Table des matières

Table des figures

1 Optimisation de la simulation “Kermann Vortice”

L’objectif est de rendre le programme de simulation “Kermann Vortice” le plus scalable possible, pour ce faire nous allons procéder à l’optimisation de ce dernier en utilisant l’outil MAQAO.

1.1 Analyse de la version initiale du programme

Dans un premier temps nous avons créé un fichier config.lua avec la commande suivante :

```
maqao oneview -create-config=config.lua
```

Voici la configuration que nous avons implémenté dans le fichier config.lua créé précédemment :

```
1  -- Name of the experiment specified in report main pages
2  experiment_name = nil
3
4  -- Name of the executable to analyze
5  executable  = "./lbm"
6  run_command = "<executable>"
7  multiruns_params = {
8  {name="R2x4",envv_OMP_NUM_THREADS="4", number_processes=2,
       number_processes_per_node = 2},
9  {name="R3x4",envv_OMP_NUM_THREADS="4", number_processes=3,
       number_processes_per_node = 3},
10 {name="R4x2",envv_OMP_NUM_THREADS="2", number_processes=4,
       number_processes_per_node = 4},
11 {name="R4x4",envv_OMP_NUM_THREADS="4", number_processes=4,
       number_processes_per_node = 4},
12 {name="R6x2",envv_OMP_NUM_THREADS="2", number_processes=6,
       number_processes_per_node = 6},
13 {name="R8x2",envv_OMP_NUM_THREADS="2", number_processes=8,
       number_processes_per_node = 8},
14 {name="R16x16",envv_OMP_NUM_THREADS="16", number_processes=16,
       number_processes_per_node = 16},
15 {name="R10x8",envv_OMP_NUM_THREADS="8", number_processes=10,
       number_processes_per_node = 16},
16 {name="R16x1",envv_OMP_NUM_THREADS="1", number_processes=16,
       number_processes_per_node = 16}
17 }
```

nous avons par la suite procédé à l’analyse du programme avec l’outil MAQAO en lançant la commande suivante :

```
maqao oneview -R1 -xp=multirun -WS -c=config.lua
```

1.2 Analyse des résultats obtenus

1.2.1 Global metrics

▶ Compared Reports

Global Metrics

Metric		r0	r1	r2	r3	r4	r5	r6	r7	r8	r9
Total Time (s)		118.51	118.03	118.57	118.03	118.45	118.33	118.18	117.55	118.13	118.46
Profiled Time (s)		115.20	114.82	115.35	114.83	115.24	115.22	115.07	114.40	114.90	115.37
Time in analyzed loops (%)		98.7	98.7	98.7	98.7	98.7	98.7	98.8	98.6	98.7	98.8
Time in analyzed innermost loops (%)		98.7	98.6	98.6	98.7	98.6	98.6	98.7	98.6	98.7	98.7
Time in user code (%)		99.9	99.9	99.9	99.9	99.9	99.9	99.9	99.9	99.9	99.9
Compilation Options		lbm: -funroll-loops is missing.	lbm: -funroll-loops is missing.	lbm: -funroll-loops is missing.	lbm: -funroll-loops is missing.	lbm: -funroll-loops is missing.	lbm: -funroll-loops is missing.	lbm: -funroll-loops is missing.	lbm: -funroll-loops is missing.	lbm: -funroll-loops is missing.	lbm: -funroll-loops is missing.
Perfect Flow Complexity		1.77	1.77	1.77	1.77	1.77	1.77	1.78	1.78	1.78	1.78
Array Access Efficiency (%)		67.2	67.2	67.3	67.2	67.2	67.3	67.2	67.2	67.2	67.2
Perfect OpenMP + MPI + Pthread		1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Perfect OpenMP + MPI + Pthread + Perfect Load Distribution		1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
No Scalar Integer	Potential Speedup	1.60	1.60	1.60	1.60	1.60	1.60	1.61	1.60	1.61	1.61
	Nb Loops to get 80%	1	1	1	1	1	1	1	1	1	1
FP Vectorised	Potential Speedup	2.72	2.72	2.72	2.72	2.72	2.72	2.72	2.72	2.72	2.72
	Nb Loops to get 80%	2	2	2	2	2	2	2	2	2	2
Fully Vectorised	Potential Speedup	7.38	7.38	7.38	7.39	7.37	7.39	7.41	7.34	7.38	7.41
	Nb Loops to get 80%	2	2	2	2	2	2	2	2	2	2
Only FP Arithmetic	Potential Speedup	2.14	2.14	2.14	2.14	2.14	2.14	2.14	2.14	2.14	2.15
	Nb Loops to get 80%	2	2	2	2	2	2	2	2	2	2
Scalability - Gap		1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	1.00	1.00

FIGURE 1 – Global metrics mutlirun initial

La section Global Metrics de la page d'accueil nous permet de constater que le programme n'est pas scalable (car le temps d'exécution est le même sur différentes exécutions). Nous pouvons également remarquer le manque de flag de compilation.

1.2.2 Scalabilité

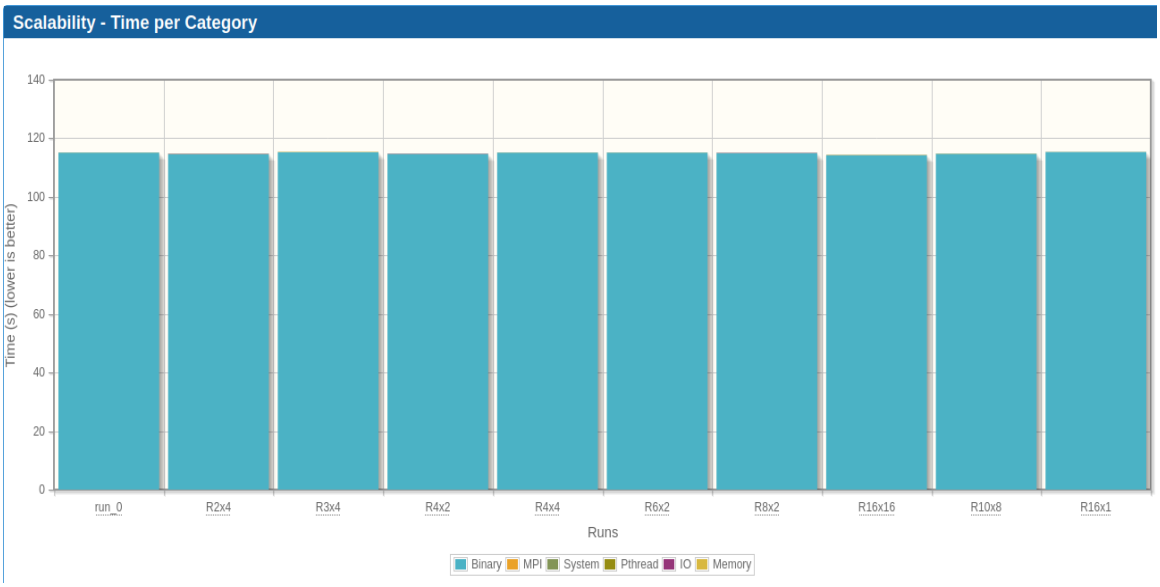

FIGURE 2 – Scalabité

En vérifiant la scalabilité du code en consultant la rubrique Scalability - Time per Category de la section Application, nous nous apercevons qu'effectivement, quelle que soit la configuration utilisée, le temps d'exécution reste constant.

1.2.3 Functions

Name	Module	Max Time Over Threads run_0 (s)	Max Time Over Threads R2x4 (s)	Max Time Over Threads R3x4 (s)	Max Time Over Threads R4x2 (s)	Max Time Over Threads R4x4 (s)	Max Time Over Threads R6x2 (s)	Max Time Over Threads R8x2 (s)	Max Time Over Threads R16x16 (s)	Max Time Over Threads R10x8 (s)	Max Time Over Threads R16x1 (s)
▸ collision	lbm	55.88	55.68	55.92	55.64	55.82	55.76	55.63	55.3	55.41	55.66
▸ propagation	lbm	55.29	55.1	55.38	55.11	55.38	55.38	55.41	55.02	55.47	55.67

FIGURE 3 – Récapitulatif des fonctions appelées

Les boucles et fonctions que nous identifions comme goulots d'étranglement sont des fonctions collision qui calcule les collisions sur chaque cellule et la fonction propagation qui propager les densités sur les mailles voisines.

1.3 Optimisation

1.3.1 Modification makefile

```
CFLAGS := -Wall -Wextra -g -I include/ -fopenmp
OFLAGS := -march=native -mtune=native -mavx2 -Ofast -ffast-math -funsafe-
    math-optimizations -finline-functions -funroll-loops -floop-interchange
    -fpeel-loops -ftree-vectorize -ftree-loop-vectorize -fomit-frame-pointer
     -flto
```

1.3.2 suppression de certaines Barrières, et le changement du schéma de communication

Exemple :

```
// Left to right phase
    lbm_comm_sync_ghosts_horizontal(mesh, mesh_to_process, COMM_SEND,
                                    mesh->right_id, mesh->width - 2);
    lbm_comm_sync_ghosts_horizontal(mesh, mesh_to_process, COMM_RECV,
                                    mesh->left_id, 0);
    // Prevend comm mixing to avoid bugs
    MPI_Barrier(MPI_COMM_WORLD);

    // Right to left phase
    lbm_comm_sync_ghosts_horizontal(mesh, mesh_to_process, COMM_SEND,
                                    mesh->left_id, 1);
    lbm_comm_sync_ghosts_horizontal(mesh, mesh_to_process, COMM_RECV,
                                    mesh->right_id, mesh->width - 1);
////
lbm_comm_sync_ghosts_horizontal(mesh_to_process, COMM_RECV, mesh->left_id,
```

```
16                                          0);
17
18      // Right to left phase
19      lbm_comm_sync_ghosts_horizontal(mesh_to_process, COMM_SEND, mesh->
        left_id,
20                                          1);
21      lbm_comm_sync_ghosts_horizontal(mesh_to_process, COMM_RECV, mesh->
        right_id,
22                                          mesh->width - 1);
23  , COMM_RECV,
24                                          mesh->right_id, mesh->width - 1);
```

1.3.3 parallélisation des boucles contenues dans les fonction “collision” et “propagation” grâce à des directives OPENMP

Exemple :

```
1
2  void collision(Mesh* mesh_out, const Mesh* mesh_in)
3  {
4      // Loop on all inner cells
5      #pragma omp parallel
6      {
7          #pragma omp for schedule(static)
8          for (size_t i = 1; i < mesh_in->width - 1; i++) {
9              for (size_t j = 1; j < mesh_in->height - 1; j++) {
10                 compute_cell_collision(Mesh_get_cell(mesh_out, i, j),
       Mesh_get_cell(mesh_in, i, j));
11             }
12         }
13     }
14 }
```

Enfin, nous avons lancé l’outil MAQAO sur notre version optimisée du code. Voici les résultats obtenus :

Global metrics

Global Metrics

Metric		r0	r1	r2	r3	r4	r5	r6
Total Time (s)		7.60	4.28	3.07	2.86	3.35	3.89	5.21
Profiled Time (s)		6.85	3.73	2.43	1.61	1.04	1.04	0.88
Time in analyzed loops (%)		98.9	93.9	81.9	66.4	45.3	36.8	30.0
Time in analyzed innermost loops (%)		97.7	92.8	80.3	65.5	44.5	36.0	29.4
Time in user code (%)		98.9	93.9	81.9	66.4	45.3	36.8	30.1
Compilation Options		OK	OK	OK	OK	OK	OK	OK
Perfect Flow Complexity		1.78	1.72	1.58	1.43	1.25	1.20	1.16
Array Access Efficiency (%)		50.5	50.4	50.6	50.8	50.9	50.6	50.6
Perfect OpenMP + MPI + Pthread		1.00	1.08	1.19	1.34	1.80	2.17	1.96
Perfect OpenMP + MPI + Pthread + Perfect Load Distribution		1.00	1.09	1.22	1.47	2.01	2.47	2.81
No Scalar Integer	Potential Speedup	1.58	1.54	1.44	1.33	1.20	1.16	1.13
	Nb Loops to get 80%	1	1	1	1	1	1	1
FP Vectorised	Potential Speedup	2.35	2.21	1.92	1.63	1.36	1.27	1.21
	Nb Loops to get 80%	2	2	2	2	2	2	2
Fully Vectorised	Potential Speedup	4.37	3.74	2.79	2.08	1.55	1.40	1.31
	Nb Loops to get 80%	2	2	2	2	2	2	2
Only FP Arithmetic	Potential Speedup	2.50	2.33	1.99	1.68	1.38	1.29	1.23
	Nb Loops to get 80%	2	2	2	2	2	2	2
Scalability - Gap		1.00	1.13	1.61	3.00	7.04	10.24	21.93

FIGURE 4 – Global metrics omptimized version

La section Global Metrics de la page d'accueil nous permet de constater une net amélioration du programme comparé à la version précédente.

Scalabilité

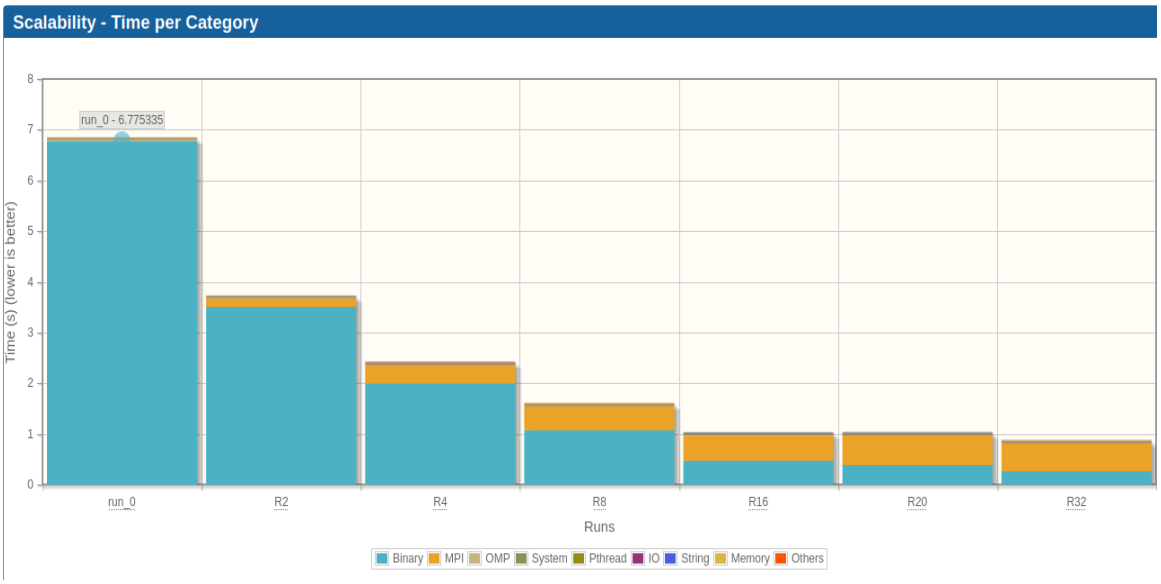

FIGURE 5 – Scalability omptimized version

En vérifiant la scalabilité du code en consultant la rubrique Scalability - Time per Category de la section Application, nous nous apercevons qu'effectivement, le programme est scalable.

Functions

Name	Module	Max Time Over Threads R2 (s)	Max Time Over Threads R4 (s)	Max Time Over Threads R8 (s)	Max Time Over Threads R16 (s)	Max Time Over Threads R20 (s)	Max Time Over Threads R32 (s)
▸ propagation._omp_fn.0	lbm	1.71	0.99	0.58	0.28	0.22	0.24
▸ collision._omp_fn.0	lbm	1.59	0.88	0.53	0.24	0.19	0.2

FIGURE 6 – Functions omptimized version

En vérifiant les fonctions du code en consultant la rubrique Fonctions - Columns Filter nous pouvons voir une net amélioration du temps d'exécution des fonction “collision” et “propagation”