Rapport Code GenMat

Makefile

Le makefile qui permet de compiler notre programme GenMat est le suivant:

```
all:exec

exec:
	gcc -g  -std=c99  GenMat.c -o GenMat -lm

clean:
	rm -Rf *~ GenMat
```

Figure 1: Makefile

Afin d'optimiser notre programme GenMat nous faisons appelle à MAQAO qui analysera notre binaire et générera un rapport contenant une vue globale de notre programme et les suggestions qui permettront d'améliorer les performances de notre programme.

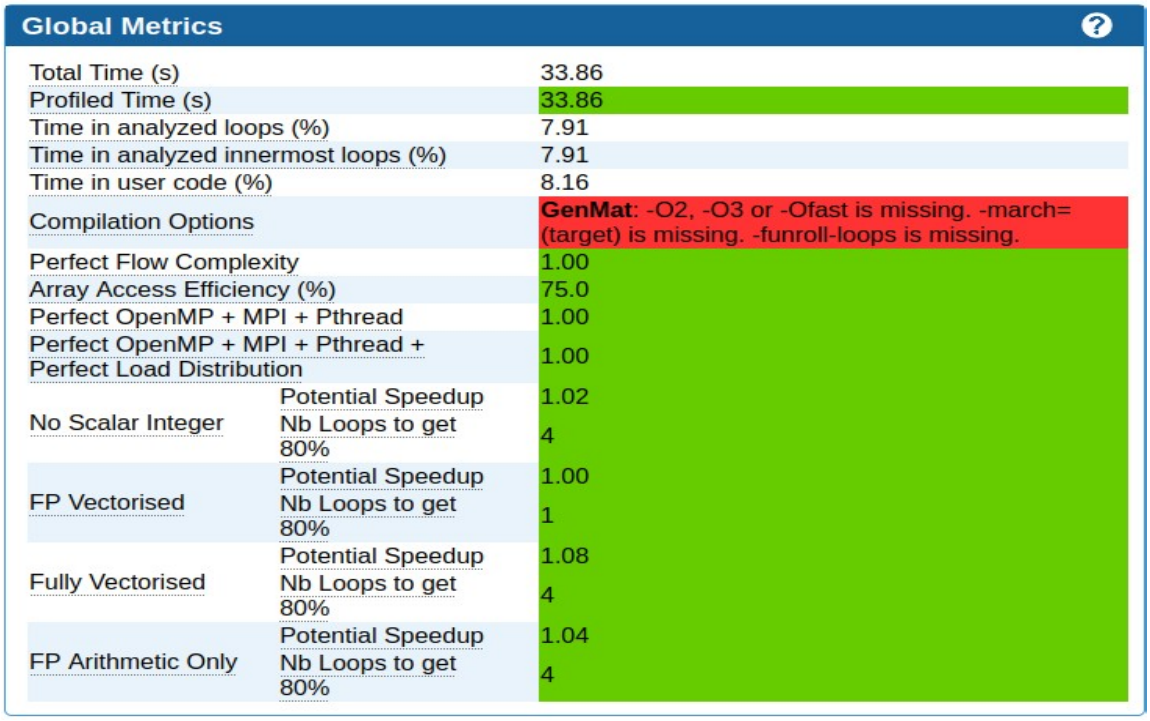
MAQAO

Après l'exécution de MAQAO, les fichier HTML suivants sont générés:

Figure 2: Fichier HTML générés par MAQAO

Global Metrics

Pour commencer notre analyse nous étudierons la page d'accueil du rapport généré par MAQAO.



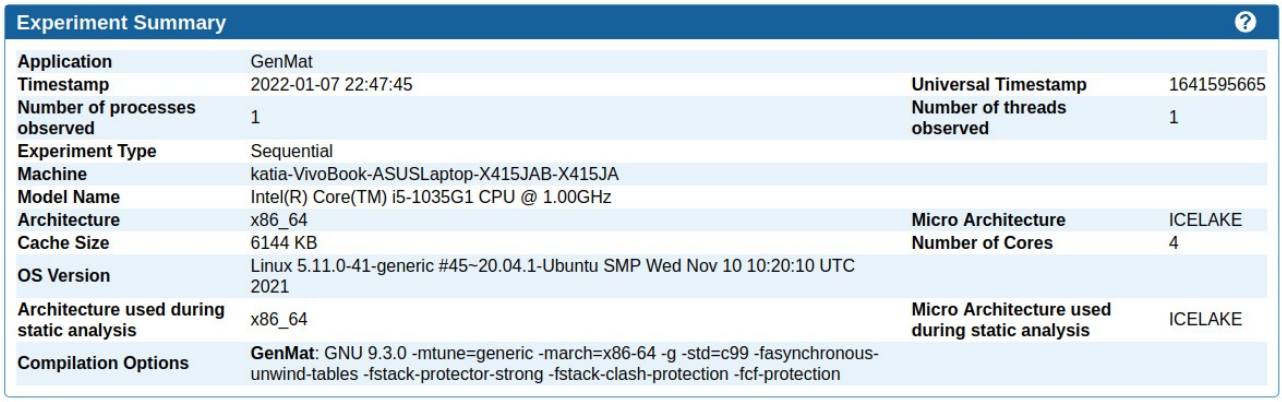
Global Metrics		
Total Time (s)		33.86
Profiled Time (s)		33.86
Time in analyzed loops (%)		7.91
Time in analyzed innermost loops (%)		7.91
Time in user code (%)		8.16
Compilation Options		**GenMat**: -O2, -O3 or -Ofast is missing. -march=(target) is missing. -funroll-loops is missing.
Perfect Flow Complexity		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	1.02
	Nb Loops to get 80%	4
FP Vectorised	Potential Speedup	1.00
	Nb Loops to get 80%	1
Fully Vectorised	Potential Speedup	1.08
	Nb Loops to get 80%	4
FP Arithmetic Only	Potential Speedup	1.04
	Nb Loops to get 80%	4

Figure 3: Global Metrics

En observant le rapport Global Metrics de notre binaire analysé par MAQAO, nous constatons que celui ci a été compiler sans flags d'optimisation ni de flags de spécification d'architecture. Nous pouvons également remarquer que les accès mémoire sont efficaces à 75%. A cette étape, nous allons prendre en compte la suggestion des flags -O2,-O3 ou -Ofast, -march=(target),-funroll-loops, pour le prochain binaire à produire.

Experiment Summary

Experiment Summary nous donne les informations liées à la machine d'exécution,ainsi que les flags ajoutés par le compilateur.

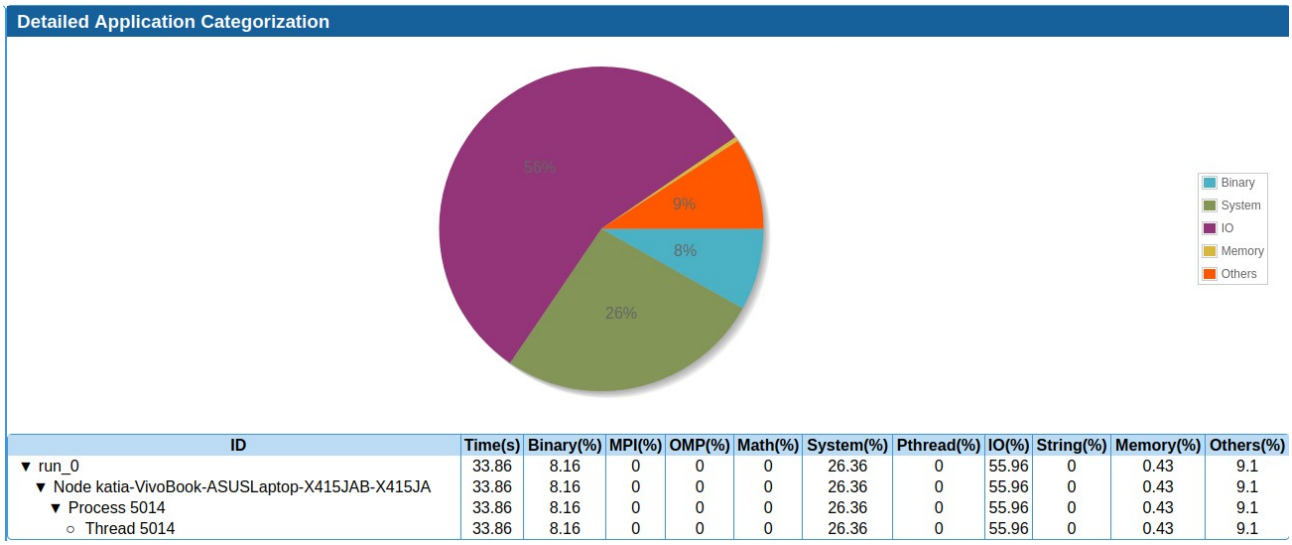


Experiment Summary			
Application	GenMat		
Timestamp	2022-01-07 22:47:45	**Universal Timestamp**	1641595665
Number of processes observed	1	**Number of threads observed**	1
Experiment Type	Sequential		
Machine	katia-VivoBook-ASUSLaptop-X415JAB-X415JA		
Model Name	Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz		
Architecture	x86_64	**Micro Architecture**	ICELAKE
Cache Size	6144 KB	**Number of Cores**	4
OS Version	Linux 5.11.0-41-generic #45~20.04.1-Ubuntu SMP Wed Nov 10 10:20:10 UTC 2021		
Architecture used during static analysis	x86_64	**Micro Architecture used during static analysis**	ICELAKE
Compilation Options	**GenMat**: GNU 9.3.0 -mtune=generic -march=x86-64 -g -std=c99 -fasynchronous-unwind-tables -fstack-protector-strong -fstack-clash-protection -fcf-protection		

Figure 4: Experiment Summary

Application

Application donne les détails sur la manière dont le temps a été reparti entre les différentes catégories (Binary, System, memory, …).

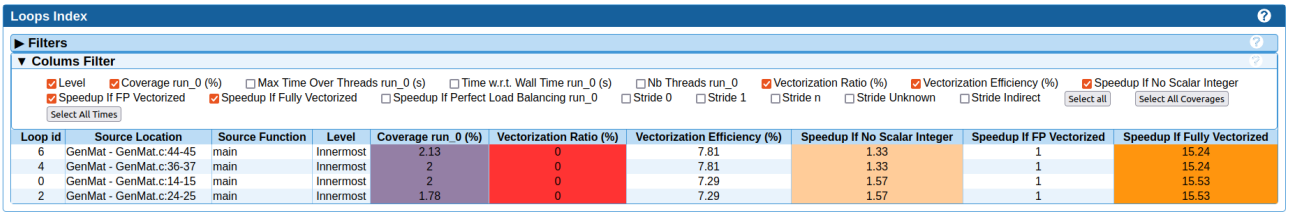


ID	Time(s)	Binary(%)	MPI(%)	OMP(%)	Math(%)	System(%)	Pthread(%)	IO(%)	String(%)	Memory(%)	Others(%)
▼ run_0	33.86	8.16	0	0	0	26.36	0	55.96	0	0.43	9.1
▼ Node katia-VivoBook-ASUSLaptop-X415JAB-X415JA	33.86	8.16	0	0	0	26.36	0	55.96	0	0.43	9.1
▼ Process 5014	33.86	8.16	0	0	0	26.36	0	55.96	0	0.43	9.1
○ Thread 5014	33.86	8.16	0	0	0	26.36	0	55.96	0	0.43	9.1

Figure 5: Application

Loops

La rubrique Loops nous donne des indications sur les boucle de notre programme, leurs temps d'exécution, couverture, vectorisation, …



Loop id	Source Location	Source Function	Level	Coverage run_0 (%)	Vectorization Ratio (%)	Vectorization Efficiency (%)	Speedup If No Scalar Integer	Speedup If FP Vectorized	Speedup If Fully Vectorized
6	GenMat - GenMat.c:44-45	main	Innermost	2.13	0	7.81	1.33	1	15.24
4	GenMat - GenMat.c:36-37	main	Innermost	2	0	7.81	1.33	1	15.24
0	GenMat - GenMat.c:14-15	main	Innermost	2	0	7.29	1.57	1	15.53
2	GenMat - GenMat.c:24-25	main	Innermost	1.78	0	7.29	1.57	1	15.53

Figure 6: Loops

Rapport CQA

Pour générer le rapport CQA il suffit de cliquer sur l'une des boucles dans la rubrique Loops:

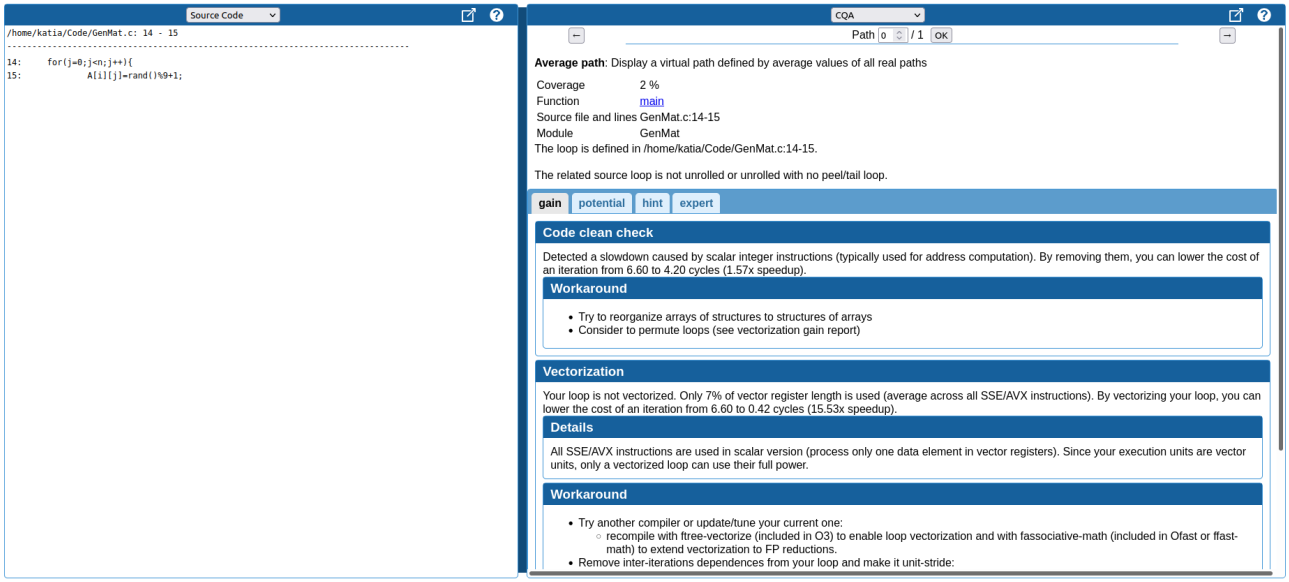


Figure 7: Rapport CQA

le rapport CQA indique les boucles chaudes c'est a dire les boucles qui doivent être optimiser dans notre cas la couverture (coverage) de nos boucles est bonne.

Gain

Dans cette rubrique MAQAO fait une estimations du gain de temps si nous vectorisons nos boucles et si nous changeons la structure de Arrays of Structure (AoS) à Structure of Arrays (SoA).

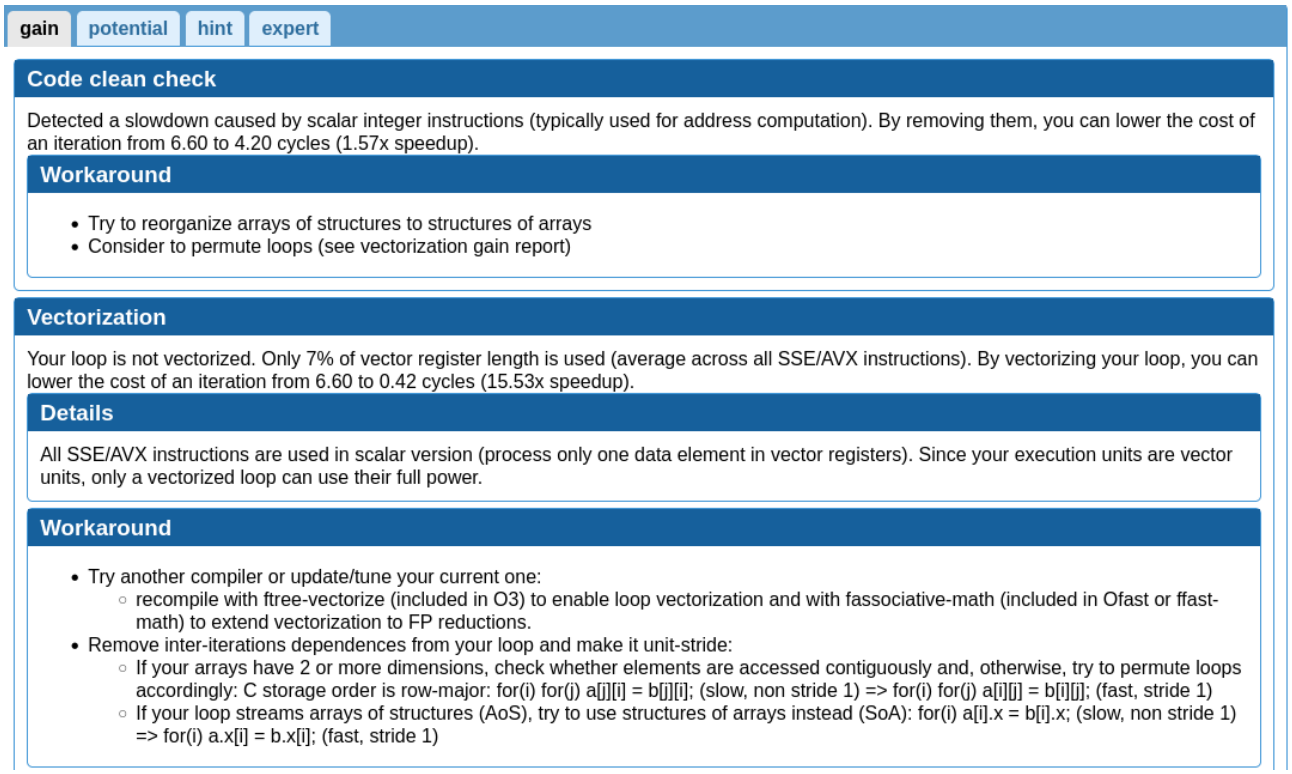


Figure 8: Gain

Optimisation

Makefile

Pour l'optimisation de notre programme nous prenons en compte les suggestions de MAQAO, en ajoutant les flags d'optimisation dans notre makefile:

```
1 all:exec
2
3 exec:
4 	gcc -g  -std=c99 -Ofast -march=native -funroll-loops GenMat.c -o GenMat -lm
5
6 clean:
7 	rm -Rf *~ GenMat
```

Figure 9: Makefile modifié

Global Metrics

Global Metrics

Total Time (s)		29.97
Profiled Time (s)		29.97
Time in analyzed loops (%)		5.91
Time in analyzed innermost loops (%)		5.89
Time in user code (%)		6.13
Compilation Options		OK
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%	1
FP Vectorised	Potential Speedup	1.00
	Nb Loops to get 80%	1
Fully Vectorised	Potential Speedup	1.06
	Nb Loops to get 80%	3
FP Arithmetic Only	Potential Speedup	1.00
	Nb Loops to get 80%	1

Figure 10: Global Metrics 2