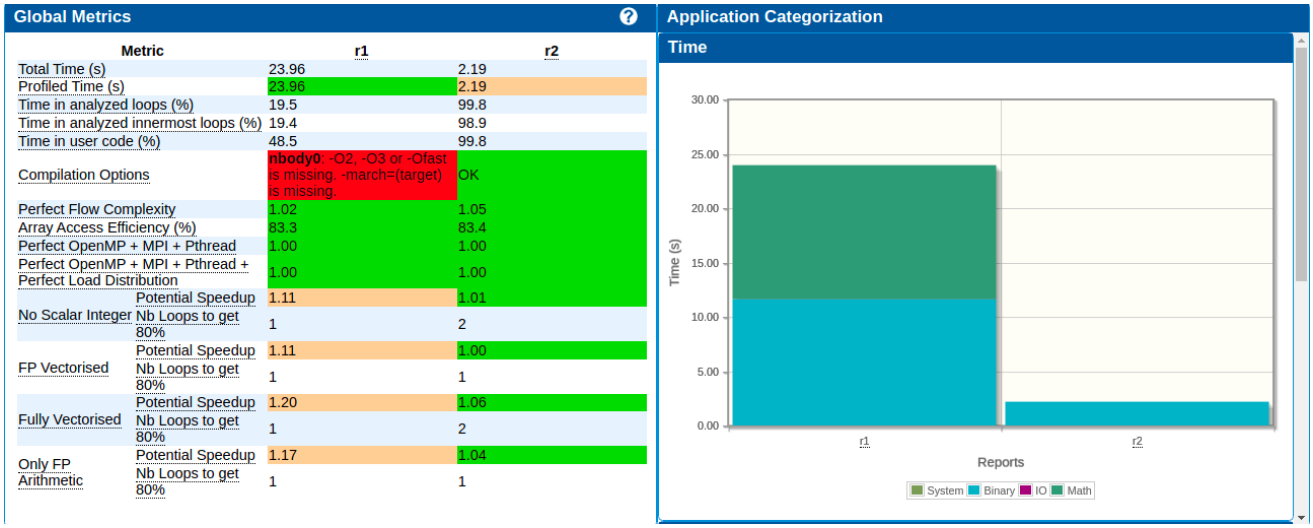
1 Critique

1.1 Les points fort de MAQAO

Les flags de compilation

MAQAO nous a beaucoup aidée lors de choix des options de compilations. Voici un exemple où on a juste changer les flags de compilations, on gardent le même code et la même machine.



Global Metrics

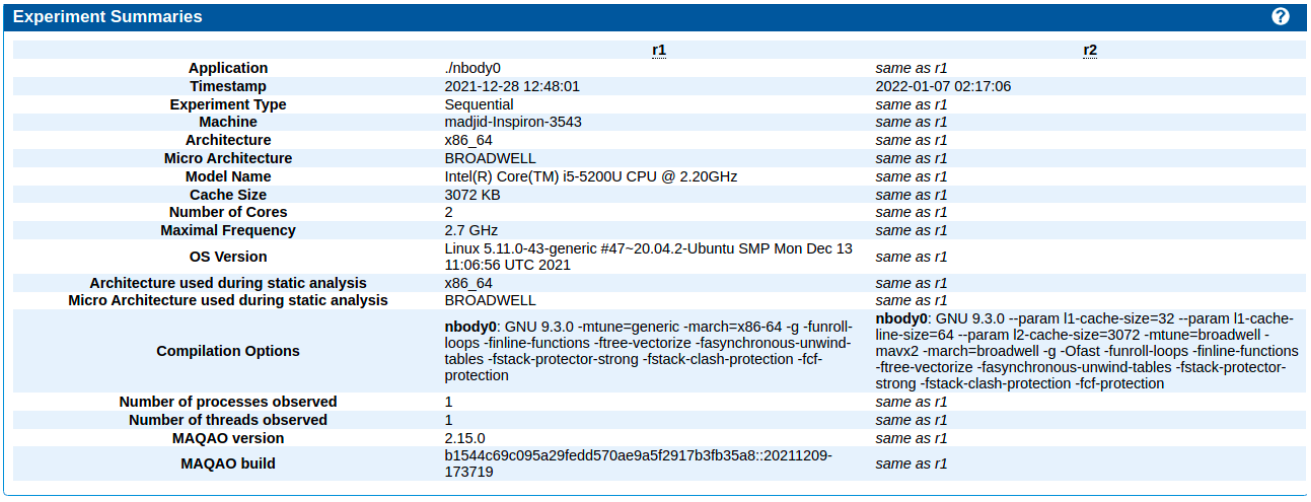
Metric		r1	r2
Total Time (s)		23.96	2.19
Profiled Time (s)		23.96	2.19
Time in analyzed loops (%)		19.5	99.8
Time in analyzed innermost loops (%)		19.4	98.9
Time in user code (%)		48.5	99.8
Compilation Options		**nbody0**: -O2, -O3 or -Ofast is missing. -march=(target) is missing.	OK
Perfect Flow Complexity		1.02	1.05
Array Access Efficiency (%)		83.3	83.4
Perfect OpenMP + MPI + Pthread		1.00	1.00
Perfect OpenMP + MPI + Pthread + Perfect Load Distribution		1.00	1.00
No Scalar Integer	Potential Speedup	1.11	1.01
	Nb Loops to get 80%	1	2
FP Vectorised	Potential Speedup	1.11	1.00
	Nb Loops to get 80%	1	1
Fully Vectorised	Potential Speedup	1.20	1.06
	Nb Loops to get 80%	1	2
Only FP Arithmetic	Potential Speedup	1.17	1.04
	Nb Loops to get 80%	1	1

Application Categorization

Time

FIGURE 1 – Nbody2D 1er version modification seulement des flags de compilation

On a ganger *10 la vitesse d'executions par rapport sans les bons flags de compilations.



Experiment Summaries

	r1	r2
Application	./nbody0	*same as r1*
Timestamp	2021-12-28 12:48:01	2022-01-07 02:17:06
Experiment Type	Sequential	*same as r1*
Machine	madjid-Inspiron-3543	*same as r1*
Architecture	x86_64	*same as r1*
Micro Architecture	BROADWELL	*same as r1*
Model Name	Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz	*same as r1*
Cache Size	3072 KB	*same as r1*
Number of Cores	2	*same as r1*
Maximal Frequency	2.7 GHz	*same as r1*
OS Version	Linux 5.11.0-43-generic #47~20.04.2-Ubuntu SMP Mon Dec 13 11:06:56 UTC 2021	*same as r1*
Architecture used during static analysis	x86_64	*same as r1*
Micro Architecture used during static analysis	BROADWELL	*same as r1*
Compilation Options	**nbody0**: GNU 9.3.0 -mtune=generic -march=x86-64 -g -funroll-loops -finline-functions -ftree-vectorize -fasynchronous-unwind-tables -fstack-protector-strong -fstack-clash-protection -fcf-protection	**nbody0**: GNU 9.3.0 --param l1-cache-size=32 --param l1-cache-line-size=64 --param l2-cache-size=3072 -mtune=broadwell -mavx2 -march=broadwell -g -Ofast -funroll-loops -finline-functions -ftree-vectorize -fasynchronous-unwind-tables -fstack-protector-strong -fstack-clash-protection -fcf-protection
Number of processes observed	1	*same as r1*
Number of threads observed	1	*same as r1*
MAQAO version	2.15.0	*same as r1*
MAQAO build	b1544c69c095a29fedd570ae9a5f2917b3fb35a8::20211209-173719	*same as r1*

FIGURE 2 – Nbody2D Information sur la machine

Chagement de l'orde des boucles

Voici une boucle de multiplication de matrices

```
1  void mul_matrix(int **matrix_a, int **matrix_b, int **matrix_ab, int l)
       {
2       for(int i = 0; i < l; i++){
3           for(int j = 0; j < l; j++){
4               matrix_ab[i][j] = 0;
5               for(int k = 0; k < l; k++){
6                   matrix_ab[i][j] += matrix_a[i][k] * matrix_b[k][j];
7               } } } }
```

Listing 1 – avant permutation

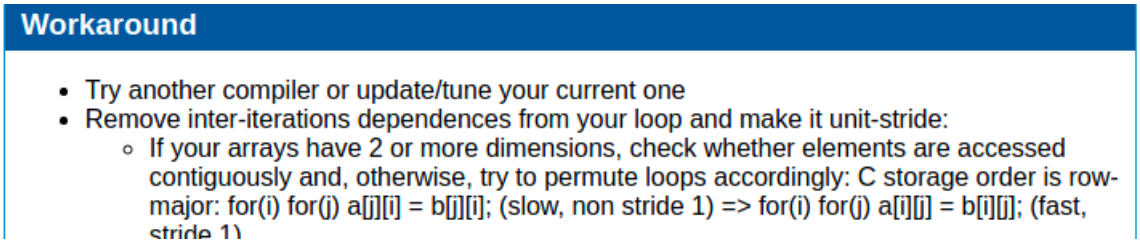


FIGURE 3 – Suggestion de MAQAO de permuter les boucles

```
1  void mul_matrix(int **matrix_a, int **matrix_b, int **matrix_ab, int l)
       {
2       for(int i = 0; i < l; i++){
3       for(int k = 0; k < l; k++){
4           for(int j = 0; j < l; j++){
5               matrix_ab[i][j] = 0;
6
7                   matrix_ab[i][j] += matrix_a[i][k] * matrix_b[k][j];
8       }}}}
```

Listing 2 – après permutation

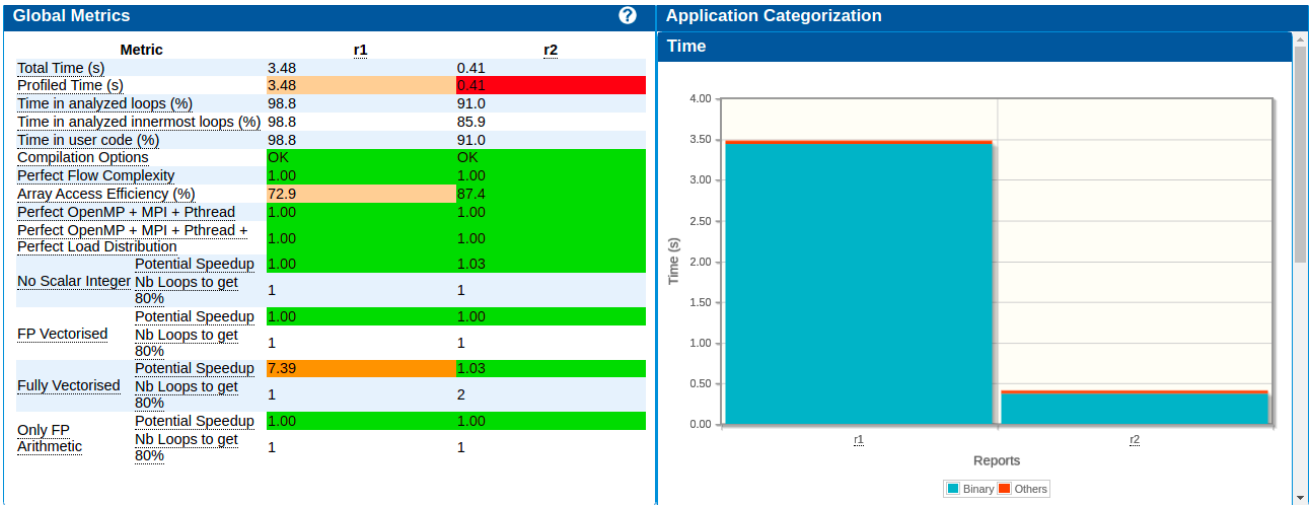


FIGURE 4 – Comparaison entre avant et après permutations des boucles

Remarque : On remarque que on changent seulement l'ordre des boucles on a ganger on temps d'execution et on vectorisation.

Les autres points fort

MAQAO nous a permet d'optimiser le programme de Nbody3D qu'on illustrer où début de notre rapport avec tout ces indications dans les different onglet chagement de structeur de AoS à SoA, l'ajout des bons flags de compilation et les indications sur la vectorisations.

1.2 Les points negative de MAQAO

Malgré ses point fort MAQAO nous a pas aidé dans tous ses suggestions meme des fois ils nous a induit en erreur ou ils nous dits de faire des modifications que on a déjà fait voici quelque exemple remarqué ou cours de notre manipulation de MAQAO.

MAQAO et CQA

Ne ce base pas sur le code source MAQAO travail execulisevemment aux niveaux binaire il est obligé un peu de deviner a quoi ressemble le code source et à quoi ressemble la compilation ils savent pas si on a changé le code source tous ce qu'ils regarde c'est le code asssembleur qui resulte.

On peut avoir des cas ou il contine de dire essayé cette solution ou bien modifier cette boucle alor que on base la modification à était déjà faite.

Exemple : Si on change une partie dans le code source et que celle ci ne change pas son code assembleur CQA vas dire rien n'a changé.Une limite de MAQAO.

1. **Cas de possix_memaligne :** Maqao nous a demandé d'aligner notre mémoire car elle n'était pas alignée en nous proposant de l'aligne avec la fonctions posix_memealign, malgré qu'on a fait le nécessaire pour ça, on a toujours cette suggestion au niveau de la dernière analyse.

```
double  * p_x = NULL;
double  * p_y = NULL;
double  * p_z = NULL;
double  * p_vx = NULL;
double  * p_vy= NULL;
double  * p_vz = NULL;
int l = 0;
l +=posix_memalign ((void **)  &p_x, 32, n* sizeof(particle_t));
l +=posix_memalign ((void **)  &p_y, 32, n* sizeof(particle_t));
l +=posix_memalign ((void **)  &p_z, 32, n* sizeof(particle_t));
l +=posix_memalign ((void **)  &p_vx, 32, n* sizeof(particle_t));
l +=posix_memalign ((void **)  &p_vy, 32, n* sizeof(particle_t));
l +=posix_memalign ((void **)  &p_vz, 32, n* sizeof(particle_t));
if ( l) return 1;
  p.x = __builtin_assume_aligned(p_x, 32);
  p.y = __builtin_assume_aligned(p_y, 32);
  p.z = __builtin_assume_aligned(p_z, 32);
  p.vx = __builtin_assume_aligned(p_vx, 32);
  p.vy = __builtin_assume_aligned(p_vy, 32);
  p.vz = __builtin_assume_aligned(p_vz, 32);
```

FIGURE 5 – Notre programme

Workaround

Use vector aligned instructions:

1. align your arrays on 32 bytes boundaries: replace { void *p = malloc (size); } with { void *p; posix_memalign (&p, 32, size); }.
2. inform your compiler that your arrays are vector aligned: if array 'foo' is 32 bytes-aligned, define a pointer 'p_foo' as __builtin_assume_aligned (foo, 32) and use it instead of 'foo' in the loop.

FIGURE 6 – La sortie CQA

2. **Cas FMA (fused multiply-add) :** Malgré le changement de toute les multiplications et additions avec le format demandé, on retrouve toujours la même suggestion dans notre rapport.

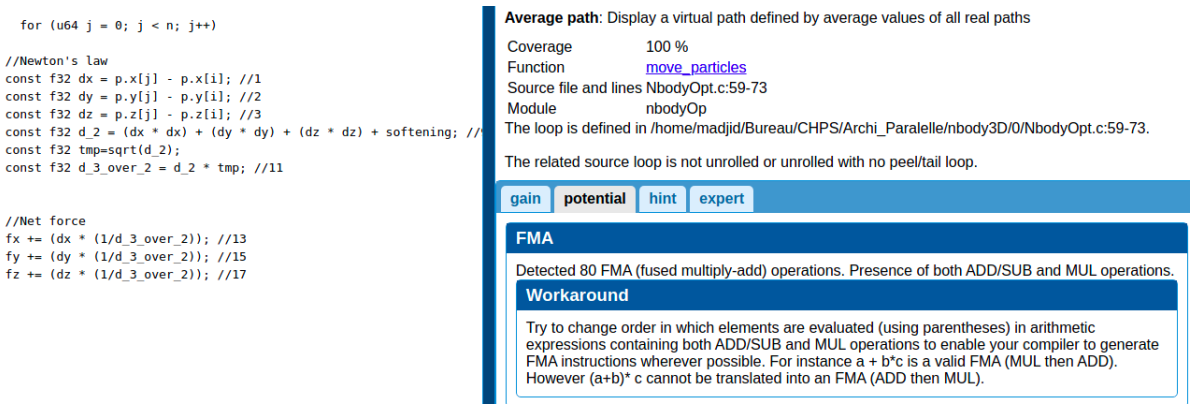


FIGURE 7 – FMA avant

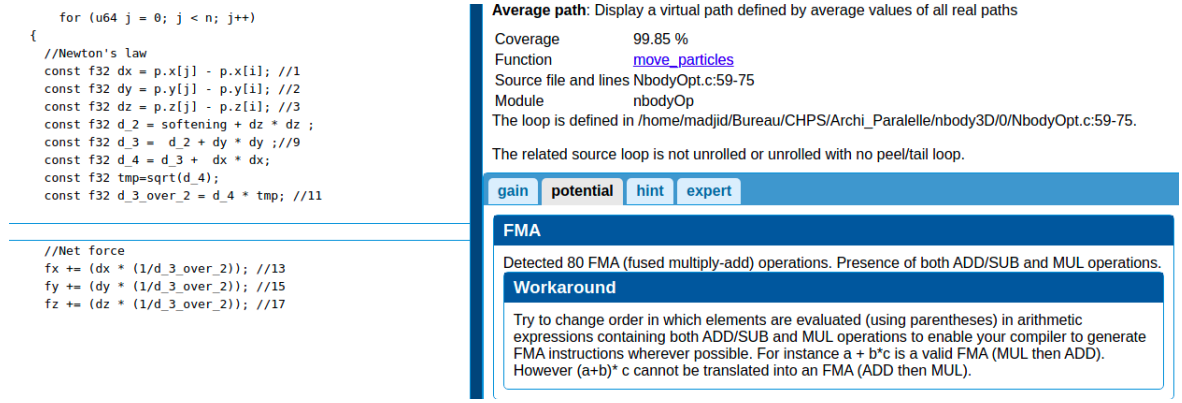


FIGURE 8 – FMA après

3. **Aspect algorithmique :** MAQAO regarde l'aspect vectorization et les operations arithmitique combien y'a d'addition, multiplication et division, il ne peut pas savoir en terme d'algorithmique combien en moin on peut avoir de calcul ADD, MUL, SUB et DIV.

Exemple :

```
const f32 tmp=sqrt(d_4);
const f32 d_3_over_2 = d_4 * tmp;
```

```
const f32 tmp=sqrt(d_4);
const f32 d_3_over_2 = tmp * tmp * tmp;
```

Dans cette exemple on aurait pas besoin de multiplier tmp trois fois il suffisé seulement de le multiplie une fois avec d_4 MAQAO n'arrive pas a simplifie les opération arithmitique.

4. Malgré que toute ces modifications que maqao nous a proposé, on a toujours pas eu un résulat de 100% au niveau de la Vectorization Efficiency, après recherche de notre coté on a trouvé que la puissance (POW) est une instruction complexe donc on du la changer à une racine carée (SQRT) pour régler notre problème de Vectorization Efficiency et avoir un fully vectorized. Maqao nous a pas aidé dans cette étape pour pouvoir utiliser toute la longueur des registres, c'est par nous même que nous avons trouvé cette instruction complexe qu'on devait changer pour avoir un resultat plus performant.

Incompréhension de l'origine de cette erreur :

```
PANIC: unprotected error in call to Lua API ([string "lua_oneview"
   ]:40457: attempt to index local '__path__' (a nil value))
```

Le programme s'execute le plus normalement possible avec un temps d'execution élevé mais cette erreur ce manifeste et que on comprend pas.