pour la version 1 de nbody3d avec les compilateurs gcc (à gauche) et icc (à droite)

Ici on peut voir grâce à l'outil perf que gcc passe beaucoup de temps dans l'instruction vsqrtpd qui permet de calculer une racine carrée, et vmulpd qui calcule la multiplication de deux flottants. Quant à icc, il a moins d'instructions chaudes.

Global Metrics

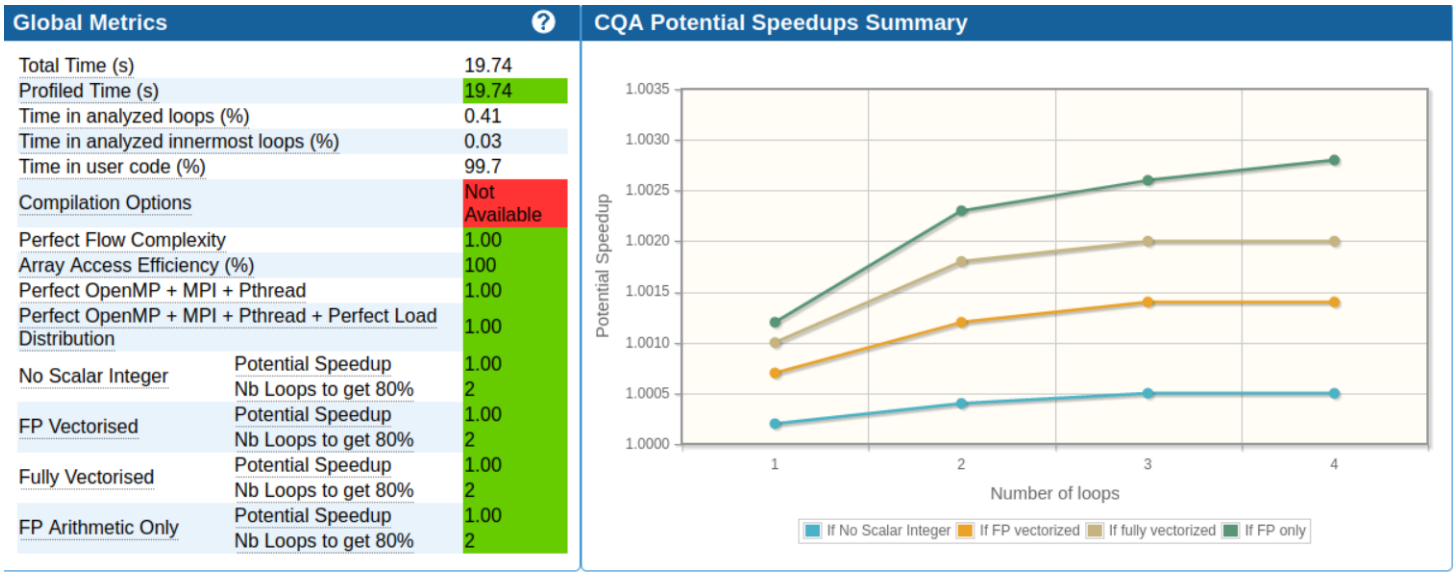
Total Time (s)		33.22
Profiled Time (s)		33.22
Time in analyzed loops (%)		99.9
Time in analyzed innermost loops (%)		99.8
Time in user code (%)		99.9
Compilation Options		Not Available
Perfect Flow Complexity		1.00
Array Access Efficiency (%)		50.0
Perfect OpenMP + MPI + Pthread		1.00
Perfect OpenMP + MPI + Pthread + Perfect Load Distribution		1.00
No Scalar Integer	Potential Speedup	1.00
	Nb Loops to get 80%	3
FP Vectorised	Potential Speedup	1.01
	Nb Loops to get 80%	2
Fully Vectorised	Potential Speedup	1.08
	Nb Loops to get 80%	1
FP Arithmetic Only	Potential Speedup	1.27
	Nb Loops to get 80%	1

CQA Potential Speedups Summary



Rapport Maqao de la version gcc

Le rapport Maqao nous montre que les accès aux tableaux sont les mêmes que la version 0 mais il y a une bien meilleure vectorisation. Les boucles peuvent encore être améliorées et le calcul des instructions arithmétiques aussi.



Rapport Maqao de la version icc

Le rapport Maqao nous montre que les tableaux sont parfaitement accédés.

Version 2 :

Dans cette version, je me suis concentré sur l'architecture des boucles. Différents changements ont pu alors être fait. Certaines instructions utilisent des variables pouvant être calculées dans une boucle plus externe, et aussi certaines instructions étaient dans une deuxième boucle externe alors que ces instructions peuvent être calculées dans une seule boucle. Ces changements ont donc été fait.

```
Average performance:          8.7 +- 0.0 GFLOP/s
Average rdtsc = 1208974925
```

```
Average performance:          13.3 +- 0.0 GFLOP/s
Average rdtsc = 804393798
```

Average Performance et Rdtsc pour la version 2 de nbody3d avec les compilateurs gcc (à gauche) et icc (à droite)

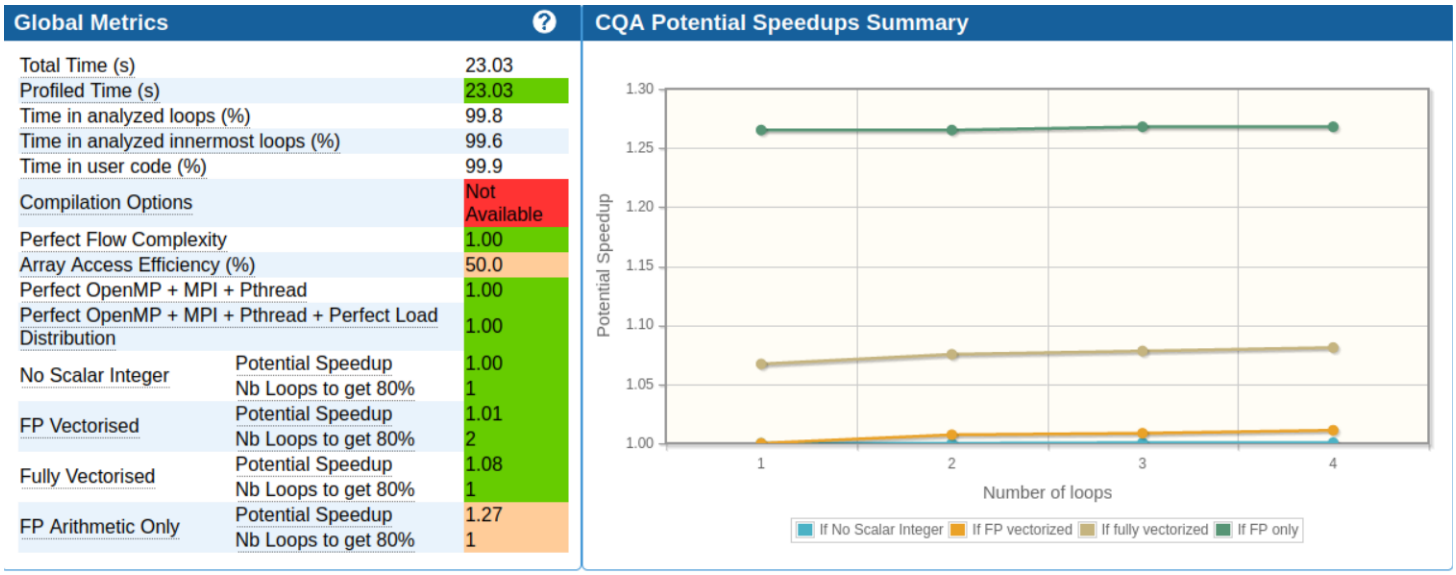
On remarque que pour la version 2, on voit que les deux compilateurs ont gagné un facteur $\simeq 1.25$ pour les deux versions par rapport à la version 1.

```
 1.24    vmovups      (%r11,%rdx,1),%zmm2
 0.79    vsubps       %zmm21,%zmm2,%zmm18
 0.64    lea          0x40(%rdx),%rdx
 0.64    cmp          %rdx,%rbx
 0.65    vmulps       %zmm19,%zmm19,%zmm1
 0.71    vfmadd231ps  %zmm20,%zmm20,%zmm1
 0.67    vaddps       %zmm14,%zmm1,%zmm1
 0.68    vfmadd231ps  %zmm18,%zmm18,%zmm1
 1.46    vcvtps2pd    %ymm1,%zmm2
 0.70    vextracti64x4 $0x1,%zmm1,%ymm1
 1.38    vcvtps2pd    %ymm1,%zmm1
29.67    vsqrtpd      %zmm2,%zmm25
13.82    vsqrtpd      %zmm1,%zmm24
 0.63    vmulpd       %zmm25,%zmm2,%zmm2
 7.05    vmulpd       %zmm24,%zmm1,%zmm1
 1.42    vcvtpd2ps    %zmm2,%ymm2
 7.10    vcvtpd2ps    %zmm1,%ymm1
 3.99    vinsertf64x4 $0x1,%ymm1,%zmm2,%zmm1
13.39    vrcp28ps     %zmm1,%zmm1
 5.34    vfmadd231ps  %zmm20,%zmm1,%zmm7
 1.36    vfmadd231ps  %zmm19,%zmm1,%zmm8
```

```
 1.03    vfixupimmpd  $0x70,0xf01b(%rip){1to8},%zmm21,%zmm30
         vmulpd       {rn-sae},%zmm10,%zmm28,%zmm31
 1.02    vscalefpd    0xf003(%rip){1to8},%zmm28,%zmm28
 3.52    vmulpd       %zmm21,%zmm30,%zmm21
 1.05    vfnmadd231pd {rn-sae},%zmm31,%zmm31,%zmm29
 4.84    vcvtpd2ps    %zmm21,%ymm21
 0.00    vfmadd213pd  %zmm31,%zmm28,%zmm29
 1.03    vfixupimmpd  $0x70,0xefe8(%rip){1to8},%zmm10,%zmm29
 0.81    vmulpd       %zmm10,%zmm29,%zmm28
 4.47    vcvtpd2ps    %zmm28,%ymm11
 2.98    vinsertf64x4 $0x1,%ymm11,%zmm21,%zmm21
10.42    vrcp28ps     %zmm21,%zmm21
 4.22    vfmadd231ps  %zmm22,%zmm21,%zmm0
 1.01    vfmadd231ps  %zmm23,%zmm21,%zmm1
 1.04    vfmadd231ps  %zmm24,%zmm21,%zmm3
 0.00    cmp          %r9,%rdx
 1.00    jb           e19
 0.00 f04: lea        0x1(%r9),%rax
```

Perfs pour la version 2 de nbody3d avec les compilateurs gcc (à gauche) et icc (à droite)

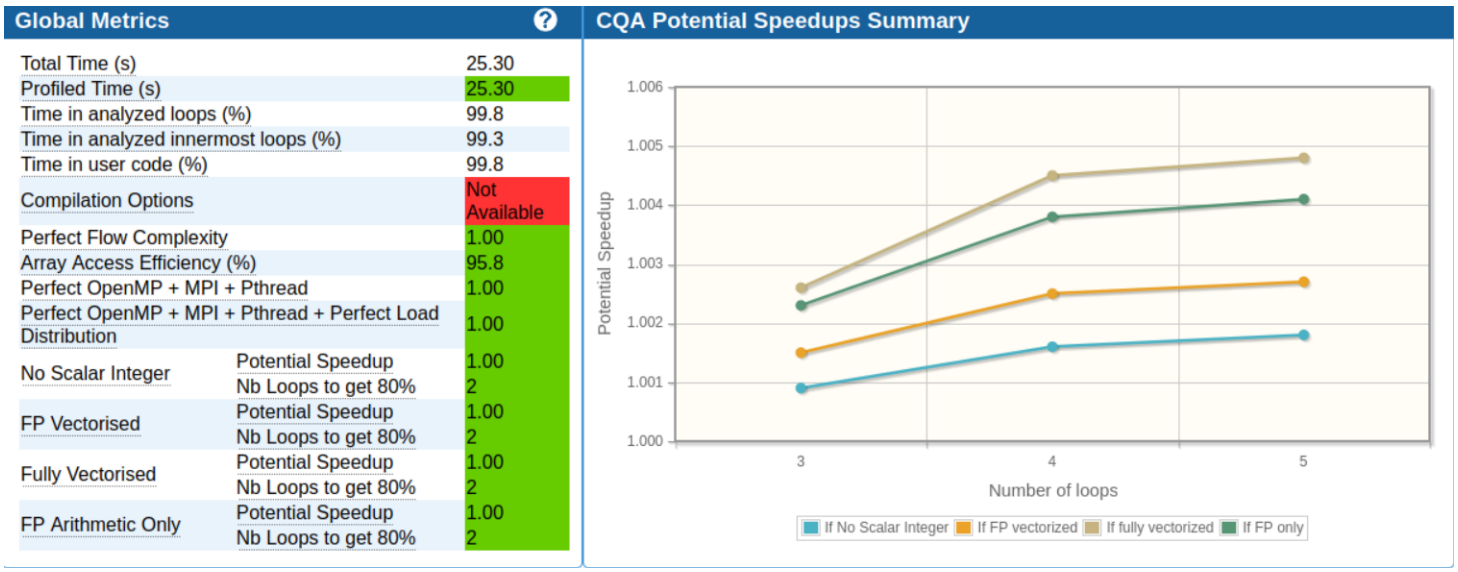
L'outil perf nous donne les mêmes informations que la verion 1, à savoir ce sont les mêmes instructions chaudes qui sont visées.



Global Metrics		
Total Time (s)		23.03
Profiled Time (s)		23.03
Time in analyzed loops (%)		99.8
Time in analyzed innermost loops (%)		99.6
Time in user code (%)		99.9
Compilation Options		Not Available
Perfect Flow Complexity		1.00
Array Access Efficiency (%)		50.0
Perfect OpenMP + MPI + Pthread		1.00
Perfect OpenMP + MPI + Pthread + Perfect Load Distribution		1.00
No Scalar Integer	Potential Speedup	1.00
	Nb Loops to get 80%	1
FP Vectorised	Potential Speedup	1.01
	Nb Loops to get 80%	2
Fully Vectorised	Potential Speedup	1.08
	Nb Loops to get 80%	1
FP Arithmetic Only	Potential Speedup	1.27
	Nb Loops to get 80%	1

Rapport Maqao de la version gcc

Le rapport Maqao montre que les boucles ont été légèrement améliorées et que le calcul des instructions arithmétiques peut être amélioré.



Global Metrics		
Total Time (s)		25.30
Profiled Time (s)		25.30
Time in analyzed loops (%)		99.8
Time in analyzed innermost loops (%)		99.3
Time in user code (%)		99.8
Compilation Options		Not Available
Perfect Flow Complexity		1.00
Array Access Efficiency (%)		95.8
Perfect OpenMP + MPI + Pthread		1.00
Perfect OpenMP + MPI + Pthread + Perfect Load Distribution		1.00
No Scalar Integer	Potential Speedup	1.00
	Nb Loops to get 80%	2
FP Vectorised	Potential Speedup	1.00
	Nb Loops to get 80%	2
Fully Vectorised	Potential Speedup	1.00
	Nb Loops to get 80%	2
FP Arithmetic Only	Potential Speedup	1.00
	Nb Loops to get 80%	2

Rapport Maqao de la version icc

Le rapport Maqao nous montre que les accès aux tableaux ont baissés mais restent plus que corrects.

Version 3 :

Pour cette version, je me suis concentré sur l'optimisation des opérations utilisées, notamment la racine carrée. L'opération pow(x,y) est une fonction qui renvoie x à la puissance de y. Cependant cette opération est très coûteuse. Ici on avait :

```
a = pow(X,3/2);
```

Je suis donc passé par la fonction sqrtf(X) qui renvoie la racine carrée du nombre x. Cette fonction est moins coûteuse, et en réécrivant le calcul de la bonne façon afin d'avoir le même résultat, on gagne en GigaFlops/s. On est obligé d'utiliser sqrtf et non sqrt car nous avons besoin d'un réel, et la fonction sqrt renvoie un entier.

```
Average performance:          31.7 +- 0.0 GFLOP/s
Average rdtsc = 354335917
```

```
Average performance:          33.2 +- 0.2 GFLOP/s
Average rdtsc = 334192782
```

Average Performance et Rdtsc pour la version 3 de nbody3d avec les compilateurs gcc (à gauche) et icc (à droite)

On remarque que pour la version 3, on voit que les deux compilateurs ont respectivement gagné un facteur $\simeq$ 3,6 et 3 pour les deux versions par rapport à la version 2. On remarque que entre cette version et la version 0, gcc a un facteur $\simeq$ 53 et icc $\simeq$ 4.

```
 1.17        vmovups       (%r11,%rdx,1),%zmm1
 4.69        vsubps        %zmm21,%zmm1,%zmm17
             lea           0x40(%rdx),%rdx
 4.80        cmp           %rdx,%rbx
 0.01        vmulps        %zmm18,%zmm18,%zmm1
 4.63        vfmadd231ps   %zmm19,%zmm19,%zmm1
 0.56        vaddps        %zmm14,%zmm1,%zmm1
 5.01        vfmadd231ps   %zmm17,%zmm17,%zmm1
 2.38        vrsqrt28ps    %zmm1,%zmm20
11.21        vrcp28ps      %zmm20,%zmm20
 7.81        vmulps        %zmm20,%zmm1,%zmm1
20.20        vrcp28ps      %zmm1,%zmm1
12.88        vfmadd231ps   %zmm19,%zmm1,%zmm7
 4.64        vfmadd231ps   %zmm1,%zmm18,%zmm8
 1.96        vfmadd231ps   %zmm17,%zmm1,%zmm9
 4.69      ↑ jne           1a0
 0.06        vextracti64x4 $0x1,%zmm9,%ymm1
 0.02        vaddps        %ymm9,%ymm1,%ymm9
 0.04        mov           -0x20(%rsp),%rdx
```

```
 3.17        vaddps        (%r10,%rbx,4),%zmm22,%zmm30
             vfmadd231ps   %zmm29,%zmm29,%zmm24
 2.55        vaddps        (%r11,%rbx,4),%zmm21,%zmm10
             add           $0x10,%rbx
 2.37        vfmadd231ps   %zmm30,%zmm30,%zmm24
 0.13        vfmadd231ps   %zmm10,%zmm10,%zmm24
 2.79        vrsqrt28ps    {sae},%zmm24,%zmm25
 1.74        vrcp28ps      {sae},%zmm25,%zmm26
 4.53        vmulps        %zmm26,%zmm26,%zmm27
 2.12        vmulps        %zmm27,%zmm26,%zmm28
12.39        vrcp28ps      %zmm28,%zmm31
 6.85        vfmadd231ps   %zmm29,%zmm31,%zmm0
 4.86        vfmadd231ps   %zmm30,%zmm31,%zmm3
 0.61        vfmadd231ps   %zmm10,%zmm31,%zmm1
 2.47        cmp           %r9,%rbx
           ↑ jb            560
 0.02  5c6:  lea           0x1(%r9),%rdx
 0.01        cmp           %r15,%rdx
 0.01      ↓ ja            6e8
```

Perfs pour la version 3 de nbody3d avec les compilateurs gcc (à gauche) et icc (à droite)

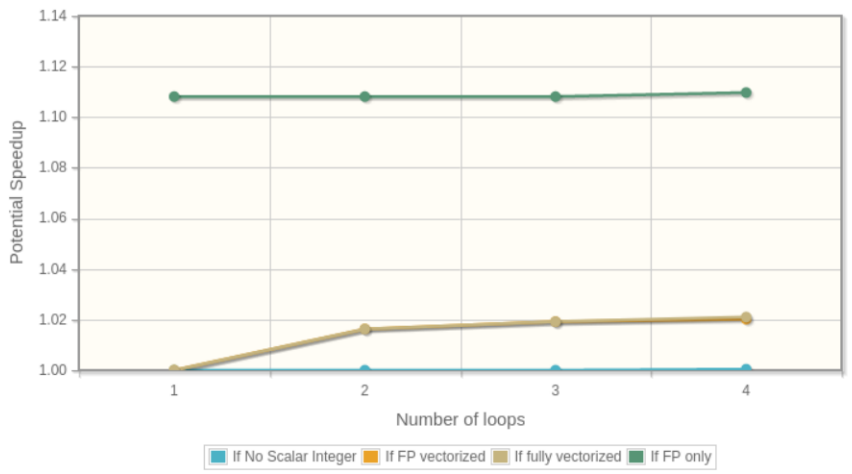
L'outil perf nous indique que pour la verion 3, les principales instructions chaudes pour gcc et icc sont vrcp28ps[2] et des instructions arithétiques, ce qui est aussi le cas pour icc mais en moins conséquent.

[2]Pour plus d'informations sur l'instruction vrcp28ps : `https://www.felixcloutier.com/x86/vrcp28ps`

Global Metrics

Total Time (s)		4.97
Profiled Time (s)		4.97
Time in analyzed loops (%)		99.8
Time in analyzed innermost loops (%)		99.6
Time in user code (%)		99.8
Compilation Options		Not Available
Perfect Flow Complexity		1.00
Array Access Efficiency (%)		50.0
Perfect OpenMP + MPI + Pthread		1.00
Perfect OpenMP + MPI + Pthread + Perfect Load Distribution		1.00
No Scalar Integer	Potential Speedup	1.00
	Nb Loops to get 80%	1
FP Vectorised	Potential Speedup	1.02
	Nb Loops to get 80%	1
Fully Vectorised	Potential Speedup	1.02
	Nb Loops to get 80%	2
FP Arithmetic Only	Potential Speedup	1.11
	Nb Loops to get 80%	1

CQA Potential Speedups Summary



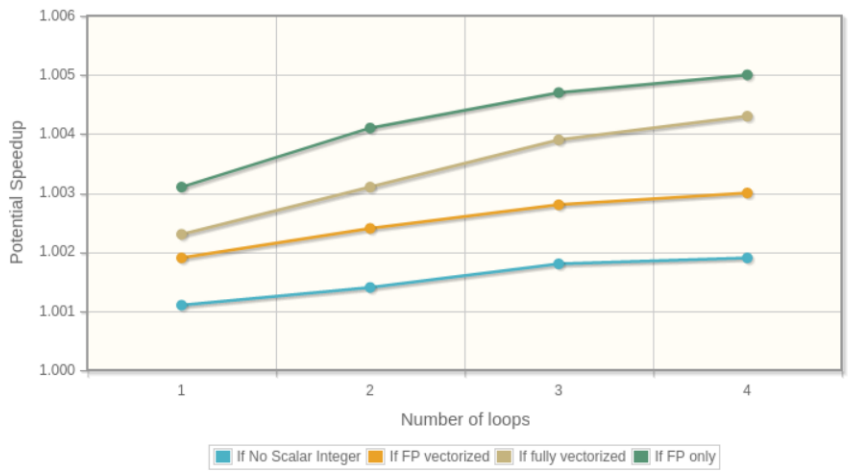
Rapport Maqao de la version gcc

Le rapport Maqao montre que les instructions arithmétiques ont été améliorées par rapport aux versions antérieures.

Global Metrics

Total Time (s)		13.20
Profiled Time (s)		13.20
Time in analyzed loops (%)		0.72
Time in analyzed innermost loops (%)		0.12
Time in user code (%)		99.5
Compilation Options		Not Available
Perfect Flow Complexity		1.00
Array Access Efficiency (%)		95.8
Perfect OpenMP + MPI + Pthread		1.00
Perfect OpenMP + MPI + Pthread + Perfect Load Distribution		1.00
No Scalar Integer	Potential Speedup	1.00
	Nb Loops to get 80%	3
FP Vectorised	Potential Speedup	1.00
	Nb Loops to get 80%	2
Fully Vectorised	Potential Speedup	1.00
	Nb Loops to get 80%	3
FP Arithmetic Only	Potential Speedup	1.00
	Nb Loops to get 80%	2

CQA Potential Speedups Summary



Rapport Maqao de la version icc

Le rapport Maqao indique qu'il y a plus de boucles vectorisables.

Version 4 :

Dans cette dernière version j'ai voulu me concentrer sur les différents flags dans le makefile afin de gagner des performances. Après des tests peu concluant, j'ai décidé de changer uniquement le flag mavx2 en mavx512f (pour gcc uniquement) pour passer au jeu d'instruction avx512, car je stagnais voire perdait en GFLOP/s. J'ai remodifié la fonction sqrtf() et j'ai transformé les malloc() en aligned_alloc(). Le fait de passer en aligned_alloc() permet d'aligner les données en mémoire des tableaux utilisés dans la structure, et permet donc une meilleure vectorisation.

```
Average performance:          32.8 +- 0.1 GFLOP/s
Average rdtsc = 353788470
```

```
Average performance:          48.2 +- 0.0 GFLOP/s
Average rdtsc = 238890580
```

Average Performance et Rdtsc pour la version 4 de nbody3d avec les compilateurs gcc (à gauche) et icc (à droite)

On remarque que pour la version 4, on voit que le compilateur gcc a légèrement gagné en performance par rapport à la version 3, tandis qu'icc a gagné un facteur passe à 48 GFLOP/s ce qui donne pour la performance un gain de facteur $\simeq$ 1,45 par rapport à la version 3.

```
 1.00        vmovups       (%r11,%rdx,1),%zmm1
 4.74        vsubps        %zmm21,%zmm1,%zmm17
 0.00        lea           0x40(%rdx),%rdx
 4.82        cmp           %rdx,%rbx
             vmulps        %zmm18,%zmm18,%zmm1
 4.61        vfmadd231ps   %zmm19,%zmm19,%zmm1
 0.53        vaddps        %zmm14,%zmm1,%zmm1
 4.66        vfmadd231ps   %zmm17,%zmm17,%zmm1
 2.38        vrsqrt28ps    %zmm1,%zmm20
11.01        vrcp28ps      %zmm20,%zmm20
 8.00        vmulps        %zmm20,%zmm1,%zmm1
20.79        vrcp28ps      %zmm1,%zmm1
12.78        vfmadd231ps   %zmm1,%zmm19,%zmm7
 4.48        vfmadd231ps   %zmm1,%zmm18,%zmm8
 1.98        vfmadd231ps   %zmm1,%zmm17,%zmm9
 4.42      ↑ jne           1a0
 0.04        vextracti64x4 $0x1,%zmm9,%ymm1
 0.03        vaddps        %ymm9,%ymm1,%ymm9
 0.05        mov           -0x20(%rsp),%rdx
```

```
 1.24  530:  vaddps        (%r8,%rbx,4),%zmm23,%zmm27
 3.52        vmovaps       %zmm18,%zmm24
 0.94        vaddps        (%r10,%rbx,4),%zmm22,%zmm28
 3.42        vfmadd231ps   %zmm27,%zmm27,%zmm24
 0.01        vaddps        (%r11,%rbx,4),%zmm16,%zmm30
 3.24        add           $0x10,%rbx
 0.09        vfmadd231ps   %zmm28,%zmm28,%zmm24
 3.66        vfmadd231ps   %zmm30,%zmm30,%zmm24
 2.94        vrsqrt28ps    %zmm24,%zmm25
 4.82        vrcp28ps      %zmm24,%zmm26
 3.59        vmulps        %zmm26,%zmm25,%zmm29
13.81        vfmadd231ps   %zmm27,%zmm29,%zmm0
 0.41        vfmadd231ps   %zmm28,%zmm29,%zmm1
 4.98        vfmadd231ps   %zmm30,%zmm29,%zmm2
             cmp           %r9,%rbx
 3.34      ↑ jb            530
 0.02  58a:  lea           0x1(%r9),%rdx
 0.01        cmp           %r15,%rdx
```

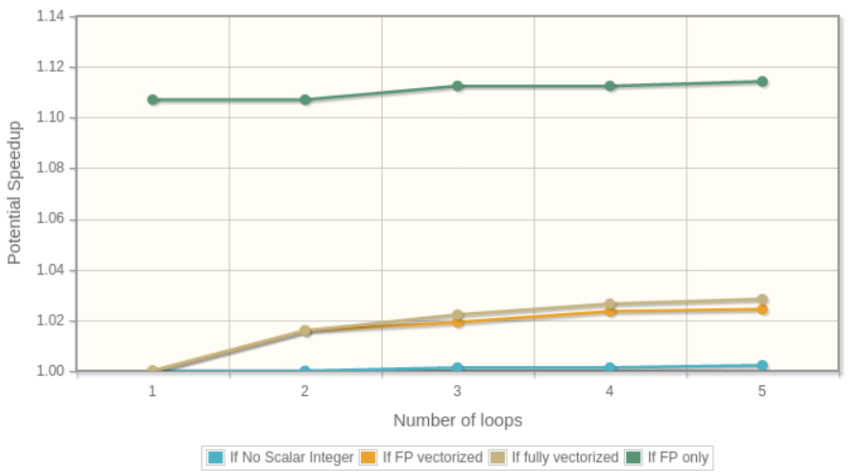
Perfs pour la version 4 de nbody3d avec les compilateurs gcc (à gauche) et icc (à droite)

L'outil perf nous indique que pour la verion 4, les principales instructions chaudes pour gcc et icc sont les mêmes que la version 3.

Global Metrics

Total Time (s)		5.69
Profiled Time (s)		5.69
Time in analyzed loops (%)		99.7
Time in analyzed innermost loops (%)		99.0
Time in user code (%)		99.5
Compilation Options		Not Available
Perfect Flow Complexity		1.00
Array Access Efficiency (%)		50.0
Perfect OpenMP + MPI + Pthread		1.00
Perfect OpenMP + MPI + Pthread + Perfect Load Distribution		1.00
No Scalar Integer	Potential Speedup	1.00
	Nb Loops to get 80%	2
FP Vectorised	Potential Speedup	1.02
	Nb Loops to get 80%	2
Fully Vectorised	Potential Speedup	1.03
	Nb Loops to get 80%	3
FP Arithmetic Only	Potential Speedup	1.11
	Nb Loops to get 80%	1

CQA Potential Speedups Summary



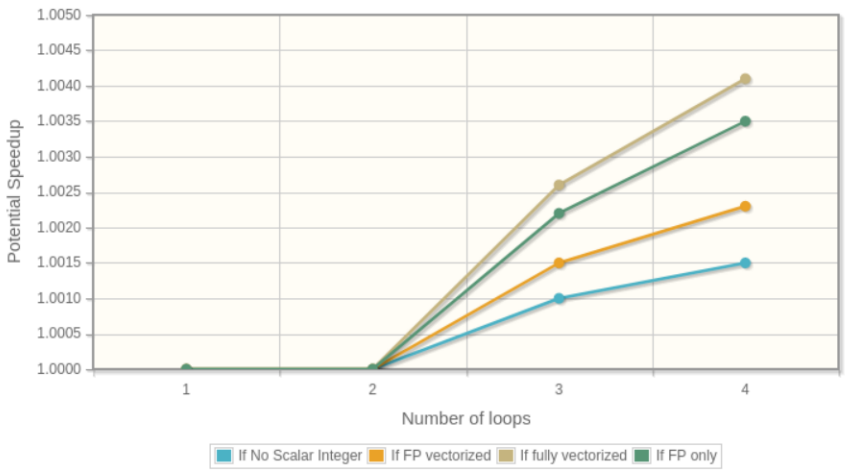
Rapport Maqao de la version gcc

Le rapport Maqao montre les mêmes résultats que la version3 si ce n'est une légère amélioration de vectorisation.

Global Metrics

Total Time (s)		8.12
Profiled Time (s)		8.12
Time in analyzed loops (%)		99.2
Time in analyzed innermost loops (%)		98.7
Time in user code (%)		99.2
Compilation Options		Not Available
Perfect Flow Complexity		1.00
Array Access Efficiency (%)		100
Perfect OpenMP + MPI + Pthread		1.00
Perfect OpenMP + MPI + Pthread + Perfect Load Distribution		1.00
No Scalar Integer	Potential Speedup	1.00
	Nb Loops to get 80%	2
FP Vectorised	Potential Speedup	1.00
	Nb Loops to get 80%	2
Fully Vectorised	Potential Speedup	1.00
	Nb Loops to get 80%	2
FP Arithmetic Only	Potential Speedup	1.00
	Nb Loops to get 80%	2

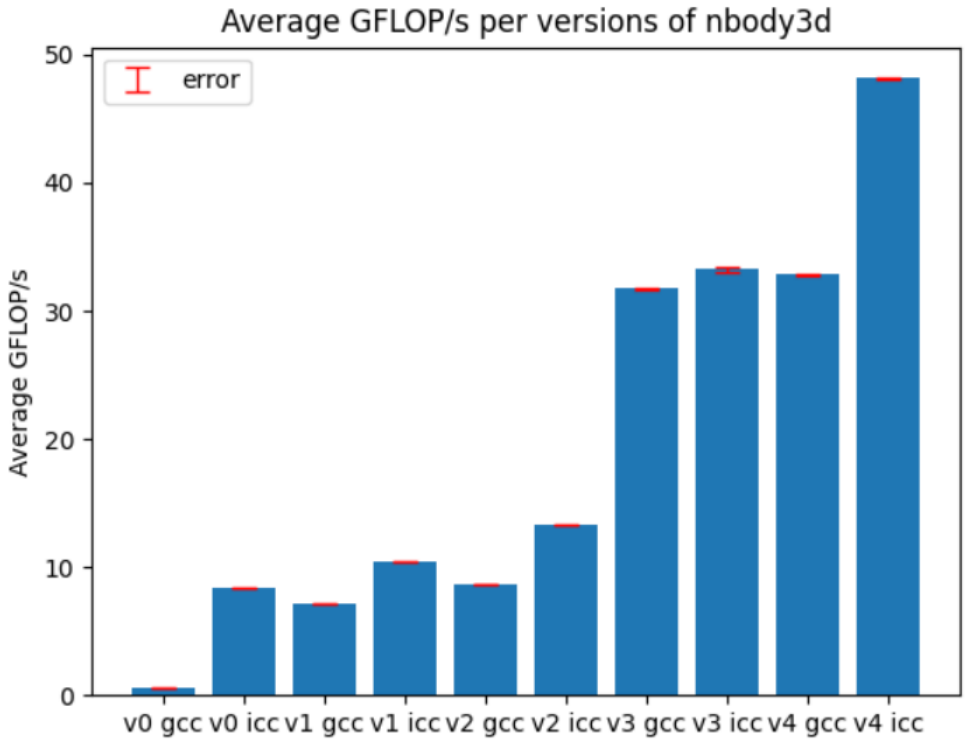
CQA Potential Speedups Summary



Rapport Maqao de la version icc

Le rapport Maqao nous montre que les tableaux sont parfaitement accédés et qu'il y a plus de vectorisations dans les boucles ainsi que moins de calcul arithmétiques dans les boucles.

Conclusion :



On peut conclure que le compilateur icc donne de meilleures performances en général par rapport à gcc, mais que le facteur gain de performance pour gcc est bien supérieur entre la version 0 et 4 ($\simeq$ 54.8) par rapport à icc ($\simeq$ 5.8).