1 Objectif

Impact de la technologie hyper-threading et d'un disque SSD sur les temps de traitement dans des configurations spécifiques.

Après plus de 6 années de bons et loyaux services, j'ai décidé de changer ma machine de développement. Il faut dire que l'ancienne (Intel Core 2 Quad Q9400 2,66 Ghz – 4 coeurs – tournant sous Windows 7 64 bits) commençait à émettre des bruits inquiétants. Les ventilateurs soufflaient au maximum quasiment en permanence. J'étais obligé de mettre de la musique pour couvrir les borborygmes de la bête et pouvoir travailler en toute quiétude.

Bon, le choix de la nouvelle machine était une autre affaire. J'ai passé l'âge de la course à la puissance – qui est forcément vaine de toute manière, vu l'évolution fulgurante de l'informatique. J'étais néanmoins sensible à deux aspects que je ne pouvais pas évaluer auparavant : est-ce que la technologie hyper-threading[1] est efficace dans la programmation multithread des algorithmes de data mining ? Est-ce qu'utiliser des fichiers temporaires pour soulager l'occupation mémoire prend une autre tournure lorsqu'on s'appuie sur un disque SSD[2].

La nouvelle bête tourne sous **Windows 8.1**. Elle est équipée d'un processeur **Core I7 4770S** cadencé à 3,1 Ghz (4 cœurs physiques mais 8 threads logiques avec la technologie hyper-threading) et d'un **disque système SSD**. Ce n'est pas la gloire mais ça me permet d'évaluer les deux thèmes ci-dessus en reprenant d'anciens tutoriels que j'avais mis en ligne il y a un certain temps déjà : "Multithreading équilibré pour la discriminante" (juin 2013), où il est question d'une implémentation multithread, le nombre de threads est paramétrable, de l'analyse discriminante ; "Sipina – Traitement des très grands fichiers" (octobre 2009), où je présentais une solution qui consistait à copier les données organisées en colonnes sur disque (bien avant l'heure – la solution date de 1998, du temps où disposer de 64 Mo de RAM était éminent[3] – une version très fruste des bases de données orientées colonnes), tous les accès lors de la construction des arbres se fait sur disque, un système de cache vient accélérer la lecture.

[1] http://fr.wikipedia.org/wiki/Hyper-Threading

[2] http://fr.wikipedia.org/wiki/Solid-state_drive

[3] "La configuration type 06/1998" (espace-cubase.org)

Nous reproduisons dans ce tutoriel les deux études précitées utilisant le logiciel SIPINA. Notre objectif est d'évaluer le comportement de ces solutions (implémentation multithread, copie des données sur disque pour alléger l'occupation mémoire) sur notre nouvelle machine qui, de par ses caractéristiques, devrait en tirer expressément avantage.

2 Implémentation multithread

Dans cette section, nous reprenons un tutoriel publié en juin 2013 ("Multithreading équilibré pour l'analyse discriminante"). L'idée était de proposer une implémentation multithread de l'analyse discriminante pour exploiter au mieux les capacités des processeurs multi-cœurs qui équipent la très grande majorité des ordinateurs de bureau actuels. Une des enjeux consistait à équilibrer au mieux les charges pour éviter que certains cœurs restent en attente quand d'autres travaillent. Et, de fait, dans la solution proposée par Sipina, la réduction du temps de calcul était quasi-proportionnelle[4] au nombre de threads sollicités tant que l'on ne dépassait pas le nombre de cœurs disponibles.

Avec ma nouvelle machine (je me plais à l'écrire…), j'étais très curieux de savoir si dépasser le nombre de cœurs physiques était profitable avec la technologie hyper-threading, étant entendu que l'on ne devrait pas dépasser le nombre de cœurs logiques. Pour rappel, l'hyper-threading consiste à associer deux processeurs logiques à une seule puce, une sorte de dédoublement des capacités de calcul qui devrait mieux supporter le code multi-threadé (Wikipédia). La question posée est simple : est-ce que l'exploitation de tous les cœurs logiques permet de réduire le temps de calcul ? Dans quelle proportion ?

J'ai donc réitéré l'expérimentation sur le fichier MIT FACE IMAGES correspondant à un problème de discrimination binaire avec 513.345 lignes et 361 variables prédictives. Sur mon ancien système (Quad Core 9400 – Windows 7 – 64 bits), le temps de traitement mono thread était de 142.73 secondes[5], avec 4 threads, nous passons à 37.75 secondes (page 7) Au-delà, lorsque le nombre de threads excède le nombre de cœurs disponibles, le gain est nul (page 8).

[4] "Quasi" car une partie des calculs restait foncièrement mono-thread dans mon implémentation de l'analyse discriminante.

[5] "Multithreading pour l'analyse discriminante", mai 2013 ; page 8.

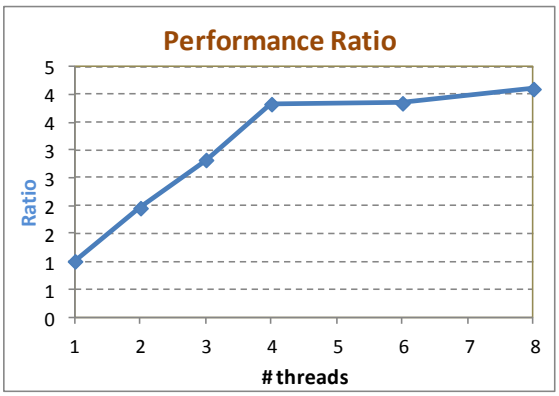
Nous réitérons la même analyse ici en augmentant graduellement le nombre de threads (1, 2, 3, 4, 6, 8). Nous surveillons l'augmentation des performances en calculant le ratio :

$$Ratio = \frac{Durée\ exec.monothread}{Durée\ exec.multithread}$$

Voyons ce que ça donne :

Threads	Durée (sec.)	Ratio
1	106.44	1.00
2	54.38	1.96
3	37.70	2.82
4	27.81	3.83
6	27.70	3.84
8	25.97	4.10

L'évolution des performances peut être représentée graphiquement.



Cette expérimentation nous inspire plusieurs commentaires :

- Tout d'abord, SIPINA détecte bien les 8 cœurs logiques lorsqu'on lui demande le nombre de processeurs disponibles.
- L'augmentation des performances (ou la réduction du temps de traitement) est quasi-proportionnel au nombre de threads utilisés, tant que l'on reste dans la limite du nombre de cœurs physiques.
- Lorsqu'on excède ce seuil (4 cœurs pour notre machine), le gain est négligeable, alors que nous restons dans la limite du nombre de cœurs logiques.

Il apparaît clairement que l'hyper-threading n'apporte rien dans notre implémentation de l'analyse discriminante. Hélas, mes compétences informatiques s'arrêtent à ce stade. Surtout que

plusieurs éléments s'entremêlent ici : la sollicitation du processeur certes, mais aussi l'accès aux données en mémoire, etc. Le phénomène de dégradation des performances de l'hyper-threading est abordé dans plusieurs articles sur le web[6]. Ce résultat est confirmé par une autre expérimentation que j'ai menée sur un portable équipé d'un processeur Core i5 double cœur[7], mais à 4 threads.

De la même manière, j'ai réitéré mon expérimentation pour l'algorithme multithread de construction des arbres de décision[8] implantée dans Sipina. Dans ce contexte également, le passage de 4 à 8 threads n'améliore pas significativement la durée d'exécution.

3 Fichiers temporaires sur disque SSD - Arbres

Dans cette section, je reprends un tutoriel d'octobre 2009 ("Sipina – Traitement des très grands fichiers"). L'idée était d'utiliser des fichiers temporaires pour soulager l'occupation mémoire. Les accès se font donc directement sur disque lors de l'exécution des algorithmes de data mining c.-à-d. durant la phase de construction des modèles. Un système de cache vient compléter le dispositif pour améliorer le temps de lecture. Nous avions utilisé la version 3.2 de Sipina.

Avec la technologie SSD (Solid-state drive)[9], les disques durs ne sont plus composés d'éléments mécaniques. Les performances effectuent un bond considérable. Dans notre cas, nous nous intéressons avant tout à la vitesse de lecture des données stockées dans des fichiers temporaires. Situé dans un contexte de construction des arbres de décision, j'étais très curieux de voir quels seraient les gains par rapport à un stockage sur disque standard.

Depuis la version 3.8, Sipina est doté du gestionnaire de mémoire FASTMM qui lui permet d'adresser plus de mémoire. Par rapport à notre tutoriel de référence, il est possible maintenant de charger l'ensemble de données (9.634.198 observations, 41 variables prédictives + la classe à prédire) en mémoire. Nous pouvons ainsi comparer les temps de traitement lorsque les données sont en totalité chargées en mémoire et lorsque nous les stockons dans des fichiers

[6] Voir par exemple "When hyper-threading hurts", https://bitsum.com/pl_when_hyperthreading_hurts.php.

[7] http://fr.wikipedia.org/wiki/Intel_Core_i5

[8] "Multithreading pour les arbres de décision", novembre 2010.

[9] http://fr.wikipedia.org/wiki/Solid-state_drive

sur disque. L'expérimentation menée sur mon ancien PC doté d'un disque dur standard est dupliquée sur ma nouvelle machine (je ne me lasse vraiment pas de l'écrire...) dotée d'un disque système SSD. Nous verrons ainsi si le passage au SSD s'avère véritablement bénéfique lorsque les traitements impliquent des accès répétés sur disque. Nous calculons le ratio :

$$Ratio = \frac{Durée\ exec.sur\ disque}{Durée\ exec.en\ mémoire}$$

Nous résumons les résultats dans le tableau ci-dessus.

Durée (sec.)	Q9400 + Std HD	Core i7 + SSD HD
Traitement mémoire	210.5	89.9
Traitements sur fichiers temporaires	271.4	114.6
Ratio	**1.3**	**1.3**

Quelques remarques :

- Le nouveau processeur est à peu près 2 fois plus puissant que l'ancien en calcul mono-thread. C'est ce que j'avais observé sur les sites de benchmark lorsque je comparais les différentes configurations[10].
- Effectuer les calculs à partir des données stockées sur disque affecte les performances par rapport au traitement en mémoire. C'est entendu. Mais nous constatons que la dégradation reste très raisonnable compte tenu du volume à manipuler.
- Le ratio est le même (≈ 1.3) que l'on travaille sur disque SSD ou disque standard. Là j'avoue que je ne m'y attendais pas du tout. Soit le SSD, c'est vraiment du pipeau ; soit la stratégie que j'ai mise en place pour accélérer les traitements lors de la construction des arbres ne permet pas de tirer parti de ses performances.

Pour comprendre ce comportement pour le moins étonnant, j'ai scruté le code source de Sipina. Je me suis rendu compte que la variable cible était toujours en mémoire durant la construction de l'arbre. Ensuite, à chaque évaluation des variables candidates de segmentation, je charge temporairement la colonne entière en mémoire. On comprend pourquoi il est si important d'organiser les données en colonnes dans ce contexte. Le coût supplémentaire (le ratio de 1.3) vient tout simplement des chargements et déchargements successifs des blocs. L'occupation

[10] http://www.cpubenchmark.net/singleThread.html, pour les processeurs seuls. La vitesse mémoire joue aussi, etc.

mémoire reste contenue puisqu'au pire, nous chargeons deux colonnes entières de valeurs. En revanche, les performances accrues du disque SSD ne lui permettent pas de se démarquer puisqu'à l'accès au disque est réduite au minimum avec la lecture en blocs des colonnes.

4 Conclusion

Je suis très mitigé à l'issue de cette expérimentation. Manifestement, je dispose d'une nouvelle machine plus puissante, les temps de calculs comparés en attestent sans problème (ouf ! je ne me voyais pas vraiment la ramener chez le revendeur). En revanche, les caractéristiques spécifiques qui me faisaient saliver d'avance se sont avérées décevantes. D'un côté, l'hyper-threading ne m'a pas permis de profiter de la puissance accrue en spécifiant plus de threads que de cœurs physiques (tout en restant dans les limites du nombre de cœurs logiques) pour les algorithmes multithreadés que j'ai programmé dans Sipina. De l'autre, le disque système SSD utilisé pour le stockage des fichiers temporaires ne se démarque pas dans mon implémentation des arbres de décision.

Les conclusions sont certes mitigées. Le fond de l'affaire, lui, demeure. La technologie informatique évolue très vite et nous offre sans cesse des nouvelles opportunités. Il est de notre rôle, chercheurs informaticiens, d'essayer d'en tirer parti dans nos implémentations des algorithmes de data mining afin de rendre nos programmes plus performants.

Enfin, last but not least, ma brave vieille machine n'ira pas eu rebut, loin s'en faut. Elle me servira dorénavant de PC de test pour évaluer la viabilité et les performances des innombrables idées plus ou moins farfelues qui me traversent l'esprit. Il nous reste encore de belles choses à vivre ensemble.