Découverte d'itemsets fréquents fermés sur architectures multicœurs

Benjamin Négrevergne*, Alexandre Termier*, Jean-François Méhaut*, Takeaki Uno**

*LIG - UJF-CNRS UMR 5217 - 681 rue de la Passerelle, B.P. 72, 38402 Saint Martin d'Hères, France
{Benjamin.Negrevergne, Jean-Francois.Mehaut, Alexandre.Termier}@imag.fr, http://www.liglab.fr
**National Institute of Informatics, 2-1-2, Hitotsubashi, Chiyoda-ku, 101-8430 Tokyo, Japan
uno@nii.jp, http://research.nii.ac.jp/∼uno/

Résumé. Dans ce papier nous proposons PLCM, un algorithme parallèle de découverte d'itemsets fréquents fermés basé sur l'algorithme LCM, reconnu comme l'algorithme séquentiel le plus efficace pour cette tâche. Nous présentons aussi une interface de parallélisme à la fois simple et puissante basée sur la notion de *Tuple Space*, qui permet d'avoir une bonne répartition dynamique du travail.

Grâce à une étude expérimentale détaillée, nous montrons que PLCM est le seul algorithme qui soit suffisamment générique pour calculer efficacement des itemsets fréquents fermés à la fois sur des bases creuses et sur des bases denses, améliorant ainsi l'état de l'art.

1 Introduction

La découverte de motifs fréquents est l'un des domaines majeurs de la fouille de données. A l'origine de ce domaine se trouvent les travaux d'Agrawal et Srikant (1994) sur la découverte d'itemsets fréquents. Le problème de la découverte d'itemsets fréquents consiste, étant donné une base de données où les transactions sont des listes d'*items* et un seuil de support minimal $minsup$, à découvrir tous les *itemsets* qui apparaissent plus de $minsup$ fois dans la base de de données. Ce problème est le plus simple du domaine de la découverte de motifs fréquents, comparativement à la recherche de séquences, d'arbres ou de graphes fréquents. Toutefois, il se retrouve dans de nombreuses applications, en particulier dans l'analyse de données de vente d'organisations commerciales. De plus, de part sa relative simplicité, toutes les améliorations algorithmiques significatives en découverte de motifs fréquents ont d'abord été réalisées dans le cas des itemsets fréquents avant d'être adaptées à des motifs plus complexes. C'est en particulier le cas pour la fermeture (Pasquier et al. (1999)) ou les méthodes sans génération de candidats (Han et al. (2000)).

En 2003 et 2004, le workshop FIMI (Goethals (2004)) a confronté les algorithmes de recherche d'itemsets fréquents afin de déterminer les meilleures solutions. Le gagnant de FIMI 2004, LCM2 de Uno et al. (2004b), combine des améliorations de haut niveau issues de la théorie de l'énumération, et des optimisations de bas niveau optimisant la répartition entre

temps de calcul et mémoire utilisée. Depuis 2004, aucun algorithme séquentiel n'a présenté de meilleures performances.

Toutefois les algorithmes séquentiels présentés à FIMI en 2004 ne sont pas capables d'exploiter le potentiel de parallélisme des processeurs modernes. En effet depuis 2005, la conception des processeurs d'ordinateurs a radicalement changé. Des limites physiques ont été atteintes, ne permettant plus d'augmenter la fréquence d'horloge et donc les performances. Par contre la loi de Moore est toujours valable, ce qui implique que tous les 18 mois le nombre de transistors que l'on peut graver sur un *die* double. Afin de permettre une augmentation des performances, les chercheurs et constructeurs ont donc proposé la solution de mettre plusieurs *cœurs* de calcul sur un même composant physique (Chip). Pour exploiter au mieux ces processeurs multicœurs, il est impératif de concevoir des algorithmes parallèles, pour lesquels les choix algorithmiques faits pour des algorithmes séquentiels ne sont plus nécessairement les mieux adaptés.

L'objectif de cet article est de présenter un algorithme parallèle d'extraction d'itemsets fréquents fermés basé sur LCM et capable d'exploiter efficacement les processeurs multicœurs des ordinateurs actuels, que l'on retrouve sur les bureaux de la majorité des chercheurs et des analystes de données. Nos expériences se focaliseront donc sur des machines mono-processeur multicœurs. Nous présentons une nouvelle interface de parallélisation basée sur la notion de *Tuple Space*. Cette interface expose deux primitives à l'implémenteur de l'algorithme, mais permet d'utiliser en internes des modèles efficaces de distribution du calcul. Nous présentons ensuite l'algorithme PLCM, qui utilise cette interface pour paralléliser LCM.

Le plan de l'article est le suivant : Dans la section 2, nous définissons brièvement le problème de l'extraction d'itemsets fréquents fermés et présentons l'état de l'art en extraction parallèle de motifs fréquents sur multicœurs. Nous décrivons dans la section 3 la méthode des Tuple Spaces et l'algorithme parallèle PLCM. La section 4 présente une étude expérimentale détaillée comparant PLCM à l'état de l'art. Enfin, dans la section 5, nous concluons et dressons quelques perpectives pour des travaux futurs.

2 Les principes de l'extraction d'itemsets fréquents fermés

Dans cette section, nous introduisons les principales notations utilisées dans l'article, et nous définissons le problème de l'extraction d'itemsets fréquents fermés. Nous présentons ensuite l'état de l'art.

2.1 Position du problème

Soit $\mathcal{I} = \{1, ..., n\}$ un ensemble d'*items*. Une base de données transactionnelle sur $\mathcal{I}$ est un ensemble $T = \{t_1, ..., t_m\}$ tel que chaque t_i est inclus dans $\mathcal{I}$. Un t_i est appelé *transaction*. Un sous-ensemble P de $\mathcal{I}$ est appelé un *itemset*. Pour un itemset P, une transaction contenant P est appelée une *occurrence* de P. La *tid-list* de P, notée $tidlist(P)$, est l'ensemble des occurrences de P. $|tidlist(P)|$ est appelé le *support* de P, ou encore la *fréquence* de P, également noté $support(P)$.

Pour un seuil minimal de support donné $minsup$, un itemset P est *fréquent* si $support(P) \geq minsup$. On notera $\mathcal{F}$ l'ensemble des itemsets fréquents.

Pour toute paire d'itemsets P et Q, on dira que P et Q sont équivalents si $tidlist(P) = tidlist(Q)$. Cette relation partitionne les itemsets en classes d'équivalences. Les itemsets maximaux au sens de l'inclusion de chaque classe d'équivalence sont appellés itemsets *fermés*. On notera $\mathcal{C}$ l'ensemble des itemsets fréquents fermés.

Le problème qui nous intéresse est, étant donné une base de donnée transactionnelle $\mathcal{T}$ et un seuil de support minimal $minsup$, d'extraire tous les itemsets fréquents fermés de $\mathcal{T}$.

2.2 État de l'art

De nombreux travaux ont été menés à la fin des années 1990 sur l'extraction parallèle de motifs fréquents sur des clusters (Agrawal et Shafer (1996); Zaki et al. (1997)). Ces travaux se concentrent à la fois sur la capacité d'augmenter la taille des bases de données traitées que sur l'amélioration des performances. Nous ne faisons que les mentionner, et nous focaliserons cet état de l'art sur l'extraction parallèle de motifs fréquents sur des architectures multicœurs, donc les problématiques sont sensiblement différentes.

Le seul algorithme existant dans la littérature pour l'extraction d'itemsets fréquents fermés a été proposé par Lucchese et al. (2007). Leur stratégie de parallélisme se base sur une technique de *vol de travail* (Blumofe et Leiserson (1999)) : lorsqu'un thread a beaucoup de travail et qu'un autre n'en a pas, ce dernier peut "voler" une partie du travail du premier thread, réduisant ainsi le déséquilibre de charge. Leur solution contient des optimisations pour améliorer l'utilisation du cache lors de la création de bases de données conditionnelles (appellées *projections* dans leur article), une technique couramment utilisée en extraction d'itemsets fréquents.

3 PLCM : LCM en parallèle

Dans cette section, nous expliquons le modèle de parallélisation que nous avons adopté et nous présentons l'algorithme PLCM. Faute de place, nous invitons le lecteur intéressé à se référer à l'article de Uno et al. (2004a) pour l'explication de l'algorithme séquentiel LCM.

3.1 Modèle de distribution du calcul

LCM est un algorithme récursif dont les appels ont une structure d'arbre. Ce type d'algorithme présente des qualités intéressantes pour la parallélisation. En effet, les nœuds de même niveau peuvent être calculés de manière indépendante, sans besoin de synchronisation complexe. Le problème est que dans l'extraction de motifs fréquents en général et pour LCM en particulier, les itérations récursives ont des temps de calcul très variables. Une distribution statique du travail engendrerait un très fort déséquilibre de charge, comme cela a été montré dans Lucchese et al. (2007).

Comme les autres travaux de l'état de l'art, nous proposons donc une approche de répartition dynamique du travail. Notre interface est basée sur le modèle Linda proposé par Gelernter (1989). Dans ce modèle les threads communiquent par le biais d'un espace de mémoire partagée appelé *Tuple Space* auquel ils peuvent ajouter ou retirer des *tuples*. Les deux primitives de notre interface de parallélisation sont donc simplement : $put(tuple)$ et $get(tuple)$. Ces deux primitives assurent à elles seules une répartition dynamique du travail. Tant qu'il y a des tuples et donc du travail les threads les récupèrent avec la primitive get. Une itération qui génère du

travail peut le distribuer en insérant des tuples dans le tuple space. Le modèle de distribution du calcul depend de la manière dont sont organisés les tuples au sein du tuple space. Les capacités de passage à l'échelle d'un programme utilisant le tuple space dépendront donc du coût de la gestion du tuple space.

Notre implémentation du tuple space stoque les tuples dans N "bancs de travail", où N est le nombre de threads utilisés. Le fait d'avoir un banc de travail assigné à chaque thread permet de limiter la contention au moment des appels aux primitives put et get. Chaque thread ajoute et consomme des tuples dans un banc qui lui est propre. Lorsque le banc d'un thread est vide, le tuple space lui donne des tuples d'un autre banc. Il s'agit d'une forme de *vol de travail*, qui est directement gérée par le tuple space et est transparente pour l'algorithme. Bien que des opérations de synchronisation soient nécessaire sur cette architecture de tuple space, il n'y a généralement pas de contention. Dans la plupart des cas, les threads calculent les tuples qu'ils ont eux même engendrés, ce qui améliore l'utilisation du cache.

3.2 Parallélisation de LCM

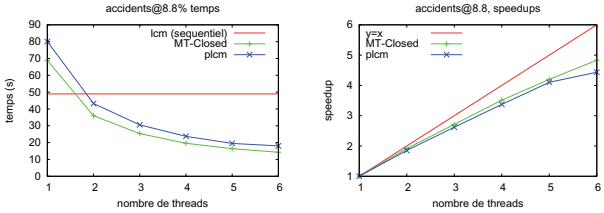
Dans le cas de PLCM un tuple contient les paramètres d'un appel récursif. La parallélisation se fait donc sur l'arbre des appels récursif : chaque appel récursif est remplacé par une opération put qui dépose dans le tuple space un tuple avec les paramètres de cet appel. Le programme principal ne fait qu'initialiser le tuple space avec un tuple contenant les paramètres correspondant à la racine de l'arbre des appels récursifs, puis il démarre N threads de traitement. Chaque thread fait une boucle qui récupère un tuple avec la procédure get, et effectue l'appel de la fonction de traitement avec les paramètres correspondants.

Afin de limiter le nombre de tuple et donc les surcoûts liés à leur gestion, à partir d'une certaine profondeur dans l'arbre de recherche les appels récursifs sont traités directement, sans créer de nouveaux tuples.

4 Expériences

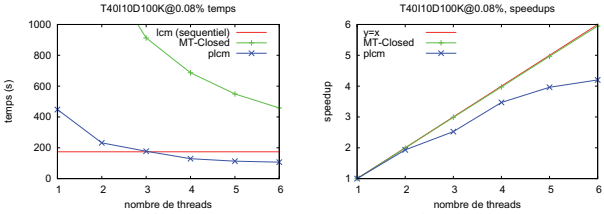
Nous comparons dans cette section PLCM avec MT_Closed (Lucchese et al. (2007)), qui est la seule autre approche parallèle d'extraction d'itemsets fréquents fermés. L'implémentation de PLCM est notre implémentation en C++ . Cette implémentation ne partage pas de code avec l'implémentation originale de LCM en C, elle a été entièrement réécrite en utilisant des Tuple Spaces. L'API des Tuple Spaces est écrite en C++ et Posix Threads. L'implémentation de MT_Closed est l'implémentation originale de ses auteurs en C++. A titre de référence, nous avons également comparé nos temps de calcul avec l'implémentation séquentielle en C de LCM. Les expériences sont menées sur un Intel Xeon 7460 à 2.66 GHz avec 12 Go de RAM. Ce processeur a 6 cœurs. Les bases de données utilisées sont les bases fournies par le workshop FIMI, disponibles sur le site web du workshop (Goethals (2004)). Nous avons effectués des tests sur toutes les bases du site FIMI, mais par manque de place nous ne reportons ici dque les résultats pour la base de données *dense* accidents, et la base de données *creuse* T10I4D100K. Ces résultats sont présentés dans les Figures 1 et 2 en terme de temps de calcul et de speedup.

Nous pouvons remarquer que pour la base de données dense, MT_closed présente de meilleurs temps d'exécution que PLCM. Ceci s'explique par le fait que MT_closed utilise



(a) Execution time (b) Speedup

FIG. 1 – *Résultats pour accidents à 8.8% (dense)*



(a) Execution time (b) Speedup

FIG. 2 – *Résultats pour T10I4D100K à 0.08% (creux)*

une représentation *bitmap* des transactions, très efficace sur les bases de données denses. Pour améliorer encore cette efficacité, MT_Closed utilise les instructions SIMD du processeur pour faire les calculs sur les bitmaps, ce qui d'après les expériences de Lucchese et al. (2007) peut amener un speedup de 50%. Par contre, PLCM présente d'aussi bonnes capacité de passage à l'échelle que MT_Closed sur ce type de bases de données.

En revanche, sur la bases de données creuse, PLCM est au minimum deux ordres de grandeur plus rapide que MT_Closed. Cette différence est dûe aux différences algorithmiques entre MT_Closed et PLCM : il y a peu d'itemsets fréquents fermés dans ce jeu de données, donc comme PLCM est linéaire en fonction du nombre d'itemsets fréquents fermés, son temps d'exécution est faible.

Notre stratégie de parallélisation permet donc de profiter pleinement de l'avantage en terme de complexité théorique qu'a LCM sur MT_Closed, tout en exploitant les performances d'un processeur multicœur.

5 Conclusion et perspectives

Nous avons présenté dans cet article une interface simple de parallélisation d'algorithmes, et l'algorithme PLCM, qui utilise cette interface pour paralléliser l'algorithme LCM. L'intérêt de la méthode de parallélisation que nous présentons et qu'elle est très générale, et peut donc être appliquée à d'autres algorithmes d'extraction de motifs fréquents.

Les expériences sur les bases de données de référence du domaine ont montré que notre algorithme donnait de bons résultats aussi bien sur les bases de données creuses que sur les bases de données denses, améliorant ainsi l'état de l'art.

Nous prévoyons de nous intéresser aux machines multi-processeurs multicœurs afin de passer à l'échelle sur plus de cœurs. Dans le cas de ces machines, de nouveaux problèmes apparaissent, en particulier la saturation du bus entre la mémoire et les cœurs.

Références

Agrawal, R. et J. C. Shafer (1996). Parallel mining of association rules. *IEEE Trans. Knowl. Data Eng. 8*(6), 962–969.

Agrawal, R. et R. Srikant (1994). Fast algorithms for mining association rules. In *Proceedings of the 20th VLDB Conference*, pp. 487–499.

Blumofe, R. D. et C. E. Leiserson (1999). Scheduling multithreaded computations by work stealing. *J. ACM 46*(5), 720–748.

Gelernter, D. (1989). Multiple tuple spaces in linda. pp. 20–27.

Goethals, B. (2003-2004). Fimi repository website. `http://fimi.cs.helsinki.fi/`.

Han, J., J. Pei, et Y. Yin (2000). Mining frequent patterns without candidate generation. In *SIGMOD'00 : Proceedings of the International Conference on Management of Data*, pp. 1–12.

Lucchese, C., S. Orlando, et R. Perego (2007). Parallel mining of frequent closed patterns : Harnessing modern computer architectures. In *ICDM*, pp. 242–251.

Pasquier, N., Yves, Y. Bastide, R. Taouil, et L. Lakhal (1999). Efficient mining of association rules using closed itemset lattices. *Information Systems 24*, 25–46.

Uno, T., T. Asai, Y. Uchida, et H. Arimura (2004a). An efficient algorithm for enumerating closed patterns in transaction databases. In *Discovery Science*, pp. 16–31.

Uno, T., M. Kiyomi, et H. Arimura (2004b). Lcm ver. 2 : Efficient mining algorithms for frequent/closed/maximal itemsets. In *FIMI*.

Zaki, M. J., S. Parthasarathy, M. Ogihara, et W. Li (1997). Parallel algorithms for discovery of association rules. *Data Min. Knowl. Discov. 1*(4), 343–373.

Summary

In this paper we present PLCM, a parallel algorithm for discovering closed frequent itemsets, based on the LCM algorithm, recognized as the most efficient serial algorithm for this task. We also present a simple yet powerfull parallelism interface based on the concept of *Tuple Space*, which allows an efficient dynamic sharing of the work.

Thanks to a detailed experimental study, we show that PLCM is the only algorithm which is generic enough to compute efficiently closed frequent itemsets both on sparse and dense databases, thus improving the state of the art.