Analyse et amélioration des performances d'un programme par compilation optimisante: Étude de cas

TER M1 Informatique

Pierre Tassel sous la direction du Pr Touati

Département Informatique
Universitée Nice Sophia Antipolis

Master Informatique

19 juin 2018

Plan

Présentation du programme

MinMax et le jeux de l'Awale

- MinMax avec coupes Alpha Bêta.
- Variante du jeux de l'Awale avec 6 cases et 4 graines par case.
- Ecrit en C++ par Pr Régin.

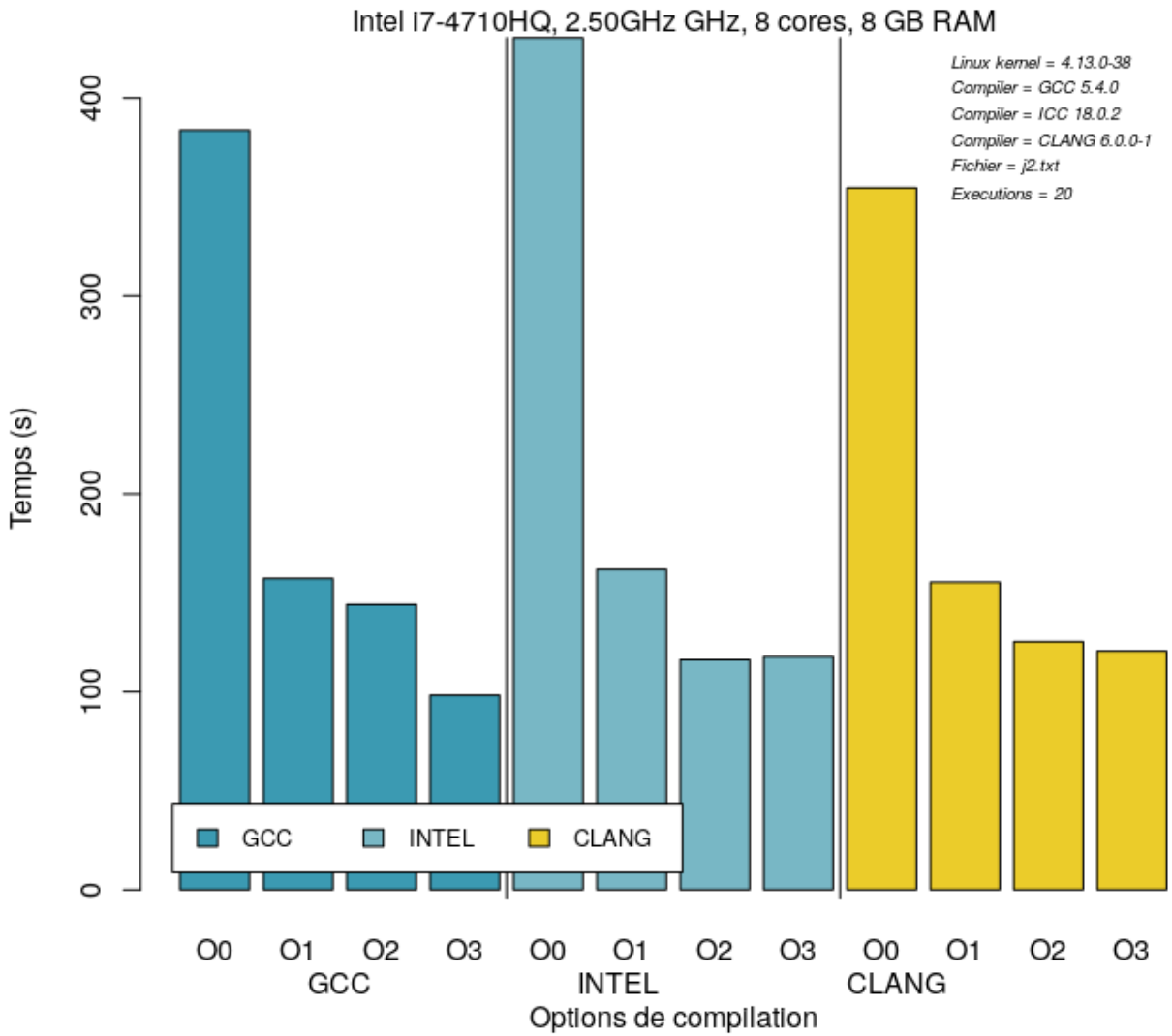
Figure – Le jeu de l'Awalé.

Démarche expérimentale

Environnement matériel et logiciel

- *Processeur Scaling* désactivé, processeur Intel core i7 8 coeurs 2,5GHz, 8Go DDR3, 256Go de SSD.
- Environnement minimaliste en CLI, Arch Linux basé sur le noyau Linux 4.14.13-1.
- Trois compilateurs analysés : GCC, ICC et CLang (dernières versions disponible).
- Fichier d'entrée généré en faisant jouer le programme contre lui même, programme déterministe.
- Les exécutions sont répétés 20 fois pour chaque configuration testée.

Analyse des performances séquentielles



Profilage du code

GProf

```
 %   cumulative          total
time   seconds   calls   s/call  name
48.83    167.72 5172852992  0.00  jouer_coup(Next*, Pos*, Pos*, int, int)
18.25    230.39 3262157329  0.00  copier(Pos*, Pos*)
17.22    289.56 3262157262  0.00  est_affame(Pos*, int)
 8.79    319.75 1457826958  0.00  calculer_coup(Next*, Pos*, int, int, int, int, bool)
 5.31    337.98 3252245939  0.00  valeur_minimaxAB(Next*, Pos*, int, int, int, int, bool)
 0.87    340.99 1775297034  0.00  evaluer(Pos*)
 0.52    342.77     67      0.03  test_fin(Pos*)
 0.09    343.65     33     10.34  decisionAB(Next*, Pos*, int, bool)
```

Figure – Profilage du code avec GProf.

Elapsed Time: 118.666s

CPU Time:	**117.755s**
Instructions Retired:	588,377,500,000
CPI Rate:	0.499
CPU Frequency Ratio:	1.000
Total Thread Count:	1
Paused Time:	0s

Top Hotspots

This section lists the most active functions in your a

Function	Module	CPU Time
jouer_coup	advisor.out	70.132s
calculer_coup	advisor.out	15.441s
copier	advisor.out	14.165s
est_affame	advisor.out	7.176s
valeur_minimaxAB	advisor.out	5.755s
[Others]		5.087s

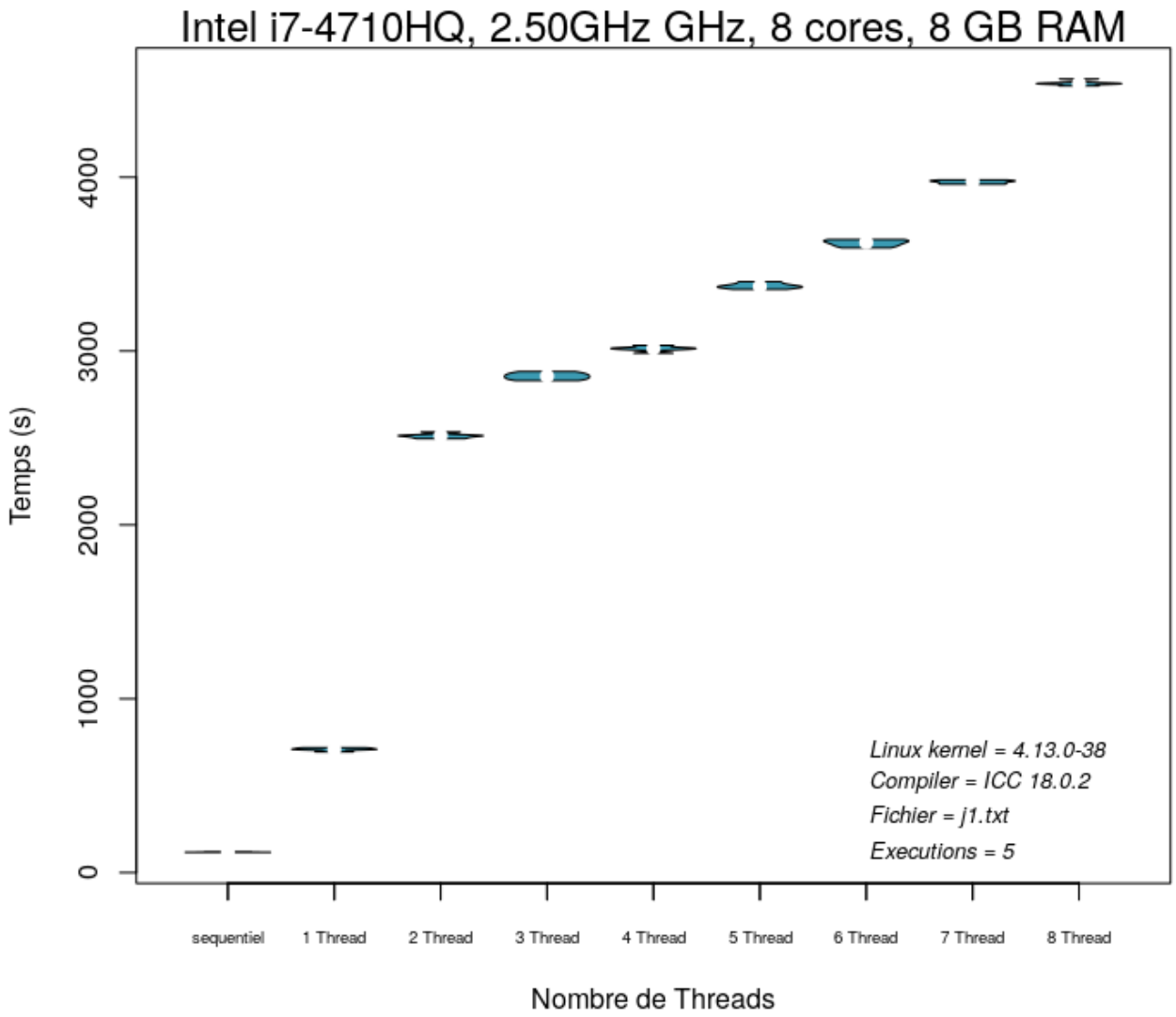
Figure – vTune analyse usage processeur.

Elapsed Time: 120.174s

CPU Time:	118.454s
Memory Bound:	**2.7%**
L1 Bound:	6.6%
DRAM Bound:	
DRAM Bandwidth Bound:	0.0%
LLC Miss:	0.1%
Loads:	157,360,120,662
Stores:	100,867,825,944
LLC Miss Count:	900,054
Average Latency (cycles):	8
Total Thread Count:	1
Paused Time:	0s

Figure – vTune analyse usage mémoire.

Approche naïve d'ajout de parallélisme



False Sharing

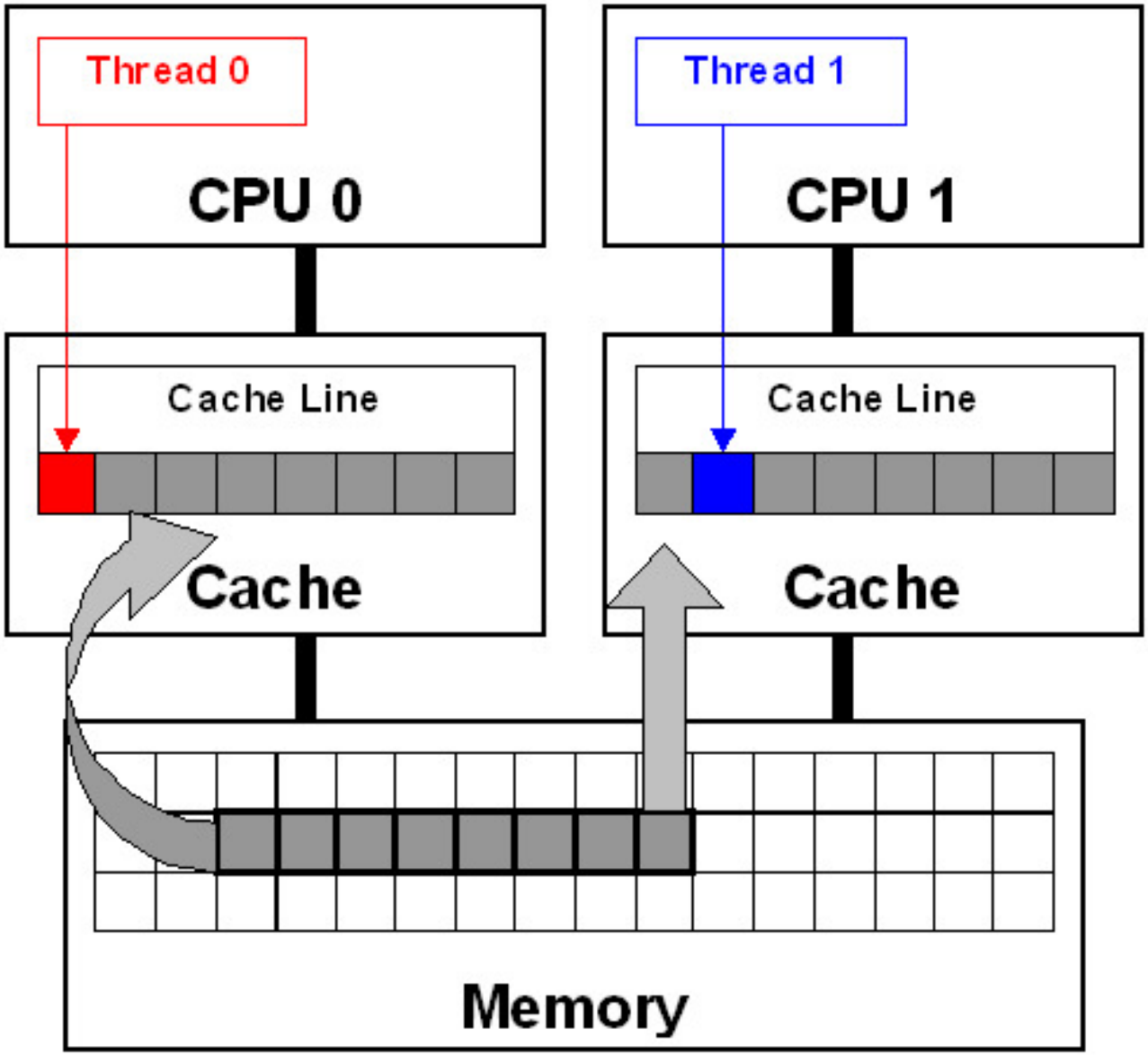
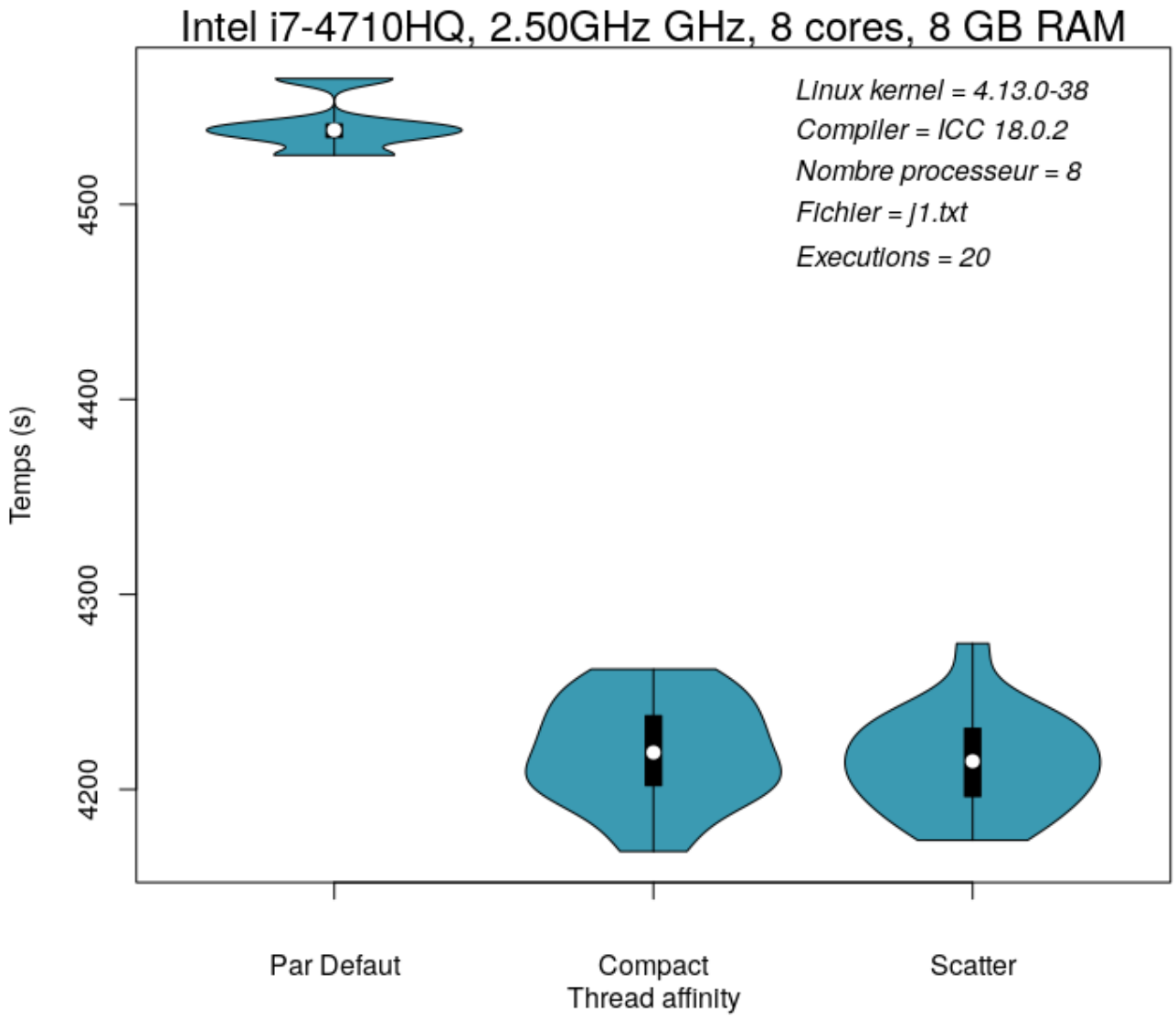


Figure – Le phénomène de False Sharing.

Source : Documentation Intel.

Thread Affinity

Figure – Analyse de l'impact du placement des threads avec le compilateur d'Intel, **-O3** avec 8 CPU.



Intel Advisor

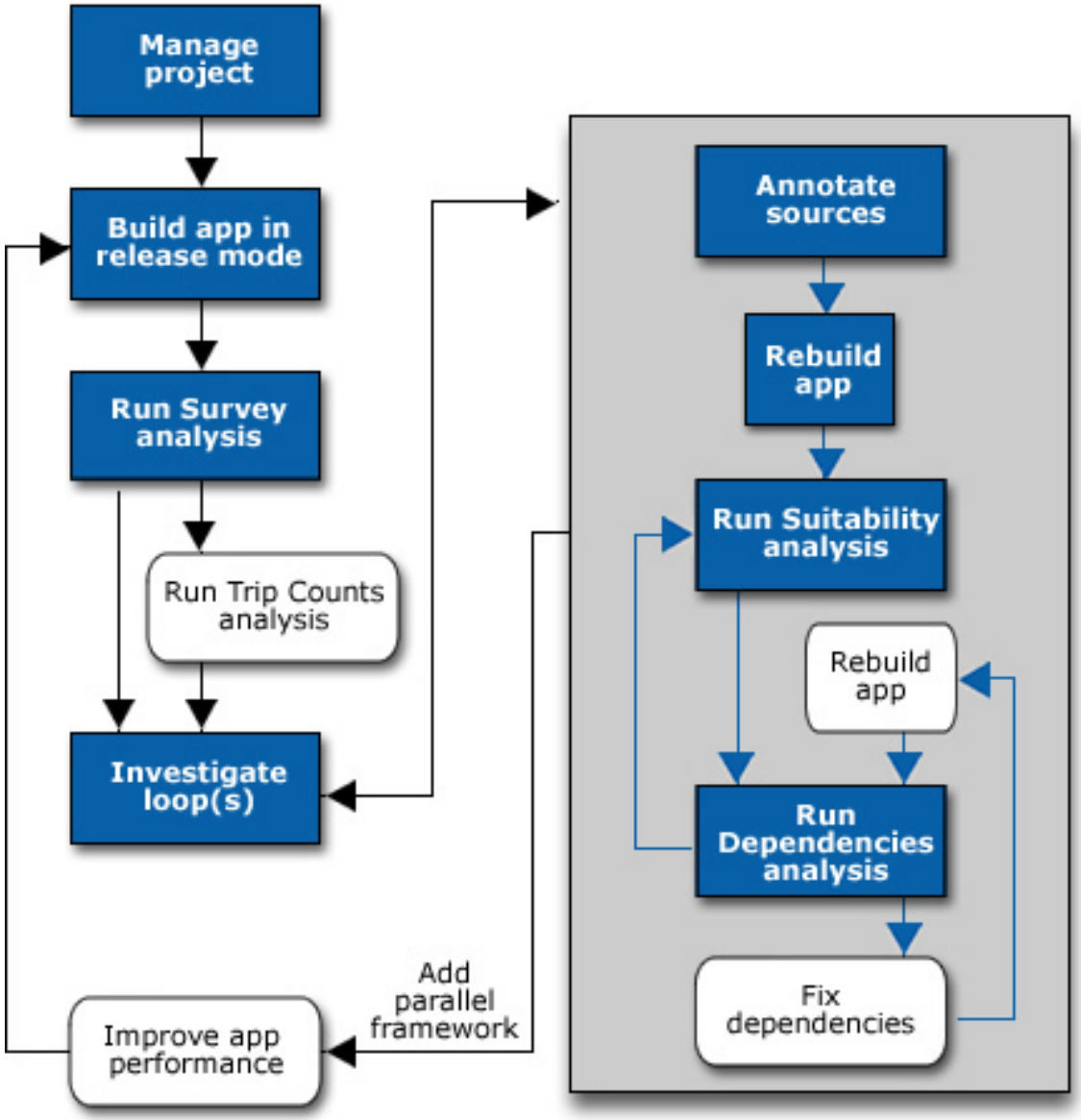


Figure – Méthode itérative pour l'ajout de parallélisme (proposée par Intel).

Source : Documentation Intel.

Difficultés de l'optimisation du code

- Algorithme très séquentiel.
- Nombre d'itérations faibles (n = 6).
- Usage de pointeurs qui pourraient aveugler les compilateurs.

Modification algorithmique

- Trier les coups à évaluer.
- Table de transposition.

Trie des coups

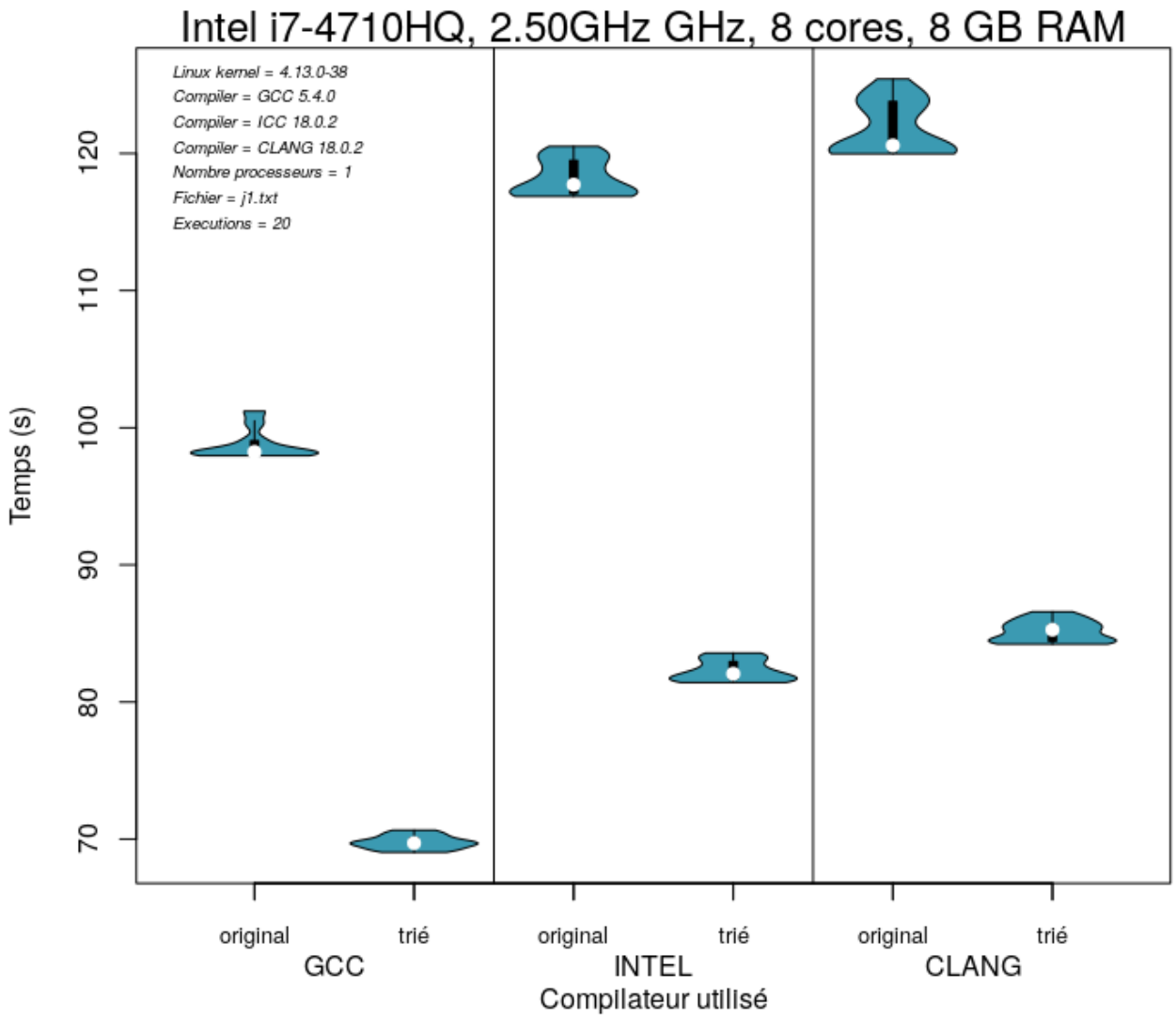


Table de transposition

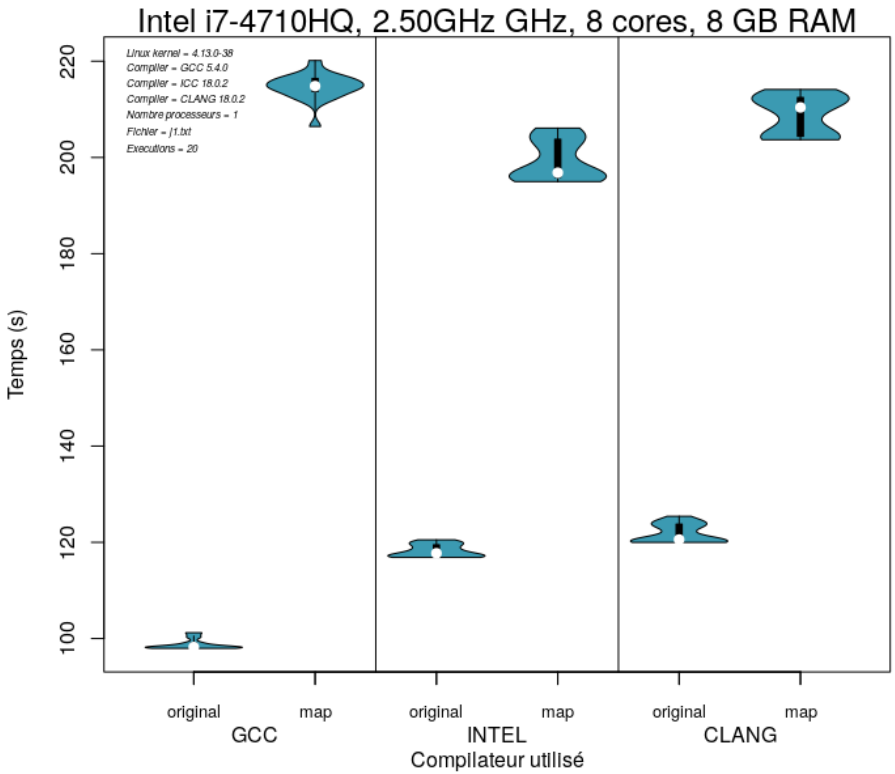


Figure – Arbre binaire de recherche.

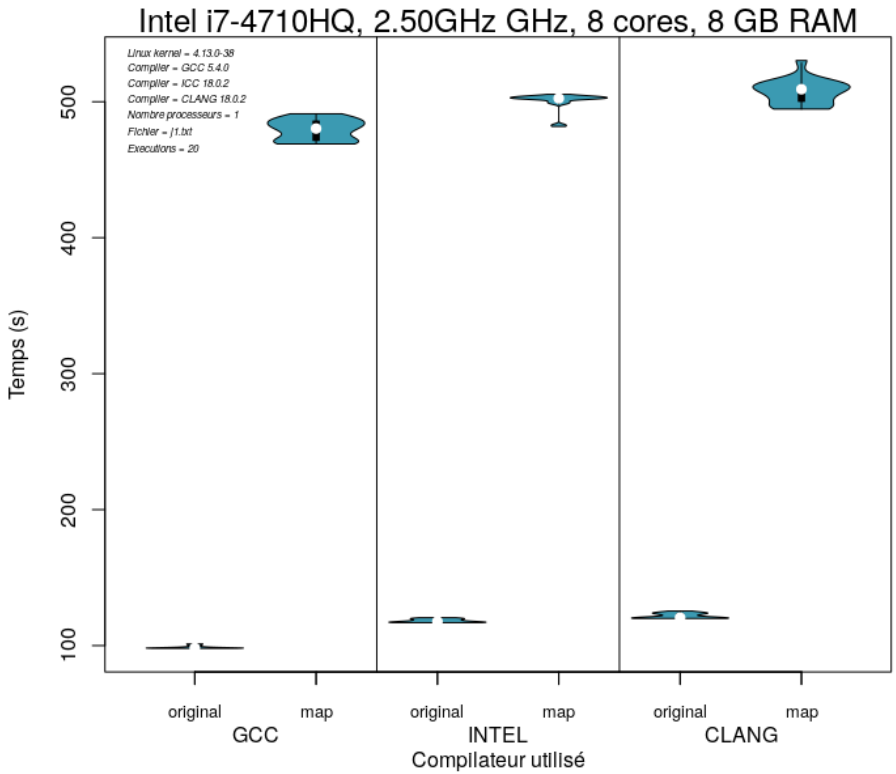


Figure – Table de hashage.

Conclusion et perspectives

- GCC bat Intel en séquentiel sur architecture Intel.
- Parallélisation avec OpenMP peut sévèrement dégrader les performances.
- Phénomène de False Sharing.
- Modifications algorithmique nécessaire pour améliorer les performances.
- Perspective : Stage à l'INRIA.