Rapport “Matrix”

Processus d’optimisation avec l’outil Maqao :

Dans un premier temps, on prend un code auquel on aimerait visualiser la performance, ensuite on l'exécute avec un makefile basic (figure 1).

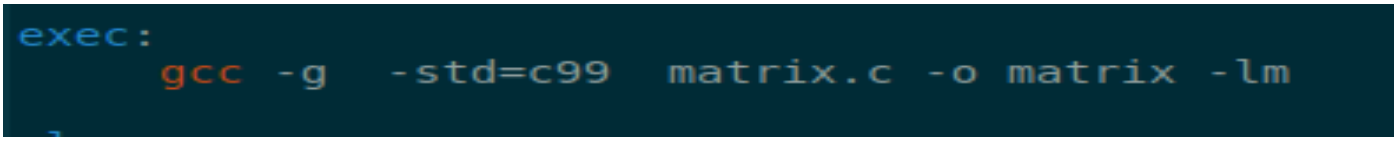


Figure -1- makefile

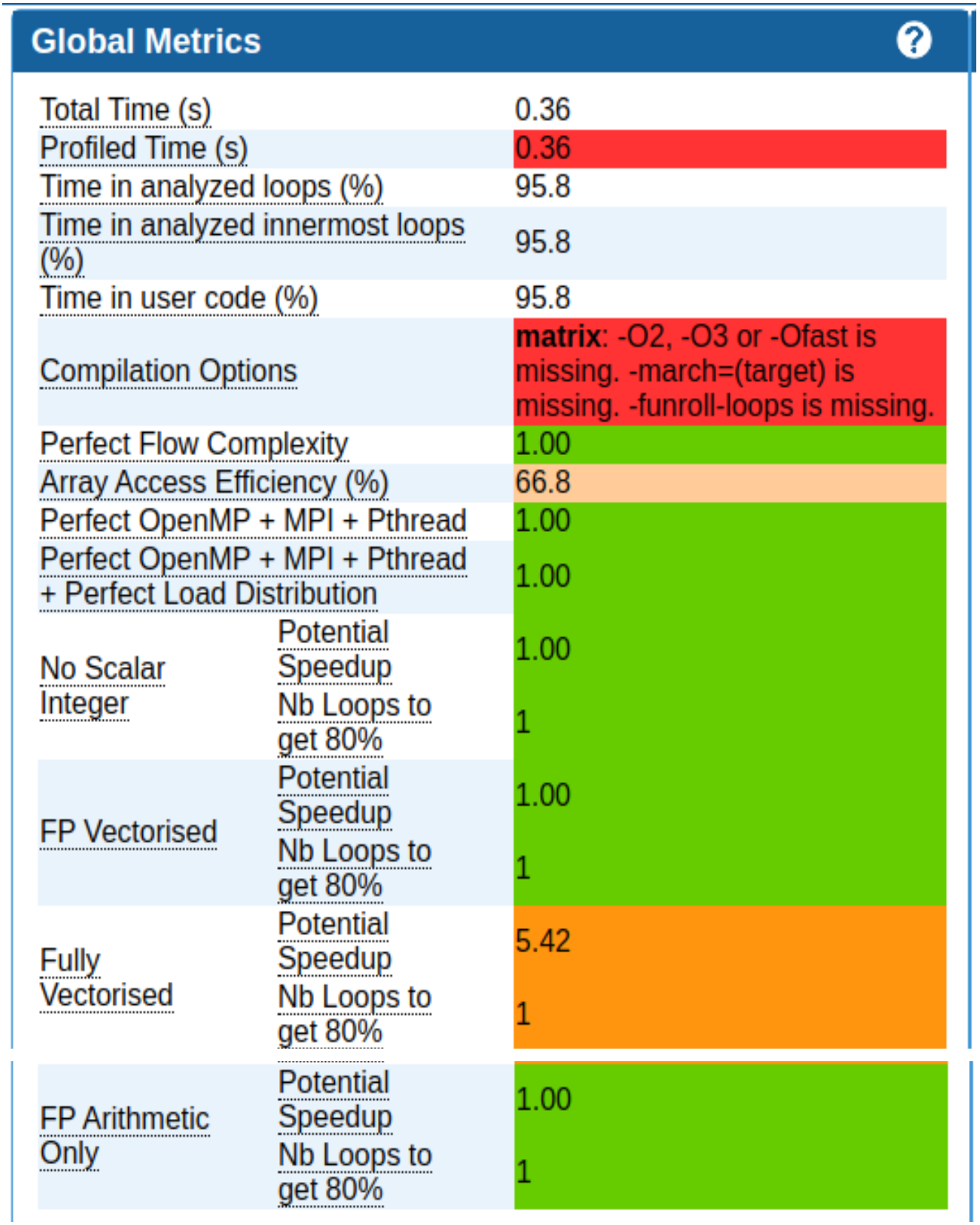
Après l’obtention de l'exécutable du code, on peut commencer à profiler les performances du code grâce à l’outil Maqao en lançant la commande suivante sur le terminal :

Figure -2- ligne de commande.

Une fois la ligne de commande lancée et traitée, maqao génère un rapport général sous format HTML à propos de l'exécutable du code et ainsi, on peut commencer à voir ce qu’il y a lieu de modifier afin d’améliorer les performances du code analysé.

Global metrics (index.html) :

index.html est la page contenant le résultat générale de l’analyse de l'exécutable par l’outil maqao, en voici un aperçu dans la figure 3 ci-dessous :



Global Metrics

Total Time (s)		0.36
Profiled Time (s)		0.36
Time in analyzed loops (%)		95.8
Time in analyzed innermost loops (%)		95.8
Time in user code (%)		95.8
Compilation Options		**matrix**: -O2, -O3 or -Ofast is missing. -march=(target) is missing. -funroll-loops is missing.
Perfect Flow Complexity		1.00
Array Access Efficiency (%)		66.8
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%	1
FP Vectorised	Potential Speedup	1.00
	Nb Loops to get 80%	1
Fully Vectorised	Potential Speedup	5.42
	Nb Loops to get 80%	1
FP Arithmetic Only	Potential Speedup	1.00
	Nb Loops to get 80%	1

Figure -3- Global metrics

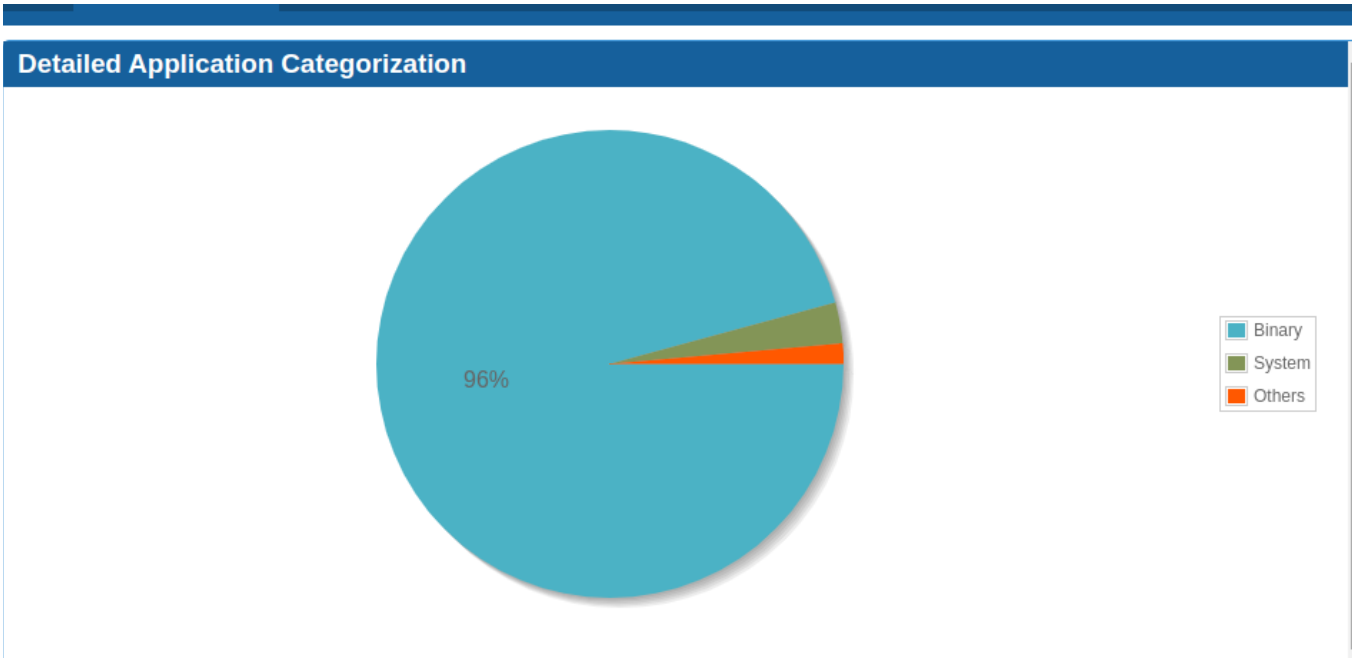
Les métriques globales qui résultent après l'analyse de l'exécutable par l'outil maqao, on peut constater qu'il y a manquement des flages de d'optimisation ou ceux de la spécification d'architecture.

Ainsi, comme on peut le voir sur la figure 3, il nous faut donc modifier notre makefile en ajoutant les flags de compilation manquants afin d'optimiser notre code.

On peut aussi améliorer la vectorisation d'une boucle dans le code afin d'optimiser le temps passé dans cette dernière et qu'elle peut être améliorée de 80%.

Application (application.html) :

La page application nous permet d'avoir plus de détails sur comment le temps d'exécution est partitionné sur plusieurs catégories telles que : les appels systeme, le binaire du code, les fonctions I/O etc. La figure 4 ci-dessous détaille tout cela pour notre code :



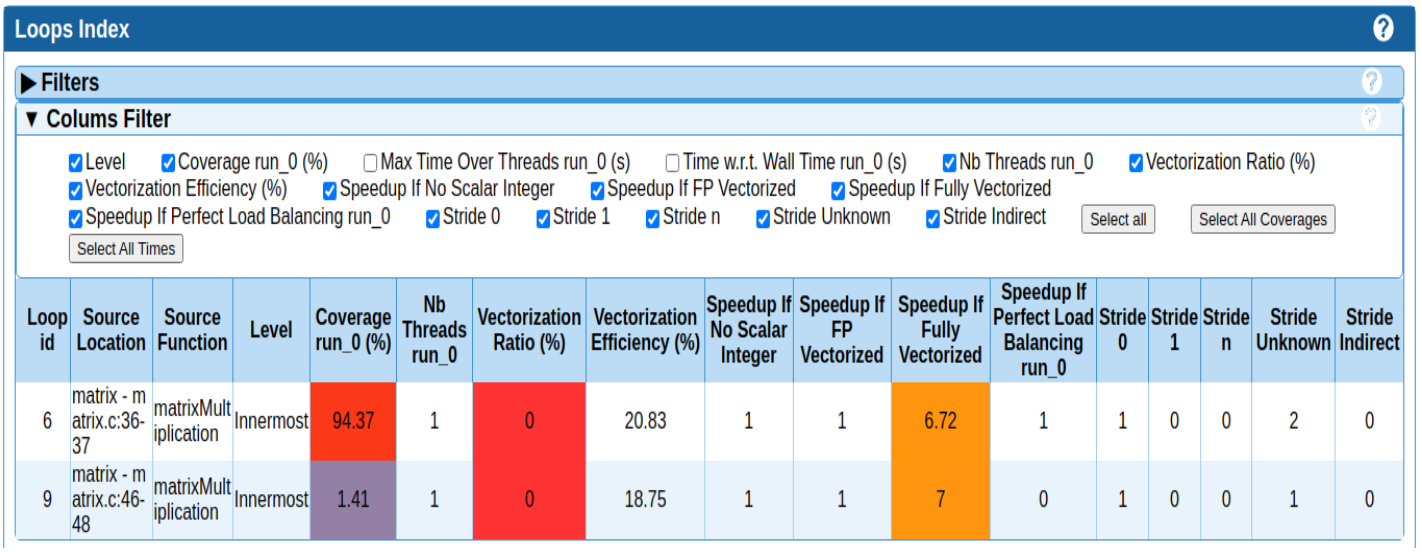
ID	Time(s)	Binary(%)	MPI(%)	OMP(%)	Math(%)	System(%)	Pthread(%)	IO(%)	String(%)	Memory(%)	Others(%)
▼ run_0	0.36	95.77	0	0	0	2.82	0	0	0	0	1.41
▼ Node benbachir-Lenovo	0.36	95.77	0	0	0	2.82	0	0	0	0	1.41

Figure -4- application.

On peut voir que pratiquement 96% du temps d'exécution est passée sur le binaire du code, puis on a 2.82% du temps d'exécution sur le système et enfin 1.41% de ce temps dans la catégorie others (autres).

Loops (loops_index.html) :

La page Loops est consacré à l'analyse détaillée des boucles du code, ainsi dans la figure 5 suivante, nous allons voir en détails le temps passé sur les boucles du code :



Loop id	Source Location	Source Function	Level	Coverage run_0 (%)	Nb Threads run_0	Vectorization Ratio (%)	Vectorization Efficiency (%)	Speedup If No Scalar Integer	Speedup If FP Vectorized	Speedup If Fully Vectorized	Speedup If Perfect Load Balancing run_0	Stride 0	Stride 1	Stride n	Stride Unknown	Stride Indirect
6	matrix - matrix.c:36-37	matrixMultiplication	Innermost	94.37	1	0	20.83	1	1	6.72	1	1	0	0	2	0
9	matrix - matrix.c:46-48	matrixMultiplication	Innermost	1.41	1	0	18.75	1	1	7	0	1	0	0	1	0

Figure -5- Loops.

Exemple de rapport CQA :

Pour exemple ,on peut constater dans le tableau descriptif à propos des boucles de la figure 5, qu'il y'a une boucle dans notre code qui un pourcentage de 94.37 de coverage, en cliquant sur coverage_run on obtient le rapport CQA à son propos,qu'on peut retrouver dans la figure 6 suivante :

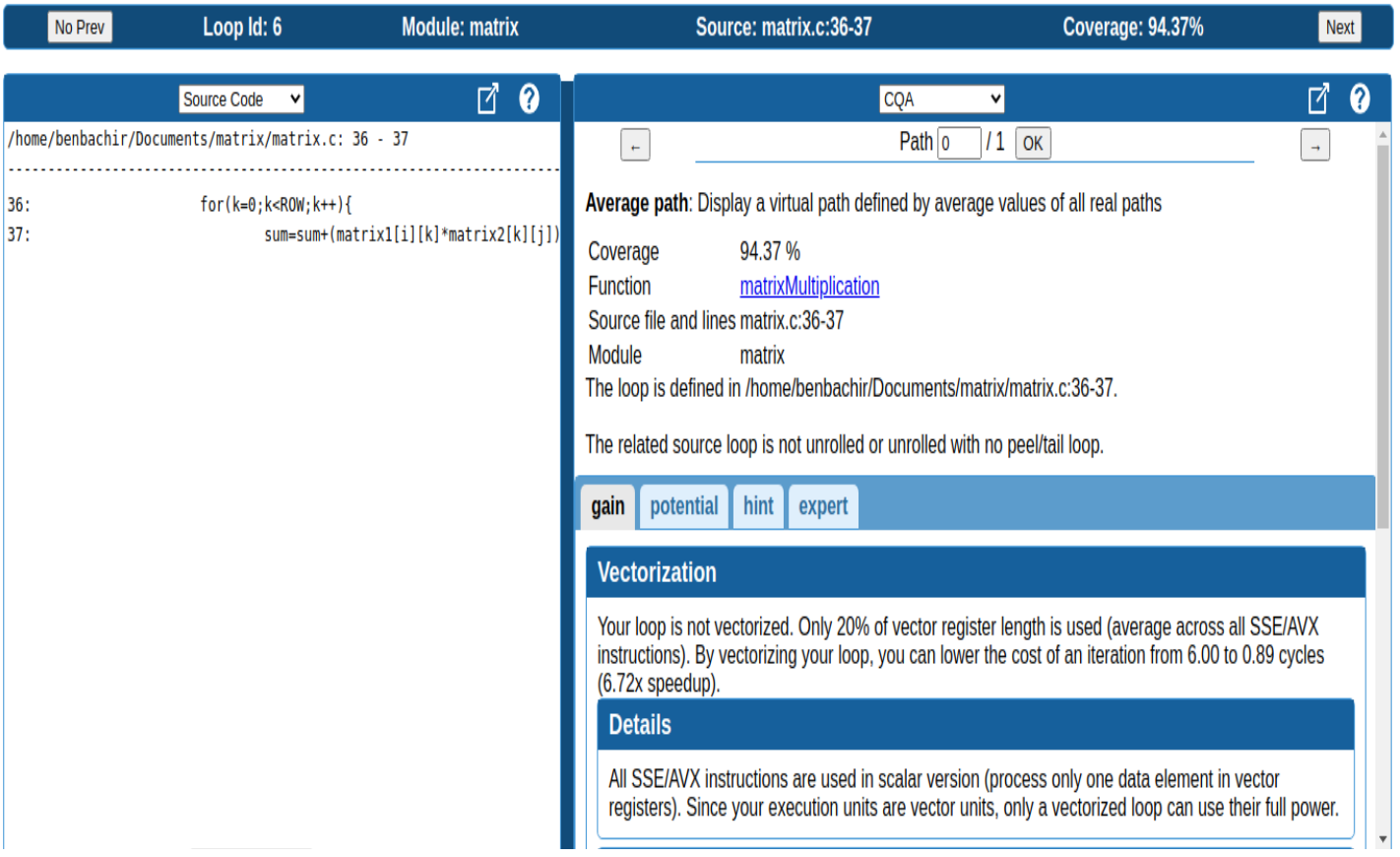


Figure -6- CQA rapport.

Le rapport CQA est un rapport qui nous guide sur les modifications qu'il faut faire afin d'optimiser notre code qui se trouve sur la partie gauche de la figure 6. Ainsi en suivant correctement les recommandations données dans ce rapport, on arrive à améliorer les performances de notre code.

Optimisation :

Makefile après modification :

Le makefile après ajout des flags suggérés par l'outils maqao, la figure 7 suivante montre l'ajout des flags :

```
gcc -g  -std=c99  -O3 -Ofast  -funroll-loops -floop-unroll-and-jam -march=native  matrix.c -o matrix -lm
```

Figure -7- Makefile.

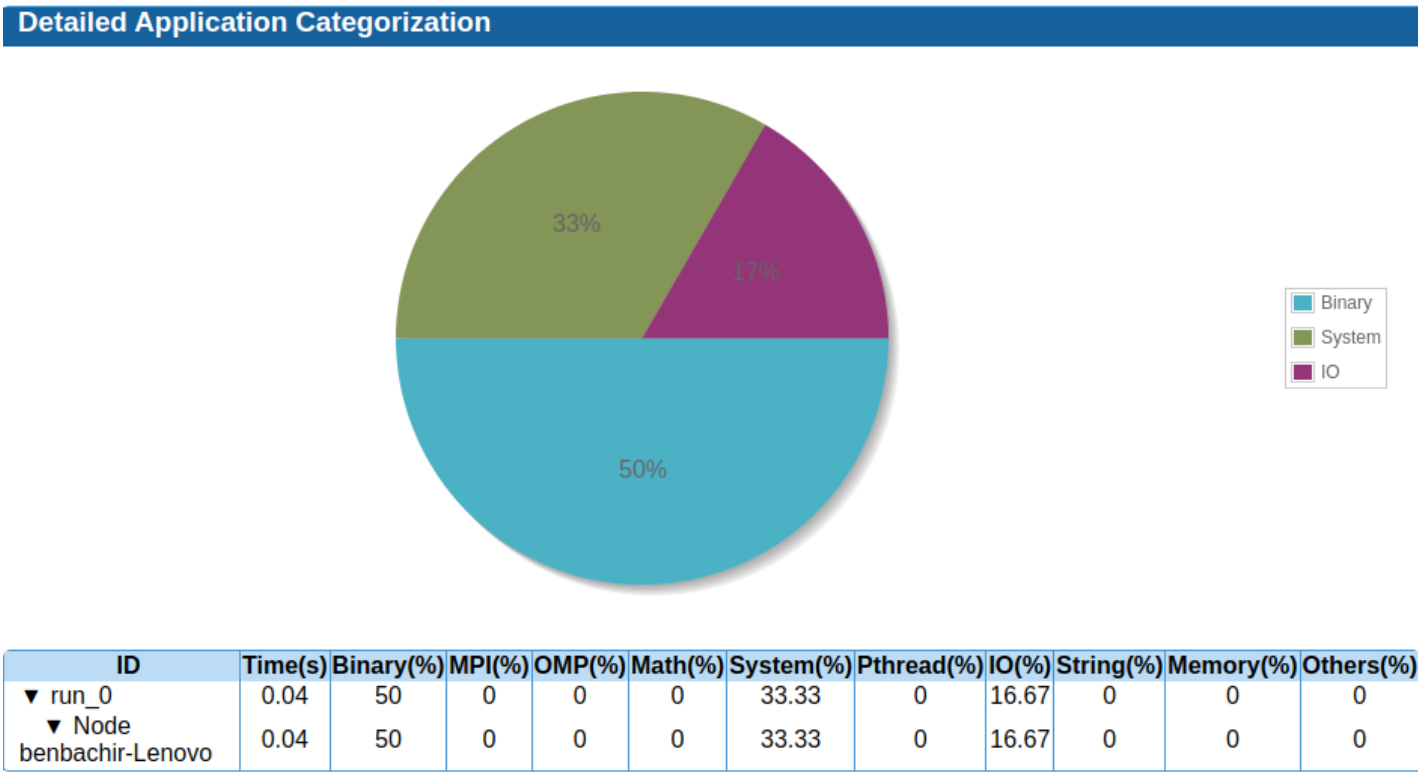
Global metrics (index.html) :

Global Metrics		
Total Time (s)		0.04
Profiled Time (s)		0.04
Time in analyzed loops (%)		50.0
Time in analyzed innermost loops (%)		50.0
Time in user code (%)		50.0
Compilation Options		OK
Perfect Flow Complexity		1.00
Array Access Efficiency (%)		83.3
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	1.00
	Nb Loops to get 80%	1
FP Arithmetic Only	Potential Speedup	1.00
	Nb Loops to get 80%	1

Figure -8- Global metrics after modifications.

On peut constater dans ce rapport qu'il ya eu une amélioration de temps d'execution (de 0.36s à 0.04s), la signalisation des options de compilation est au statut OK et nos boucles sont à présent fully vectorized.

Application (application.html) :

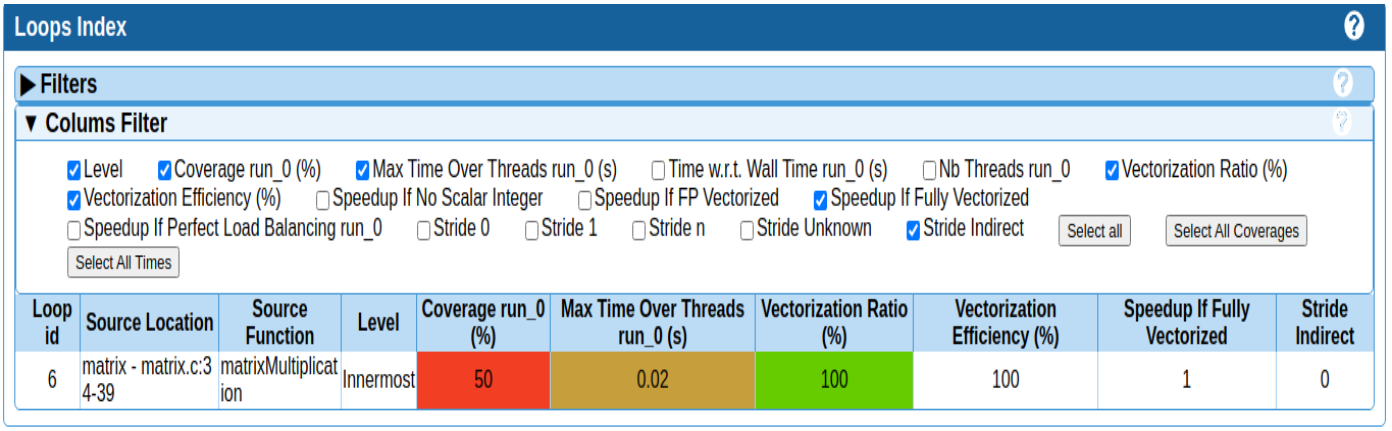


ID	Time(s)	Binary(%)	MPI(%)	OMP(%)	Math(%)	System(%)	Pthread(%)	IO(%)	String(%)	Memory(%)	Others(%)
▼ run_0	0.04	50	0	0	0	33.33	0	16.67	0	0	0
▼ Node benbachir-Lenovo	0.04	50	0	0	0	33.33	0	16.67	0	0	0

Figure -9- application after modifications.

On peut constater que le temps d'exécution est partitionné entre 50% dans le binaire du code, 33% dans le système et 17% dans dans l'I/O des fonctions.

Loops (loops_index.html) :



Loop id	Source Location	Source Function	Level	Coverage run_0 (%)	Max Time Over Threads run_0 (s)	Vectorization Ratio (%)	Vectorization Efficiency (%)	Speedup If Fully Vectorized	Stride Indirect
6	matrix - matrix.c:34-39	matrixMultiplication	Innermost	50	0.02	100	100	1	0

Figure -10- Loops Index after modifications.

On peut remarquer que le coverage_run(le couverage) à diminué de 94% à 50% et que le ratio de la vectorisation est à présent à 100%.

Conclusion :

On peut donc conclure à travers cette utilisation de l'outil maqao sur un code basique sur des matrices que c'est un outil très puissant qui permet un gain conséquent en optimisation et amélioration de la performance d'applications.