行政院國家科學委員會專題研究計畫 成果報告

設計與建構一個具有多因子及自我學習能力排程演算法之高效能網格計算環境

研究成果報告(精簡版)

計 畫 類 別 ：	個別型
計 畫 編 號 ：	NSC 97-2221-E-216-020-
執 行 期 間 ：	97 年 08 月 01 日至 98 年 07 月 31 日
執 行 單 位 ：	中華大學資訊工程學系

計 畫 主 持 人 ： 游坤明
共 同 主 持 人 ： 唐傳義
計畫參與人員： 碩士班研究生-兼任助理人員：吳秉璋
博士班研究生-兼任助理人員：周嘉奕

報 告 附 件 ： 出席國際會議研究心得報告及發表論文

處 理 方 式 ： 本計畫可公開查詢

中 華 民 國 98 年 10 月 22 日

行政院國家科學委員會補助專題研究計畫 ■ 成 果 報 告
□ 期中進度報告

設計與建構一個具有多因子及自我學習能力排程演算法之高效能網格計算環境

計畫類別：■ 個別型計畫　　□ 整合型計畫
計畫編號：NSC　97-2221-E-216-020
執行期間：　97 年 8 月 1 日至　98 年 7 月 31 日

計畫主持人：游坤明
共同主持人：唐傳義
計畫參與人員：周嘉奕、吳秉璋

成果報告類型(依經費核定清單規定繳交)：■精簡報告　□完整報告

本成果報告包括以下應繳交之附件：
□赴國外出差或研習心得報告一份
□赴大陸地區出差或研習心得報告一份
■出席國際學術會議心得報告及發表之論文各一份
□國際合作研究計畫國外研究報告書一份

處理方式：除產學合作研究計畫、提升產業技術及人才培育研究計畫、列管計畫及下列情形者外，得立即公開查詢
□涉及專利或其他智慧財產權，□一年□二年後可公開查詢

執行單位：中華大學資訊工程學系

中　華　民　國　98　年　10　月　20　日

設計與建構一個具有多因子及自我學習能力排程演算法之高效能網格計算環境

游坤明

中華大學資訊工程學系

摘要

隨著電腦科學的發展，資訊處理以及分析在各研究領域扮演了相當重要的角色。雖然根據莫耳定律提到，電晶體的數目平均每兩年會成長一倍，但是計算量的需要卻成長的相對快速。於是藉由系統匯流排，區域網路或網際網路連接多個計算資料是目前的主流，叢集及網格系統便是典型的高效能計算平台。本計畫主要運用上述高效能計算平台對生物資訊及資料探勘領域中兩個重要的問題設計平行演算法，分別為平行化合物推論以及平行頻繁項目集探勘。化合物推論的問題目標為列舉出所有相同特徵的化合物，在計畫執行中，我們在主從架構的叢集系統上設計並實作該平行演算法。並且採用兩階段的搜尋策略，主節點可以先運用 bread-first search 產生適量的候選項目後，將候選項目分配給從屬節點後使用 depth-first search 找到所有的解。頻繁項目集探勘的目標是在一個給定的交易資料庫中找到所有符合給定支持度的高頻項目集。我們亦順利在叢集系統及網格系統設計平行演算法，實驗結果觀察到所設計的方法在叢集系統減少計算所需時間，而在網格系統亦可以平衡異質節點間的計算負載。計畫進行中所得到的研究成果已發表一篇國際期刊論文於 Expert System with Application (SCI)，三篇國際研討會論文(兩篇為 EI)，以及一篇國內研討會論文。

Keywords: 分支與界定, 叢集運算, 網格運算, 化合物推論, 頻繁項目

Abstract

With the development of computer science, information processing and data analysis play an important role in various research areas. According to Moore’s law, describing a long-term trend of computing hardware, the number of transistors can be doubling approximately every two years. With the growth of the amount of data, the computing power also needed to grow up. Therefore, connect multiple computing units through system bus, interconnection network, or internet to produce high throughput computing power is currently the trend. Cluster system and grid system are the typical high performance computing platform. This project used cluster or grid system to solve two important problems in Bioinformatics and Data Mining named Chemical Compound Inference (*CCI*) problem and Frequent Pattern Mining (*FPM*) problem respectively. The goal of *CCI* is to enumerate all chemical compounds having the same characteristics; we developed our parallel algorithm on cluster system constructed of master/slave architecture. Proposed method used two stages searching strategy, in the first stage, bread-first search was used to generate set of candidate items on master node. Then each participated slave node use depth-first search to generate all possible solutions after received candidate items from master node. The goal of *FPM* is to find all frequent patterns confirmed to given support from a transactional database. We also designed and implemented two parallel algorithms for *FPM* problem on cluster and grid system. The results illustrated proposed method could reduce the computation time on cluster system and could balance the workload among participated nodes on grid system. Finally, we sum up our works to publish an international journal paper and four conference papers.

Keywords: branch and bound, cluster computing, grid computing, chemical compound inference, frequent patterns

1 緣由與目的

連結計算資源成為一大型高效能資源是叢集運算 (Cluster Computing) 及網格運算 (Grid Computing) 的基本概念。與傳統叢集電腦 (Cluster Computing) 不同之處，網格可藉由共同的規範將座落於不同網域及地理空間的叢集系統、個人電腦乃至於行動計算裝置的資源整合成一虛擬的高效能計算平台。網格運算是一個鬆散整合的分散式系統，參與網格的計算資源稱為計算節點 (Computing Node)，在計算節點的作業系統之上藉由中介軟體 (middleware) 將不同節點連結在一起，允許它們共用資源，並形成一個虛擬組織 (virtual organization)。共用計算能力、儲存能力讓使用者可得到超越單一設備極限的工作能力。

許多生物資訊及資料探勘領域的問題都已被證明為 NP 問題，例如多序列比對(Multiple Sequence Alignment)，最短共同子字串(Shortest Common Superstring)，最小權值等距演化樹(Minimum Ultrametric Tree)，等。運用單一電腦是不可能在多項式時間內得到這些問題的最佳解。尤其當輸入資料增加時，平行處理是解決該類問題的一個主要策略，本計畫分別探討在生物資訊及資料探勘中重要的問題並設計平行演算法。分別為 (1) 化合物推論問題 (Inference of a chemical structure from path frequency) 以及 (2) 頻繁項目集探勘 (Frequent patterns mining)。

化合物推論 (Chemical Compounds Inference, *CCI*) 在生物資訊中是一個重要的問題，該問題主要目的是列舉出所有擁有相同特性的化合物。應用上包含了 structure determination using mass-spectrum [1; 2], reconstructing molecular structure with given signatures [3; 4], 以及 classification of compounds [5] 等。從特徵空間 (特徵值) 尋找相對應的輸入空間 (化合物) 是一個 pre-image 問題。我們要解決的問題亦已被證明為 NP-hard，利用叢集及網格系統的計算能力來降低計算時間是本計畫的研究目標。

本計畫亦於叢集及網格系統中加入資料探勘領域中一個相當重要的研究應用--頻繁項目集的探勘。頻繁項目集是許多重要資料探勘研究的基礎，如關聯式法則、時間序列、分類式法則、群組式法則…等。然而資料庫的交易量很多時，或者給定的最小支持度很小時，探勘的時間往往會成長非常快速。於是，平行化技術可以讓探勘所需的計算分散在各個計算節點，並且降低計算時間。該應用與之前兩個生物資訊的應用最大的不同在於前兩個應用是 CPU-bound，計算節點的計算能力決定了效能。然而找尋頻繁項目集不單是需要高效的計算能力，還有大量的資料需要從資料庫載入以及在計算過程中亦會有許多資料需要透過網路連線交換。於是，一個考慮到網路頻寬、磁碟存取速度、儲存空間的排程演算法可以讓探勘頻繁項目集的應用有更高效能的表現

2 研究方法與成果

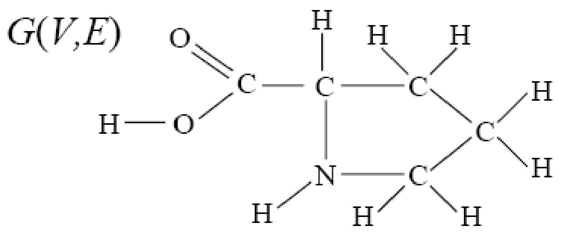
本計畫已完成項目為：(1) 設計並實作化合物推論演算法於叢集系統中，(2) 設計並實作頻繁項目集探勘演算法於叢集及網格系統。

2.1 平行化合物推論

對化合物推論問題 (Chemical Compound Inference Problem, CCI) 我們提出一 Parallel Branch-and-Bound Chemical Compound Inference with Path Frequency (PB-CIPF) 演算法。首先我們先定義問題本身：

Definition 1. Let G(V,E) be an undirected vertex-labeled connected graph and ∑ be a set of vertex labels.

由於要解決的問題是化合物，因此可以合理假設圖上最大 degree 是一個常數值，鍵結數是由原子所決定。圖 1 為一個 feature vector 的例子。f_1(G) 代表的就是 G(V,E) 這個圖的 path frequency.



	C	N	O	H	CC	CN	CO	CH	NC	NN	• • •	OH	• • •
$f_1(G)$	5	1	2	9	8	2	2	7	2	0	• • •	1	• • •

圖 1 Examples of feature vectors

利用 path frequency 推論回化合物的問題稱為 Chemical Compound Inference (CCI)。定義如下：

Definition 2. Given a feature vector *v* of level *K*, output a graph G(V,E) satisfying $f_K(G) = v$ and $\sum_{w:\{v,w\}\in E} m(\{v,w\}) \le val(l(v))$ for all $v \in V$. If there does not exist such G(V,E), output “no solution”.

因為 CCI 問題已被證明為 NP-hard 的問題，於是我們採用平行處理的技術來降低計算所需的時間。本計畫採用叢集系統做為平行程式執行的平台，叢集系統的主要特性為計算節點的硬體規格相同，於是在負載平衡的設計上不需要考慮節點的異質特性。圖 2 描述了我們的方法的概念，可以將圖 2 視為一分支與界定樹 (branch and bound tree)，PB-CIPF 在執行期間分成 BFS 以及 DFS 兩個階段。由於叢集系統中，我們採用的是標準的主從架構，在第一階段時，主節點 (master node) 採用 bread-first search (BFS) 的搜尋策略，當產生了一定數量的候選項目後，再將這些項目採用區塊分配 (block distribution) 的策略分給所有參與計算的節點。而後各節點便在本機端使用分支與界定演算法結合 depth-first search (DFS) 策略求解，直到沒有任何候選項目為止。

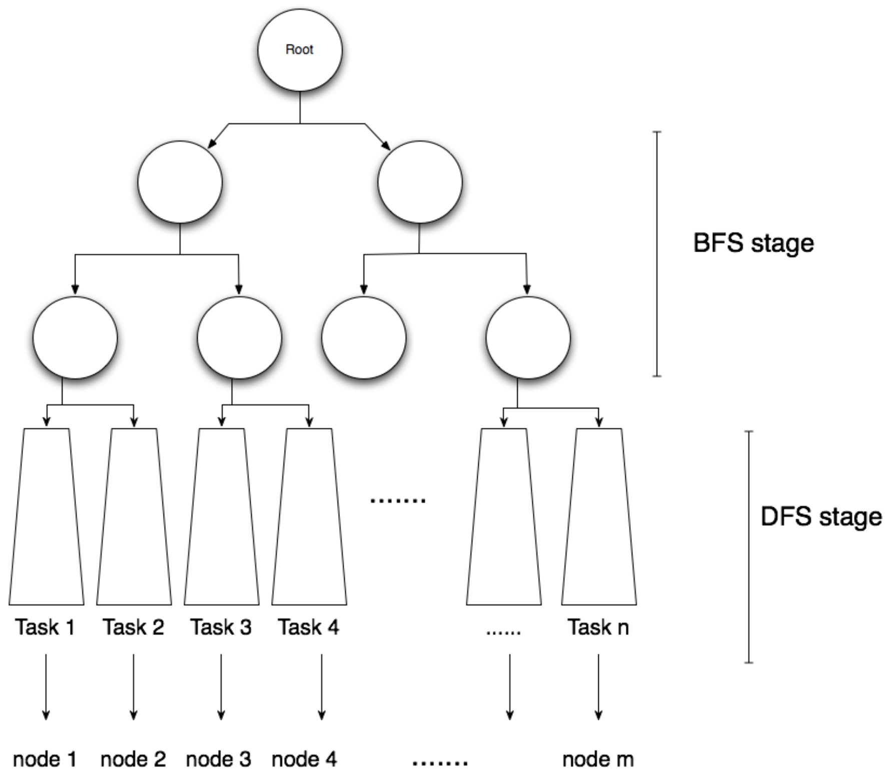


圖 2. Procedure diagram of PB-CIPF

我們提出的分支與界定演算法可以分成三個階段：

1. Branching stage: 採用 BFS 或 DFS 的搜尋策略，並在搜尋到的候選項目中插入新的原子及相對映的鍵結數。
2. Bounding stage: 如果新產生的候選項目的鍵結數目大於該原子本身的限制，則該候選項目就被捨棄。
3. Terminating stage: 重覆上述兩個階段，直到在佇列中沒有任何候選項目為止。

圖 3 為主節點採用 BFS 策略時的一個範例，圖 4 為從屬節點採用 DFS 策略時的範例。

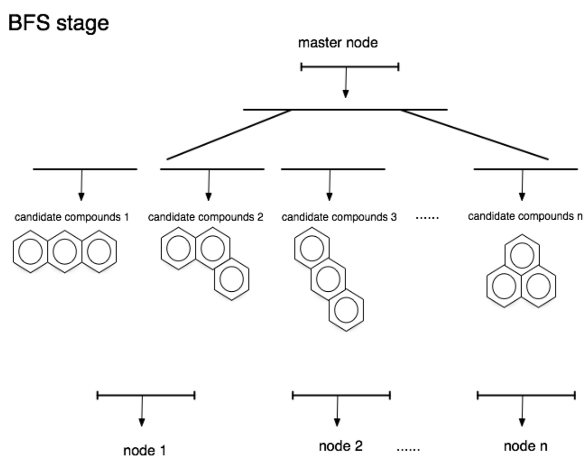


圖 3. 主節點採用 BFS 的範例

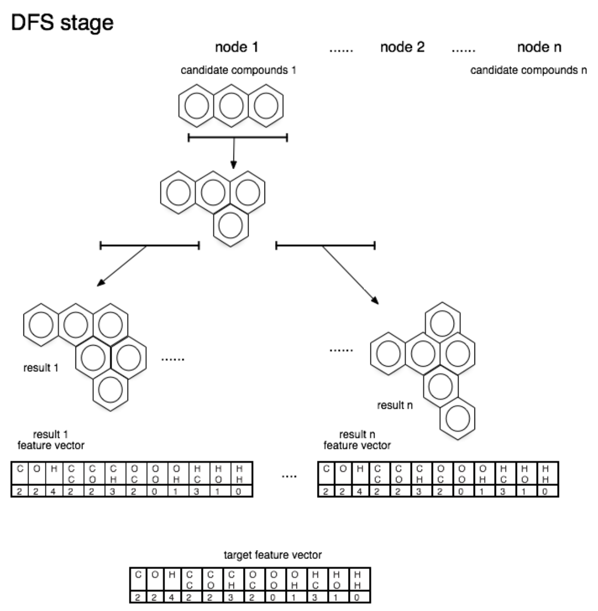
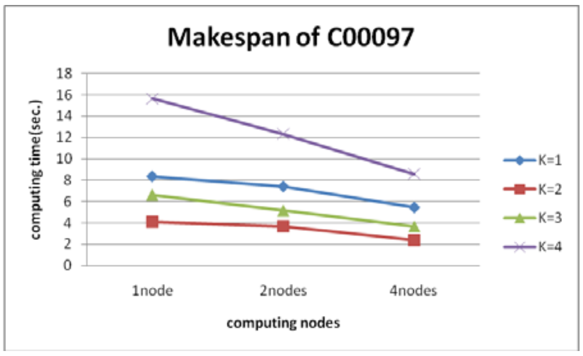
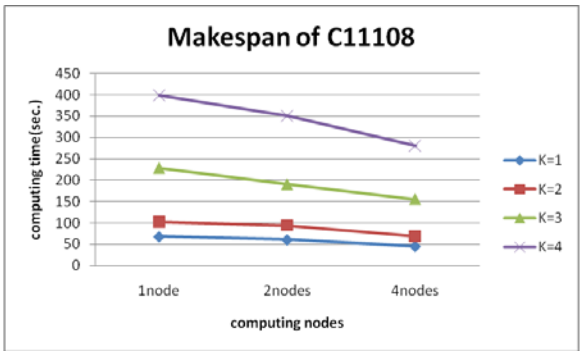


圖 4. 從屬節點採用 DFS 的範例

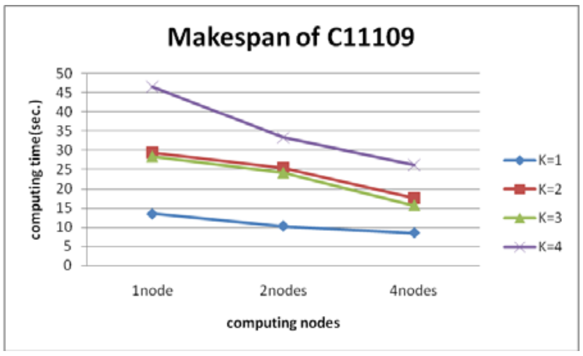
在實驗的結果中，我們使用的叢集系統硬體規格為 AMD Athlon XP 2000+ 的處理器以及 1GB 記憶的電腦，並且採用 C 以及 MPI 做為通訊函式庫作我們的程式。為了驗證程式的正確性及效能，我們使用了 KEGG LIGAND Compound Database 中的化合物為做實驗資料。效能評估以程式執行的 makespan 做為比較的依據，並且對 K=1 到 K=4 不同限制條件分別在 1, 2, 4 個節點上執行。(見圖 5) 結果中可以觀到，我們提出的方法能夠在計算節點增加時，降低計算所需要的時間。



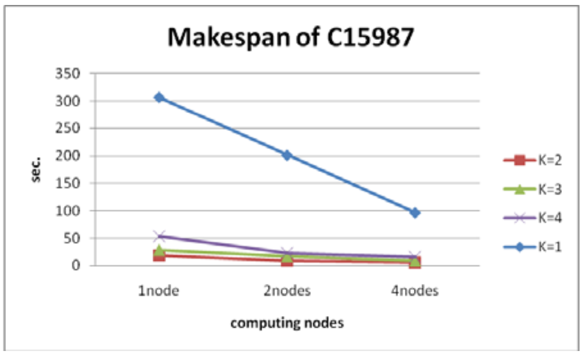
(a)



(b)



(c)



(d)

圖 5. Makespan of C00097, C11108, C11109, C15987

2.2 平行頻繁項目集探勘

探勘頻繁項目集是許多資料探勘研究的基礎，分類（Classification）、分群（Clustering）、關聯式法則（Association rule）、時間序列（Time sequence）。以關聯式法則為例子，主要的目標是關聯式法則是一個能從大量資料中去找出資料間彼此關聯性的技術。例如一家資訊商品量販店從過去 3 個月的交易紀錄中找出顧客最常購買的商品關聯性，發現凡是購買了噴墨印表機的顧客，都會在一個月內回到店內來購買墨水匣，因此店家便可以利用這樣的發現去訂出促銷決策，提升其業績及利潤。

然而隨著資訊系統被廣泛應用，資料庫的交易筆數增加非常快速，面對大量的資料，只要做一個檢索的動作可能就需要非常多的時間，更違論要去找出項目之間的相關性，當探勘時間需要好幾個工作天甚至是一個禮拜之時，探勘得到的資料早已過時。因此，平行運算技術是減少探勘所需時間的主要策略。

我們已設計並實作平行頻繁項目集探勘演算法，並且於叢集系統及網格系統中執行並驗證。首先，設計平行程式時，最重要的是需要資料計算時間的瓶頸的所在，所以我們實作先前 Javed [6] 所提出的 PFP-tree 演算法，並且將各階段計算所需時間列於表 1。表

中可以觀察到，整體的效能瓶頸在於資料交換的步驟 (Exchange)，於是我們提出的方法就是以減少資料交換所需時間為目標。

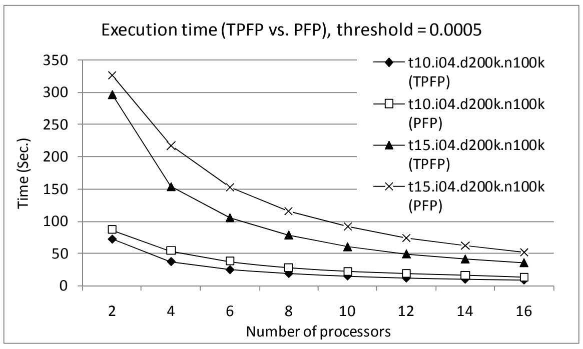
表 1. PFP-tree 各階段計算所需時間(t20.i04.d200k.n100k, threshold = 0.0005)

	P01	*P02*	*P03*	*P04*	*P05*	*P06*	*P07*	*P08*
Data Transfer	0.91	0.04	0.06	0.12	0.06	0.05	0.08	0.04
Header Table	0.6	0.59	0.6	0.59	0.6	0.6	0.6	0.59
All reduce	1.05	0.17	0.18	0.17	0.17	0.17	0.17	0.17
FP-tree	13.62	12.49	12.49	12.40	12.40	12.42	12.43	12.51
Exchange	98.29	157.11	204.78	233.51	241.07	235.06	223.06	197.02
FP-growth	18.06	26.34	27.09	31.1	24.51	22.44	20.07	12.59
Total	132.53	196.74	245.2	277.89	278.81	270.74	256.41	222.92

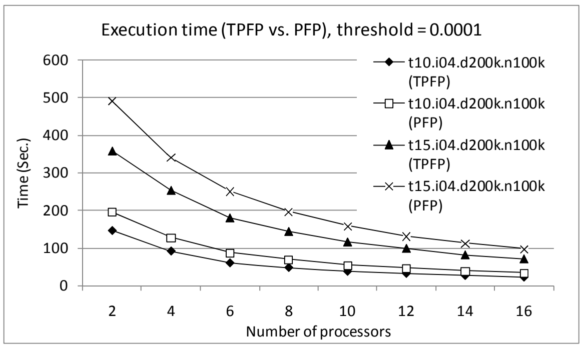
對叢集系統以及格網系統分別提出了 Tidset-based Parallel FP-tree (TPFP-tree) 以及 Balanced Tdiset Parallel FP-tree (BTP-tree)。同時我們也分別對於上述兩個演算法分別在實際的叢集系統及網格系統執行並驗證我們所提出的方法。叢集系統的軟硬體規格如表 2，其實驗結果列於圖 6。圖中可以觀察到，當資料量愈大時，或其支持度愈小時，我們的平行程式愈能夠展現出其擴充性。

表 2. Hardware and Software Specification of PC cluster

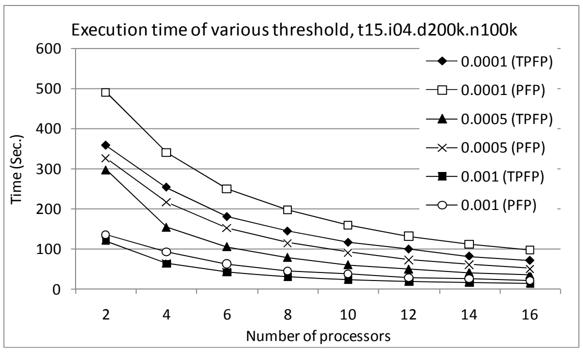
Hardware Specification	
CPU	AMD Athlon XP 2000+
Memory	1GB DDR ram
Network	100 Mbps interconnection network
Disk	80GB IDE H.D.
Software Configuration	
OS and Compiler	Linux 2.6 Gcc/G++ 4.03
Message Passing Library	MPICH2 1.0.5 mpi4py 0.5



(a)



(b)



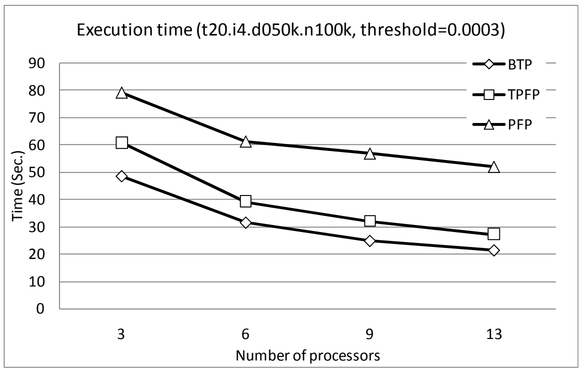
(c)

圖 6. Execution time of TPFP-tree

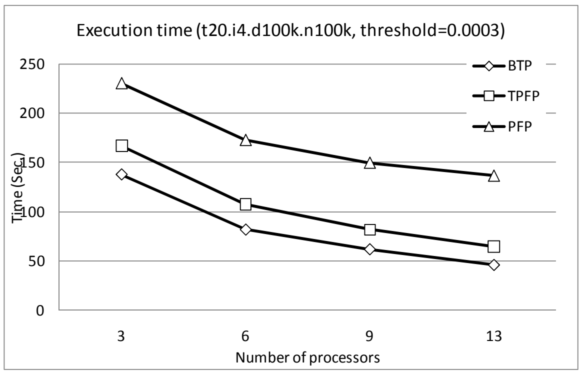
相同的，表 3 為 BTP-tree 執行時的網格系統軟硬體規格，而圖 7 為 BTP-tree 在網格系統上的執行時間。實驗結果中可以觀察到，即是在網格的異質環境下，我們提出的 BTP-tree 依然能夠平衡不同節點間的負載，並且降低計算所需時間。

表 3. Hardware and Software Specification of Multi-cluster Grid

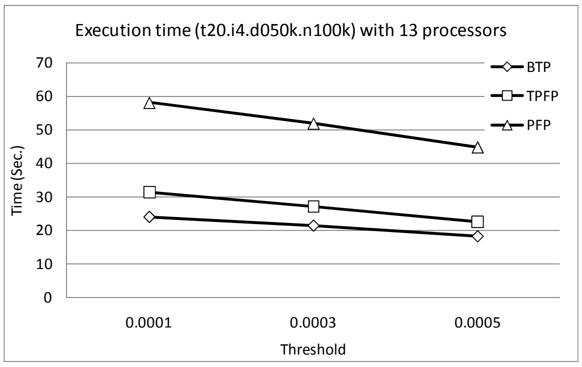
	Cluster 1	Cluster 2	Cluster 3
	Hardware Specification		
Number of Nodes	5	5	3
CPU	Pentium 4 3.2G	AMD XP 2.0G	Pentium 4 3.0G
Memory	512 MB	1024 MB	1024 MB
Network	100 Mbps interconnection network		
	Software Configuration		
OS and Compiler	Linux 2.6 Gcc/G++ 4.03 Globus Toolkit 4.0		
Message Passing Library	MPICH-G2 MPICH2 1.0.5 mpi4py 0.5		



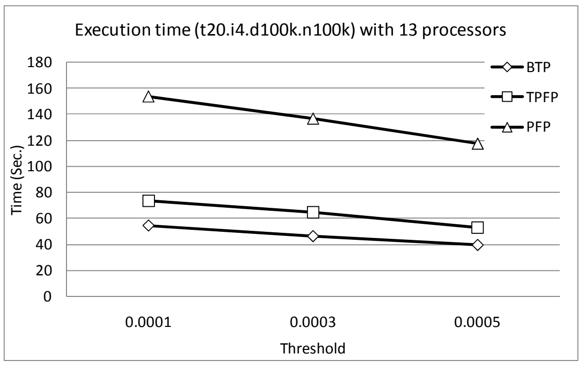
(a)



(b)



(c)



(d)

圖 7. Execution time of BTP-tree

3 計畫成果自評

本計畫之研究成果已達到計畫預期之目標--成功為兩個於生物資訊及資料探勘領域中重要的問題設計平行演算法，分別為化合物推論以及頻繁項目集探勘。對化合物推論問題，我們所設計出的平行演算法可以運用叢集系統的計算資源以降低計算時間，並且亦使用 KEGG 資料庫中的化合物來驗證程式的正確性。化合物推論的部份不但已順利完成預期目標，亦將所獲得的成果整理成論文並且發表了兩篇國際研討會論文 (其中一篇為 EI)，以及一篇國內研討會論文。在頻繁項目集探勘的問題部份，我們已達到預期目標在叢集及網格系統分別設計一平行演算法，從實驗結果中可以觀察到我們提出的方法已經能夠藉由叢集系統的計算能力減少所需時間。並且在網格系統上，我們提出的方法能夠有效的平衡計算節點間的負載，以達到高效能計算的目標。除了完預期目標外，亦將所獲得的成果整理成論文並且發了一篇國際期刊論文 (SCI)，以及一篇國際研討會論文 (EI)。最後感謝國科會給予機會讓本計畫有此研究成果，下一個計畫我們將更加努力，爭取經費以建立更完備的研究環境，同時亦感謝參與計畫的同學的努力讓本計畫得以完成預期目標。

已發表之論文如下：

1. Kun-Ming Yu, Jiayi Zhou, "Parallel Tid-based Frequent Pattern Mining Algorithm on a PC Cluster and Grid Computing System," *Expert System with Applications*, In press, 2009. (SCI) (2008 impact factor: **2.596**; Subject categories: Computer Science, Artificial Intelligence)
2. Kun-Ming Yu, Hui-Yuan Wang, Jiayi Zhou, Chun-Yuan Lin, Chuan Yi Tang, "Parallel Branch and Bound approach with MPI Technology in Inferring Chemical Compounds with Path

Frequency," *Proceedings of the 2009 IEEE International Conference on Granular Computing (GrC)*, pp. 733-738, 2009. (EI) (2009.08.17 - 2009.08.19; Lushan Mountain / Nanchang, China)

3. Kun-Ming Yu, Yi-Yan Chang, Jiayi Zhou, Chun-Yuan Huang, Whei-meih Chang, Chun-Yuan Lin, Chuan Yi Tang, "Chemical Compounds with Path Frequency Using Multi Core Technology," *Proceedings of the 4th International ICST Conference on Scalable Information Systems (INFOSCALE)*, 2009. (2009.06.10 - 2009.06.11, Hong Kong, China)
4. Jiayi Zhou, Kun-Ming Yu, "Balanced Tidset-based Parallel FP-tree Algorithm for the Frequent Pattern Mining on Grid System," *Proceedings of 4th International Conference on Semantics, Knowledge and Grid (SKG 2008)*, pp. 103-108, 2008. (2008.12.03 - 2008.12.05; Beijing, China)
5. 游坤明, 張義諺, 周嘉奕, 林俊淵, 唐傳義, "Inferring a Chemical Compound from Path Frequency Using Multi-Core Technology," *2008 生物資訊科技與應用研討會*, 2008. (2008.11.07; Hsinchu, Taiwan)

4 參考文獻

[1] B. Buchanan, and E. Feigenbaum, DENDRAL and Meta-DENDRAL: Their Applications Dimension. Artificial Intelligence 11 (1978) 5-24.

[2] K. Funatsu, and S. Sasaki, Recent advances in the automated structure elucidation system, chemics. utilization of two-dimensional nmr spectral information and development of peripheral functions for examination of candidates. J. Chem. Inf. Comput. Sci 36 (1996) 190-204.

[3] J. Faulon, C. Churchwell, and D. Visco Jr, The signature molecular descriptor. 2. Enumerating molecules from their extended valence sequences. J. Chem. Inf. Comput. Sci 43 (2003) 721-734.

[4] L. Hall, R. Dailey, and L. Kier, Design of molecules from quantitative structure-activity relationship models. 3. Role of higher order path counts: path 3. Journal of Chemical Information and Computer Sciences 33 (1993) 598-603.

[5] M. Deshpande, M. Kuramochi, N. Wale, and G. Karypis, Frequent substructure-based approaches for classifying chemical compounds. IEEE Transactions on Knowledge and Data Engineering 17 (2005) 1036-1050.

[6] A. Javed, and A. Khokhar, Frequent pattern mining on message passing multiprocessor systems. Distributed and Parallel Databases 16 (2004) 321-334.

行政院國家科學委員會補助國內專家學者出席國際學術會議報告

97 年 9 月 10 日

報告人姓名	游坤明	服務機構及職稱	中華大學 資工系副教授
時間 會議地點	97/8/26 –97/8/28 Hangzhou, China	本會核定 補助文號	
會議 名稱	(中文) 2008 IEEE 粒度計算國際研討會 (英文) 2008 IEEE International Conference on Granular Computing		
發表 論文 題目	(中文) 一個運用權重負載平衡技術進行平行化 Apriori 關連式法則探勘 (英文) A Weighted Load-Balancing Parallel Apriori Algorithm for Association Rule Mining		

一、 參加會議經過

會議的開幕典禮由大會主席致簡單的歡迎詞,並由會議的委員會主席說明本屆的投稿與錄取篇數及本屆會議的特色與會議重點後，隨即展開，本屆會議分別於第一天及第二天的會議議程中各安排了五、六個場次之粒度計算與生物資料處理暨應用領域之最新趨勢之專題報告: (1). Intelligence and Language How Could Human Being Have Language?, (2). What is Granular Computing?: Highlights and Fallacies, (3). Security and Integrity in Outsourcing of Data Mining, (4). Cloud Computing, (5). Cloud Computing Panel (6). Challenges Techniques for Mining Real Clonical Data, (7). Granular Computing:Based on Fuzzy and Tolerance Relation, (8). Granular Computing: Toward inner logic in decision systems, (9). Mechanism Approach to the Research of Atificial Intelligence, (10). Nonstructure Information Retrieval Tolerant Granular Space Model, 以及(11). Clustering Algorithms with Automatic Selection of Cluster Number，分別由 Setsuo Ohsuga、Tsau Young Lin、 David Wai-lok. Cheung、Dennis Quan、 Meng Ye、 Wesley W. Chu、Bo Zhang and Ling Zhang、 Lech Polkowski、Yixin Zhong、ZhongZhi Shi 以及 Michel Ng 作精彩的專題報告。正式之論文報告分別於專題報告後進行，本人之論文『A Weighted Load-Balancing Parallel Apriori Algorithm for Association Rule Mining』被安排在第二天的 –"Intelligent Data Analysis and Applications"之場次發表。

二、 與會心得

GrC 2008 是一個在粒度計算(Granular computing)研究領域中具有指標性的最重要國際會議,此次會議共有近四百篇的論文投稿,但僅錄取了一百七十餘篇優秀的粒度計算資訊系統領域的論文,由會議的進行過程中可以看出主辦單位對會議的流程安排相當用心,參與此次會議之篇學者可藉著此次會議,在粒度計算資訊系統與生物資訊運用之各研究領域互相作深入的討論，而且可藉此機會認識在此領域中之權威研究學者，對於此後從事此領域之研究有相當之助益。

三、　考察參觀活動(無是項活動者省略)

無。

四、　建議

無。

五、攜回資料名稱及內容

1. 大會議程
2. The Proceeding of 2008 IEEE International Conference on Granular Computing 研討會論文集。

六、　其他

A Weighted Load-Balancing Parallel Apriori Algorithm for Association Rule Mining

Kun-Ming Yu[1], Jia-Ling Zhou[2]
[1]*Department of Computer Science and Information Engineering, Chung Hua University*
[2]*Department of Information Management, Chung Hua University*
[1]*yu@ chu.edu.tw,* [2]*jlzhou@pdlab.csie.chu.edu.tw*

Abstract

Because of the exponential growth in worldwide information, companies have to deal with an ever growing amount of digital information. One of the most important challenges for data mining is quickly and correctly finding the relationship between data. The Apriori algorithm is the most popular technique in association rules mining; however, when applying this method, a database has to be scanned many times and many candidate itemsets are generated. Parallel computing is an effective strategy for accelerating the mining process. In this paper, the Weighted Distributed Parallel Apriori algorithm (WDPA) is presented as a solution to this problem. In the proposed method, metadata are stored in TID forms, thus only a single scan to the database is needed. The TID counts are also taken into consideration, and therefore better load-balancing as well as reducing idle time for processors can be achieved. According to the experimental results, WDPA outperforms other algorithms while having lower minimum support.

1. Introduction

With the rapid development of information technology, companies have been working on digitizing all areas of business to improve efficiency and thus competitiveness. However, the consequences of full-digitization are that tremendous amounts of data are generated. It is important to extract meaningful information from scattered data, and data mining techniques are developed for that purpose. There are many techniques being used for data mining, for example, Classification, Regression, Time Series, Clustering, Association Rules and Sequence. Association rule [1, 2] is one of the most useful techniques in data mining. Generally, it takes long to find the association rules between datasets when a database contains a large number of transactions. By applying parallel-distributed data mining techniques, the mining process can be effectively speeded up. With parallel-distributed data mining the calculation is done in a distributed environment [3, 7, 8, 9, 12], but most of the time, irregular and imbalanced computation loads are allocated between processors and thus the overall performance is degraded.

In this paper the Weighted Distributed Parallel Apriori algorithm (WDPA) is presented as a solution for this problem. In the proposed method, a database has only to be scanned once because metadata are stored in TID tables. This approach also takes the TID count into consideration. Therefore, WDPA improves load-balancing as well as reduces idle time of processors.

The experimental results in this study showed that the running time of WDPA was significantly faster than that of previous methods. In some cases, WDPA only used about 2% of the time used in previous methods. This can be achieved because WDPA successfully reduced the number of scan iterations to databases and was able to evenly distribute workloads among processors.

The paper is organized as follows: In section 2, association rule and parallel distributed algorithms are explained. The WDPA algorithm is proposed in section 3. Section 4 gives the experimental results. Finally, the conclusion is given in section 5.

2. Related Work

Frequent pattern mining problem is defined as follows. Let DB = $\{T_1, T_2, \ldots, T_k\}$ be a database of transactions, where each transaction T_e consists of I, I = $\{i_1, i_2, \ldots, i_m\}$ be a set of all items. Assuming A, B are itemsets, A, B $\subseteq$ I, A $\cap$ B=$\emptyset$, A$\rightarrow$B denotes there is an association rule between A and B. Each association rule has *support* and *confidence* to confirm the validity of the rule. *Support* denotes the occurrence rate of an itemset in a DB. *Confidence* denotes the proportion of data items containing Y in all items containing X in a DB. When the *support* and *confidence* are greater than or equal to the pre-defined *minimum support* and *minimum confidence*, the association rule is considered to be a valid rule.

The Apriori algorithm was proposed by R. Agrawal and R. Srikant in 1994 A.D. [2]. The Apriori algorithm is one of the most representative algorithms in mining association rules. It is based on the assumption that subsets of low-frequency itemsets must be low-frequency as well. Even though the Apriori algorithm takes lots of time to calculate combination of itemsets, the design of the data structure makes it easy for the algorithm to be parallelized. Therefore, some scholars propose that parallel- distributed Apriori algorithms be used [3, 7, 8, 9, 10, 11, 12, 13, 14]. For example, CD, DD, FDM, FPM, DMA etc. Recently, Ye [12] proposed a parallel-distributed algorithm using Trie Structure [6]. Ye's algorithm distributes computing workload using the Trie Structure to speed up the computation, however, this causes significant variance between the sizes of candidate itemsets distributed among processors. Moreover, this method also requires a database to be scanned many times. The problems related to multiple-scan and load-imbalance gets worse when dealing with large databases and huge itemsets. Therefore, a Weighted Distributed Parallel Apriori algorithm (WDPA) is proposed. By storing the TIDs of itemsets and precisely calculating and distributing computation workloads, WDPA is able to effectively accelerate the computation of itemsets and reduce the required scan iterations to a database and balancing the load, thus significantly reduces processor idle time.

3. Weighted Distributed Parallel Apriori (WDPA) Algorithm

To avoid the problems associated with load-imbalance and multiple-scan, the WDPA algorithm is proposed so that a database only needs to be scanned once while maintaining load balancing among processors. In the algorithm, each transaction has a *Transaction IDentification*, called TID. By using hash functions to store TID in table structure, the number of itemsets can be quickly calculated without the need of rescanning the database.

In the WDPA parallel-distributed processing algorithm, the number of combinations for items, called a lattice, is first calculated. Lattice is the number of combinations calculated from candidate (k+1)-itemset counts by frequent k-itemsets. Equation (1) is the required count of itemset combinations of I_i, Equation (2) represents the total number of k-itemsets. Using block division to do frequent itemset distribution after calculating the number of combinations, is called Block_Lattice (BL) (Figure 1). From Figures 1 and 2 we can see that the Lattice count decreases gradually at the lower-left part of the matrix, thus if itemsets are distributed sequentially, the load-imbalance distribution will occur. Therefore by cyclic partitioning of lattice, itemsets are distributed to the processors cyclically to balance the distribution of itemsets. This is called $Cyclic\_Lattice(CL)$.

$$Cnt\_Lattice(I_i) = [(len(freq_{k-1}) - 1) - i] \tag{1}$$

$$TotalCnt\_Lattice = \sum_{i=0}^{len(freq_{k-1})-1} Cnt\_Lattice(I_i) \tag{2}$$

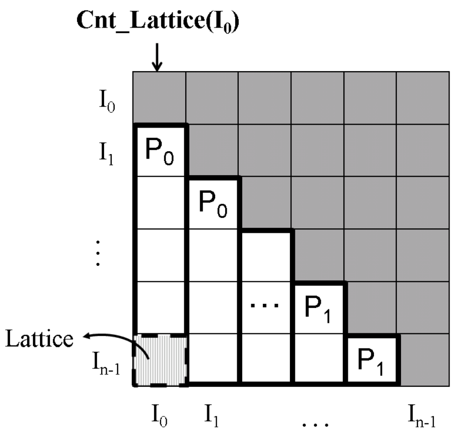


Figure 1. Block partitioning

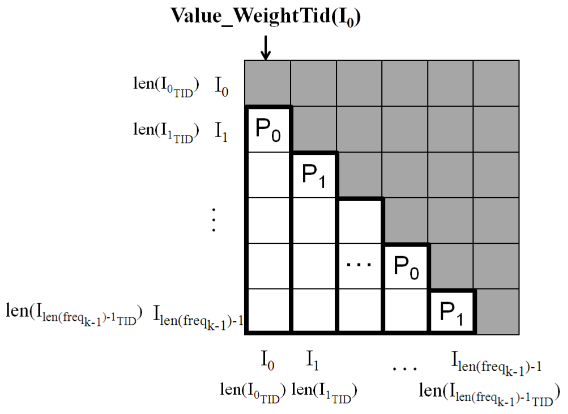


Figure 2. Cyclic partitioning

By only calculating the Lattice number and ignoring the length of the itemsets' TID, an uneven distribution of workload occurs. Therefore, this algorithm also takes TID length into consideration and regards it as a weight value, which makes the distribution of itemsets more accurate and more even. Equation (3) calculates the weighted value of itemset

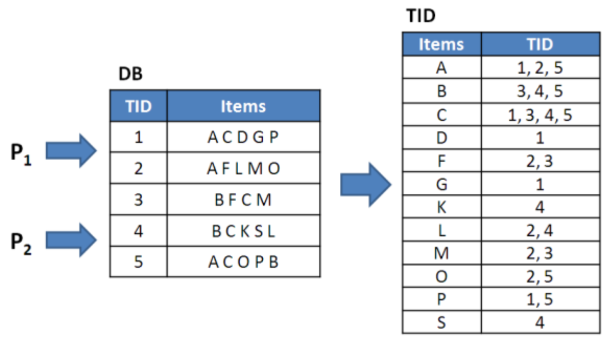
Ii. Equation (4) represents the total weight value of k-itemsets.

$$Value\_WeightTid(I_i) = \sum_{j=i+1}^{len(freq_{k-1})-1} len(I_{i_{TID}}) \times len(I_{j_{TID}}) \quad (3)$$

$$TotalValue\_WeightTid = \sum_{i=0}^{len(freq_{k=1})-1} \sum_{j=i+1}^{len(freq_{k-1})-1} len(I_{i_{TID}}) \times len(I_{j_{TID}}) \quad (4)$$

There are two methods of partitioning weighted TID, the *Block_WeightTid(BWT)* partitions and distributes TID by block, the *Cyclic_WeightTid(CWT)* partitions and distributes TID in cyclic.

An example of the algorithms is given below:
For step1 and step2, P_1 (MP), P_2 (SP) read and scan database, then build level-1 candidate itemsets. (Figure 3)

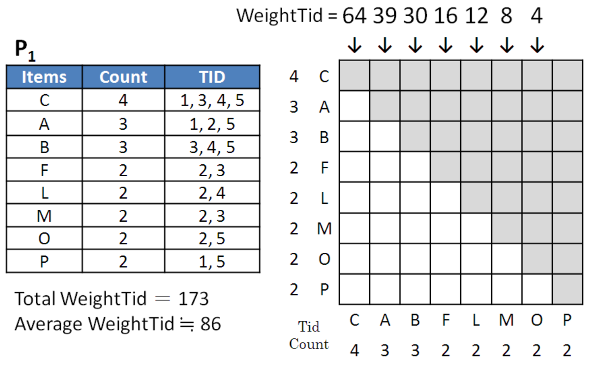


TID	Items
1	A C D G P
2	A F L M O
3	B F C M
4	B C K S L
5	A C O P B

Items	TID
A	1, 2, 5
B	3, 4, 5
C	1, 3, 4, 5
D	1
F	2, 3
G	1
K	4
L	2, 4
M	2, 3
O	2, 5
P	1, 5
S	4

Figure 3. Scan database and creating TID forms

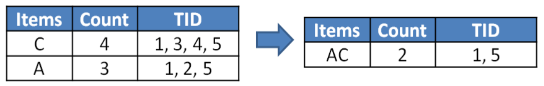
Figure 4 shows that P_1 collects itemsets that match given support into frequent 1-itemsets, then uses CWT to calculate and distribute the itemsets on P_1. P_1: {C, B}, P_2: {A, F, L, M, O}. (Step 3 and Step 4)



Items	Count	TID
C	4	1, 3, 4, 5
A	3	1, 2, 5
B	3	3, 4, 5
F	2	2, 3
L	2	2, 4
M	2	2, 3
O	2	2, 5
P	2	1, 5

Figure 4. Distributing frequent 1-itemsets on P_1

Figure 5 describes P_1 and P_2 combining level-2 candidate itemsets and calculating itemset counts according to the TID table. Figure 6 represents level-2 candidate itemset counts on P_1 and P_2. (Take level-2 candidate A and C for example, the intersection of A and C on TID, [1, 5], is the resulting set, AC) (Step 5 and Step 6)



Items	Count	TID
C	4	1, 3, 4, 5
A	3	1, 2, 5

Items	Count	TID
AC	2	1, 5

Figure 5. Itemsets are calculated by counting the TID forms

P_1			P_2		
Itemsets	Count	TID	Itemsets	Count	TID
AC	2	1, 5	AB	1	5
BC	3	3, 4, 5	AF	1	2
CF	1	3	AL	1	2
CL	1	3	AM	1	2
CM	1	3	AO	2	2, 5
CO	1	5	AP	2	1, 5
CP	2	1, 5	FL	1	2
BF	1	3	FM	2	2, 3
BL	1	4	FO	1	2
BM	1	3	FP	0	(null)
BO	1	5	LM	1	2
BP	1	5	LO	1	2
			LP	0	(null)
			MO	1	2
			MP	0	(null)

Figure 6. P_1 and P_2 level-2 candidate itemsets

Select the itemsets that match the given support value, and save them as frequent 2-itemsets. Because the frequent 1-itemsets are larger, candidate itemsets that required combination computation will be larger, too. In this case, distributing the itemsets in Cyclic will produce better results. On the other hand, if there are frequent itemsets above level 1, candidate itemsets that required combination computation will be smaller, too. In this case, distributing the itemsets in Block will produce better results.

P_1 receives P_2 itemsets, and repeats execution step 4 to step 9 until there are no more frequent itemsets. Figure 7 illustrates the use of BWT to calculate and distribute the itemsets on P_1. P_1: {BC, AC}, P_2: {AO, AP, CP}. (Step 3 and step 4)

Figure 8 represents P_1 combined level-3 candidate itemsets matching given support value into frequent 3-itemset.

P$_1$

Itemsets	Count	TID
BC	3	3, 4, 5
AC	2	1, 5
AO	2	2, 5
AP	2	1, 5
CP	2	1, 5
FM	2	2, 3

Total WeightTid = 70
Average WeightTid = 35

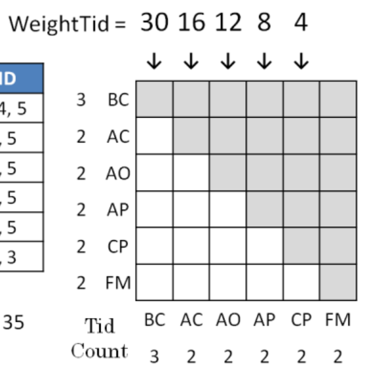


Figure 7. Distributing frequent 2-itemsets on P$_1$

P$_1$

Itemsets	Count	TID
ACP	2	1, 5

Figure 8. The resulting frequent 3-itemsets

Because the resulting frequent 3-itemset contains only one itemset, no more combination operation can be made, the mining process ends here. The number of resulting frequent itemsets are: level-1: Eight, level-2: Six, level-3: One.

The algorithms are described in detail below:

Input: a transaction database DB = $\{T_1, T_2, ..., T_n\}$, and each transaction $T_i \subseteq I$, $I = \{i_1, i_2, ..., i_m\}$. A given minimum support s. P is the number of processors. (p_1 is master processor (MP), and $p_2, p_3, ..., p_p$ are salve processors (SPs))
Output: All frequent itemsets.
Method:
Step 1. Each processor reads the database DB.
Step 2.Each processor scans DB and creates the transaction identification set (TID).
Step 3. Each processor calculates candidate k-itemset counts, when the count is greater than s, let freqk be frequent k-itemsets.
Step 4. MN equally divides the $freq_k$ into p disjointed partitions and assigns $itemsets_i$ to p_i. $Itemsets_i$ denote that SPs were assigned to the itemsets from MN. The frequent pattern dividing method:
(1) Block_Lattice (BL)
(2) Cyclic_Lattice (CL)
(3) Block_WeightTid (BWT)
(4) Cyclic_WeightTid (CWT)
Step 5. Each processor receives the $itemsets_i$ and the combination candidate (k+1)-itemsets.
Step 6. Each processor candidate itemsets is calculated by counting the TID forms.
Step 7. When itemset count is greater than s then it is a frequent (k+1)-itemset, and itemset appeared in transaction id is saved to (k+1)-TID.
Step 8. SPs send frequent itemsets to MN.
Step_9.MN receives SPs itemsets, and repeats execution step4 to setp9 until there are no more frequent itemsets.

4. Experiments

In order to evaluate the performance of the proposed algorithm, the WDPA was implemented along with the algorithm proposed by Ye [12]. The program was executed in a PC cluster with 16 computing processors. Table 1 gives the hardware and software specifications. Synthesized datasets generated by IBM's Quest Synthetic Data Generator [4] were used to verify the algorithm. Moreover, the database T10I4D50N100K, T10I4D100N100K, T10I4D200N100K was used to examine the WDPA. From the experimental results, our proposed method balances the workload among processors and saves on processor idle time because of the way CWT distributes itemsets. Therefore, the following experiments are calculated based on the CWT method.

Table 1. Hardware and Software Specifications

Hardware Environment	
CPU	AMD Athlon Processor 2200+
Memory	1GB DDR Ram
Network	100 Mbps interconnection network
Disk	80GB IDE H.D.
Software Environment	
O.S.	ReadHat Linux 7.3
Library	MPICH2 1.0.3

Figure 9 shows the speed up of four WDPA methods. From the results, it can be seen that the speeded up of the four methods is similar, but the CWT itemsets distributed used weighted TID and cyclic partition, therefore the CWT have more accurately parallel-distributed itemsets. According to the experiment, the CWT, regardless of processor numbers of 1,2,4,8,16, achieves better results than the other methods.

Figure 10 shows the execution time of the WDPA and Ye’s algorithm on different processors. WDPA(8) denotes that the WDPA algorithm used eight processors. Because Ye’s algorithm needs to repeatedly scan the database, the loads are imbalanced between processors. Therefore, from Figure 10, WDPA is nearly 120 times faster than Ye’s algoritm in the 16 processors case. The use of TID form in WDPA accurately parallelize the workloads, hence it effectively reduced the database scanning and saved on processor idle time.

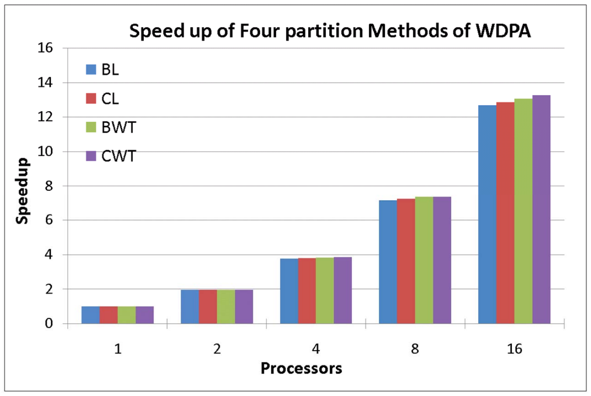


Figure 9. Speedup of Four partition Methods of WDPA (T10I4D100KN100K, minsup: 0.2%)

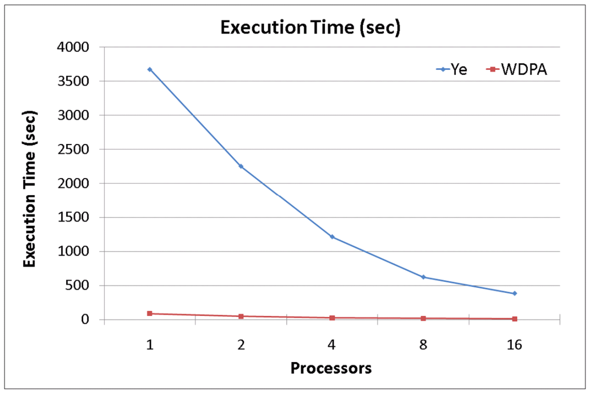


Figure 11. Execution Time (WDPA vs. Ye's algorithm)(T10I4D100KN100K, minsup: 0.2%)

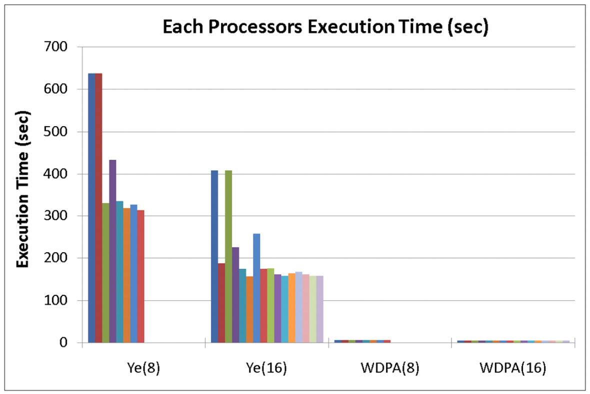


Figure 10. Each Processors Execution Time (T10I4D50KN100K, minsup: 0.2%)

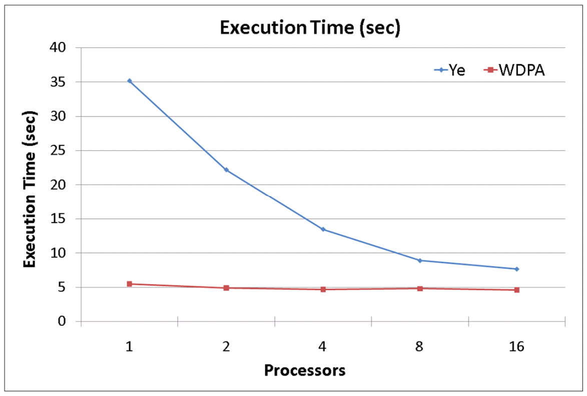


Figure 12. Execution Time (WDPA vs. Ye's algorithm)(T10I4D100KN100K, minsup: 0.3%)

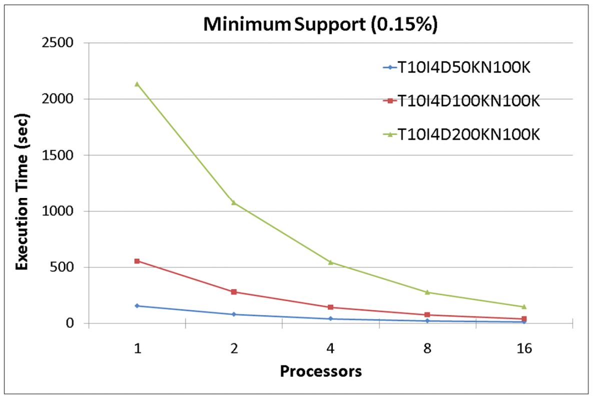


Figure 13. WDPA Execution Time, minsup: 0.15%

Figure 11 and 12 show the execution time and speedup under different given supports. Ye's algorithm requires that a database being re-scanned for every itemset to be counted during the mining process, so when there is lower support, Ye's algorithm takes longer to re-scan the database. On the other hand, using the TID table with precise distribution of itemsets, the WDPA scans the database once only. This greatly reduced the time spent on database scanning and balanced the computation workload among processors. Thus, there is an obvious performance advantage of the WDPA algorithm over Ye's algorithm.

Figures 13 and 14 give the execution time and speed up with different databases. With the increased size of the database, the length of the TID of itemsets in the table will increase. Therefore, when the size of the database increased, the execution took longer. Moreover, by parallel-distributing the processing, large databases can be more effectively mined and itemsets will be allocated to different processors to perform the calculation, this significantly speeding up the mining process.

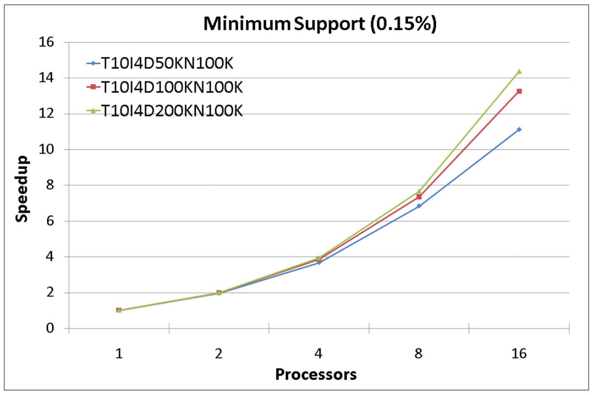


Figure 14. WDPA Speedup, minsup: 0.15%

Figure 15 illustrates the execution time of various minimum supports. The number of frequent itemsets as well as their length increased with a lower support in WDPA. Therefore, by using this method of calculation WDPA effectively accelerated the computation. Thus WDPA can achieve better speedup when there was lower minimum support.

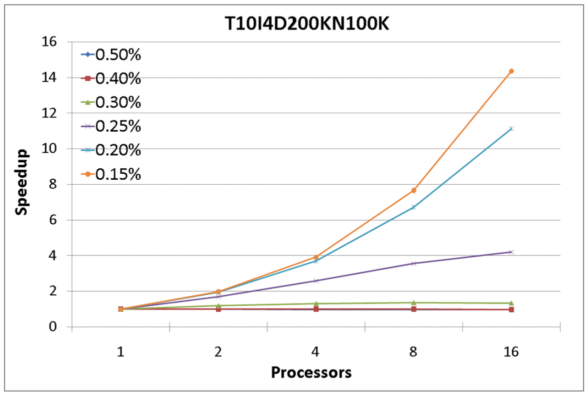


Figure 15. WDPA Speedup

5. Conclusion

Determining the association between items in a huge database, is a worthwhile research topic. However, the process of generating itemsets and confirming them is time consuming. Parallel-distributed computation strategies provide workable solutions to this problem. In this paper, a Weighted Distributed Parallel Apriori algorithm (WDPA) is proposed, in which the TID of itemsets is stored in a table to compute their occurrence. WDPA effectively reduced the required scan iterations to a database as well as accelerated the calculation of itemsets. By taking the factor of itemset counts into consideration, this approach effectively balanced workloads among processors and reduced processor idle time. Experimental results show that WDPA achieved higher speedups than pervious works in the case of high data volume and low support.

References

[1] Agawal, R., Imilinski, T., and Swami, A. “Mining Association Rules between Sets of Items in Large Databases,” *Procceeding of the 1993 ACM SIGMOD International Conference on Management of Data*, Vol. 22, Issue 2, June 1993, pp. 207-216.
[2] Agrawal, R. and Srikant, R. “Fast Algorithms for Mining Association Rules,” *Proceeding of 20th International conference on Very Large Database*, 1994, pp. 487-499.
[3] Agrawal, R. and Shafer, J.C. “Parallel Mining of Association Rules,” *IEEE Transaction On Knowledge And Data Engineering*, Vol. 8, Issue 6, December 1996, pp. 962-969.
[4] Almaden, “I. Quest synthetic data generation code,” http://www.almaden.ibm.com/cs/quest/syndata.html.
[5] Apte, C. and Weiss, S.M. “Data Mining with Decision Trees and Decision Rules,” *Future Generation Computer Systems*, Vol. 13, Issue 2-3, November 1997, pp. 197-210.
[6] Bodon, F. “A fast Apriori implementation,” *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations*, 2003.
[7] Cheung, D.W., Han, J., Ng, V.T., Fu, A.W., and Fu, Y. “A fast distributed algorithm for mining association rules,” *Fourth International Conference on Parallel and Distributed Information Systems*, December 1996, pp. 31-42.
[8] Cheung, D.W., Lee, S.D., and Xiao, Y. “Effect of Data Skewness and Workload Balance in Parallel Data Mining,” *IEEE Transactions On Knowledge And Data Engineering*, Vol. 14, Issue 3, 2002, pp. 498-514.
[9] Cheung, D.W., Ng, V.T., and Fu, A.W. “Efficient Mining of Association Rules in Distributed Databases,” *IEEE Transaction On Knowledge And Data Engineering*, Vol. 8, Issue 6, December 1996, pp. 911-922.
[10] Einakian, S. and Ghanbari, M. “Parallel Implementation of Association Rule in Data Mining,” *Proceedings of the 38th Southeastern Symposium on System Theory*, 2006, pp. 21-26.
[11] Parthasarathy, S., Zaki, M.J., Ogihara, M., and Li, W. “Parallel Data Mining for Association Rules on Shared-Memory Systems,” *Knowledge and Information Systems*, Vol. 3, Issue 1, 2001, pp.1-29.
[12] Ye, Y. and Chiang, C.C. “A Parallel Apriori Algorithm for Frequent Itemsets Mining,” *Proceedings of the Fourth International Conference on Software Engineering Research, Management and Applications*, 2006, pp. 87-94.
[13] Zaki, M.J., Ogihara, M., Parthasarathy, S., and Li, W. “Parallel Data Mining for Association Rules on Shared-memory Multi-processors,” *Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM)*, 1996.
[14] Zaki, M.J., Parthasarathy, S., Ogihara, M., and Li, W. “Parallel Algorithms for Discovery of Association Rules,” *Data Mining and Knowledge Discovery*, Vol. 1, Issue 4, 1997, pp. 343-373.

Expert Systems with Applications xxx (2009) xxx–xxx

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Parallel TID-based frequent pattern mining algorithm on a PC Cluster and grid computing system

Kun-Ming Yu [a], Jiayi Zhou [b,*]

[a] Department of Computer Science and Information Engineering, Chung Hua University, 707, Section 2, WuFu Road, HsinChu 300, Taiwan, ROC
[b] Institute of Engineering and Science, Chung Hua University, 707, Section 2, WuFu Road, HsinChu 300, Taiwan, ROC

ARTICLE INFO

Keywords:
Frequent pattern mining
Association rules
Data mining
Grid computing
Cluster computing

ABSTRACT

The mining of frequent patterns from transaction-oriented databases is an important subject. Frequent patterns are fundamental in generating association rules, time series, etc. Most frequent pattern mining algorithms can be classified into two categories: generate-and-test approach (Apriori-like) and pattern growth approach (FP-tree). In recent years, many techniques have been proposed for frequent pattern mining based on the FP-tree approach since it only needs two database scans. However, for pattern growth methods, the execution time increases rapidly when the database size increases or when the given support is small. Therefore, parallel-distributed computing is a good strategy for solving this problem. Some parallel algorithms have been proposed, but the execution time is still costly when the database size is large. In this paper, two parallel mining algorithms are proposed; Tidset-based Parallel FP-tree (*TPFP-tree*) and Balanced Tidset-based Parallel FP-tree (*BTP-tree*) for frequent pattern mining on PC Clusters and multi-cluster grids. In order to exchange transactions efficiently, a transaction identification set (*Tidset*) was used to directly select transactions instead of scanning the database. Since a Grid system is a heterogeneous computing environment, the proposed *BTP-tree* can balance the loading according to the computing ability of the processors. *BTP-tree, TPFP-tree and PFP-tree* were implemented, and datasets generated with an IBM Quest Synthetic Data Generator were used to verify the performance of *TPFP-tree* and *BTP-tree*. The experimental results showed that the *TPFP-tree* needed less execution time on a PC Cluster than the *PFP-tree* when the database increased. Moreover, the *BTP-tree* shortened the execution time significantly and had a better load balance capability than both the *TPFP-tree* and *PFP-tree* on a multi-cluster grid.

1. Introduction

Extracting frequent patterns in a transaction-oriented database is vital in the mining of association rules (Agrawal & Srikant, 1994; Park, Chen, & Yu, 1995), time series, classification (Gorodetsky, Karasaeyv, & Samoilov, 2003), etc. The basic problem in frequent pattern mining is finding the number of times for a given pattern appears in a database. Most of the research in this area has either used the generate-and-test (*Apriori*-like) or the pattern growth approach (*FP-growth*) (Coenen, Leng, & Ahmed, 2004; Han, Pei, Yin, & Mao, 2004).

For the *Apriori*-like approach (Lazcorreta, Botella, & Fernández-Caballero, 2008; Park et al., 1995), the core idea is that if any length of the k pattern is not frequent in the database, then the super-pattern (length $k + 1$) cannot be frequent. However, this approach generates a large number of candidate datasets and repetitively scans the database to verify whether it is frequent or not. For example, 2^{50} (about 10^{15}) candidate datasets may be needed to verify whether a set is frequent or not in a database with 50 items.

Han et al. (2004) propose a novel data structure and method for mining frequent patterns: the Frequent Pattern (*FP*) tree data structure which only stores compressed, necessary information for mining. Moreover, a mining algorithm – *FP growth* – based on *FP-tree* was also developed. Unlike the *Apriori* algorithm, the *FP-tree* only scans a database twice and the mining information is obtained from the proposed data structure.

Based on the above, many methods derived from *FP-tree* have been proposed (Hong, Lin, & Wu, 2008; Zhou & Yu, 2008). Moreover, these also proved that FP-tree-like algorithms performed better than *Apriori*-like algorithm. However, even though FP-tree performed better, the execution time still increased significantly when the database was large. A parallel and distribution technique is a good strategy for overcoming this problem. Many parallel-distributed methods have been proposed (Chen, Huang, Chen, & Wu, 2005; Holt & Chung, 2004; Li, Zhu, & Ogihara, 2003; Lin, Lee, Chen, & Yu, 2002; Pramudiono & Kitsuregawa, 2003; Tang & Turkia,

* Corresponding author. Tel.: +886 3 5186360; fax: +886 3 5186416.
E-mail addresses: yu@chu.edu.tw (K.-M. Yu), jyzhou@pdlab.csie.chu.edu.tw (J. Zhou).

0957-4174/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2009.07.072

2006). Javed and Khokhar (2004) propose a parallel *FP-tree* mining algorithm (*PFP-tree*) to solve the problem. The results show that parallel computing is a good approach for solving this problem. However, for *PFP-tree*, when the given threshold is small, or the average length of the transaction is long, too much information should be exchanged among processors. The performance deteriorates notably when the database increases or the given support decreases.

In this study, a balanced parallel frequent-pattern mining algorithm was developed to solve frequent pattern mining problems on a PC Cluster. A PC Cluster is a typical parallel computing environment with homogeneous hardware and software resources. It consists of many computers interconnected with a fast network connection. In addition, a transaction identification set (*Tidset*) was used to directly select transactions instead of scanning whole databases. The goal of the proposed Tidset-based Parallel FP-tree (*TPFP-tree*) algorithm was to reduce both communication and tree insertion cost, thus decreasing execution time.

However, we need more computing resources when the problem sizes increase. Grid computing and pervasive computing can solve complex problems with large-scale computation power and data storage resources (Foster & Kesselman, 1998). Grid is a loosely coupled computing architecture based on internet connection, it shares heterogeneous computing and storage resources related to traditional cluster systems, and it can easily add additional computing resources at lower cost. In order to mine the frequent patterns for a large database, large computation resources are needed. However, the grid computing consisted of heterogeneous resources. Therefore, the mining algorithm should enhance balance in a heterogeneous computing environment. Hence, a Balanced Tidset Parallel FP-tree (*BTP-tree*) algorithm for mining frequent patterns on a grid computing system is proposed. The *BTP-tree* dispatches mining items according to the computation ability of the participating nodes for load balancing. Moreover, the *BTP-tree* can also reduce both the communication and tree insertion cost.

The experimental results show that the proposed algorithms – the *TPFP-tree* and *BTP-tree* could shorten the execution time for various datasets on a PC-Cluster and a multi-cluster grid, respectively. Both algorithms as well as a *PFP-tree* were implemented on a PC Cluster and a multi-cluster grid system with *MPICH* library. The results showed that on the grid system the *BTP-tree* performed better with different size databases, different given thresholds, and different computing nodes. Moreover, the *BTP-tree* also had a better load balancing ability in heterogeneous computing environments.

This paper is organized as follows: In Section 2, the *FP-tree*, *FP-growth*, *PFP-tree*, and grid computing systems are described. The *TPFP-tree* and *BTP-tree* are introduced in Section 3 and Section 4 illustrates our experimental results. Finally, the conclusion and future work are discussed in Section 5.

2. Related work

The frequent pattern mining problem can be defined as follows: $DB = \{T_1, T_2, \ldots, T_n\}$ is a set of transactions. Each transaction $T_i \subseteq I$ where $I = \{i_1, i_2, \ldots, i_m\}$ is a set of all items in a database. $Sup_{DB}(x)$ means the number of transactions in a database that contains pattern x, $Sup_{DB}(x) = |\{t | t \in DB \text{ and } x \subseteq t\}|$. The problem of frequent pattern mining is to find itemset x where $Sup_{DB}(x) \geqslant \xi$ for a given threshold $(1 \leqslant \xi \leqslant |DB|)$.

2.1. Frequent pattern growth (FP-growth)

Han et al. proposed the *FP-growth* (Han et al., 2004) algorithm with which the database only needs to be scanned twice. The FP-growth algorithm can be decomposed into two phases: the *FP-tree* construction and the mining of frequent patterns from *it*.

FP-tree is a data structure representing the necessary information for mining. It consists of a root node labeled as null and child nodes consisting of the item-name, support and node link. Moreover, the database only needs to be scanned twice. The first scan is to create a frequent 1-itemset sorted in descending order in the header table. Secondly, it extracts frequent items from T_i $(i = [1, \ldots, n])$. After sorting the items the frequent items are inserted in the tree. For tree insertion, increase the support of the node if the node corresponding to the item's name is found; otherwise, create a new node and set the support to 1. The header table keeps the node-link which connects nodes with the same item name in the *FP-tree* during the mining process.

The *FP-growth* is then used for mining frequent patterns. It selects an item as mining target from the header table. The prefix path can be found via the node-link, following the node to the root to get the (conditional) pattern base of the item. Then a new *FP-tree,* the conditional *FP-tree*, is constructed based on the pattern. This mining process is repeated until an empty tree with a single path is found. Hence, the frequent patterns based on selected items are found. Then one mining target after another is selected from the header table to find all frequent item sets.

Since the *FP-tree* reduces the number of database scans and uses less memory to represent the necessary information, many frequent pattern mining algorithms are based on its data structure (Lin, Hong, & Lu, 2009; Yan, Zhang, & Zhang, 2009).

2.2. Parallel FP-tree algorithm

To mine frequent patterns from a transactional database requires intensive computation. The execution time increases significantly when the database size is large or the given support is small. Since the database can be divided into different sets of transactions, parallel and distribution technique is a good strategy to solve frequent pattern mining problem. The multi-processor computing environment can consist of homogeneous or heterogeneous computing resources. A PC Cluster is a typical homogeneous computing environment. It has the same hardware and software specifications and is connected by fast networking, e.g., Gigabit Ethernet, InfiniBand, etc. Message Passing Library (MPI) is used to send and receive messages between processors. Since a PC Cluster provides high performance computing, it has been used to shorten computation time for mining frequent patterns (Javed & Khokhar, 2004; Pramudiono & Kitsuregawa, 2003; Tang & Turkia, 2006).

Pramudiono and Kitsuregawa (2003) proposed a parallel *FP-tree* algorithm which exchanges the *conditional pattern base* to parallel the *FP-tree*. Moreover, they also introduced a *path depth* notation to break down the granularity of parallel processing in conditional pattern bases. However, the experimental results show that the ideal speedup ratio is not achieved since it is unbalanced.

Javed et al. proposed a *PFP-tree* algorithm (Javed & Khokhar, 2004). The *PFP-tree* is developed for a *SIMD* computing environment. It is based on the *FP-tree* data structure and divides *DB* into different partitions DB_i ($i = 1, \ldots, p$, *p is number of processors*). After that, p_i constructs a local header table (*LHT*) from its own database. Then the master computing node (*MN*) aggregates the *LHT* from slave computing nodes (SN) to create a global header table (*GHT*). Consequently, *SN* creates a local FP-tree according to *GHT*. *MN* assigns each *SN* to the mining item by block distribution. Finally, each *SN* partially exchanges the *FP-tree* using the *FP-growth* mining algorithm to find all frequent patterns.

The main idea of the *PFP-tree* algorithm is using a special tree exchange technique. This technique reduces the repeated data exchange by grouping the *SNs*. The *SNs* need to communicate with each other at most $\log_p$ rounds. For example, processor p_i communicates with processor $p^{\%pr}_{(\frac{p}{2}+i)}$ in round r where $0 \leqslant i \leqslant p$ and $1 \leqslant r \leqslant \log_p$. However, too many trees need to be exchanged if the given threshold is small. Thus, for the worst case, more processors will lead to the execution time being longer, since too many subtrees are needed to send, receive and insert back.

2.3. Grid computing system

Grid computing is a loosely coupled distributed system, it allows for the sharing of processing powers, storage resources, and services from different geographical locations. Unlike the conventional high performance computing (e.g., Cluster computing) which is connected by a high speed network, grid computing nodes are connected by a variety of networks (e.g., gigabit network, internet, etc.). Since a grid connects various computing systems with different software and hardware specifications at different geographical locations, middleware plays an important role in integrating these resources. Globus, Sun Grid Engine, gLite, etc. are the best known grid middleware suppliers. Of these, Globus is the most popular and is widely used as open source grid middleware. There are many types of grid computing systems. Multi-cluster is the most popular and widely used. In the multi-cluster grid, resources are distributed across different networks on a multi-cluster grid. Moreover, each cluster can be a grid site with a grid head in each site. Jobs are dispatched to a grid head and the grid head then dispatched the jobs to the computing node inside a cluster according to its job scheduling algorithm. The administrator issues a certificate to the grid head and permits it to manage computing nodes inside the cluster. Moreover, when the cluster size varies, it only requires the grid head to adjust the setting instead of reconfiguring the entire grid system.

Since the Grid system has sizeable computing and storage resources. Some researchers have developed their data mining application on a Grid system (Cannataro, Talia, & Trunfio, 2002; Ciglaric, Pancur, Ster, & Dobnikar, 2005; Jiang & Yu, 2005). Cannataro et al. (2002) propose a knowledge grid system and discuss its use in distributed data mining services. Their study focused on the underlying framework and developed a distributing tool, based on JXTA peer-to-peer technology. Ciglaric et al. (2005) implemented the *Apriori* and the *FP-tree* algorithm in a grid environment. The results indicated benefits when using *Apriori* and *FP-tree* with a grid system. However, the proposed algorithm did not consider the balancing issue, therefore the performance did not achieve the ideal speedup when the number of processors increased.

3. Proposed parallel FP-tree algorithms

In this paper, parallel algorithms for frequent pattern mining on Cluster and Grid systems are proposed. In spite of the results of other research (Javed & Khokhar, 2004; Pramudiono & Kitsuregawa, 2003), there are still two important issues that need to be considered for a parallel algorithm to improve frequent pattern mining, one is reducing the communication cost and the other is balancing the computing node workload. In order to evaluate the execution time of different computing stages in detail, the *PFP-tree* (Javed & Khokhar, 2004) algorithm was implemented. Table 1 shows the execution time for each stage of the *PFP-tree*. It can be observed that the exchange stage dominated the others. Thus, the exchange stage was analyzed in depth. First, the exchange stage examined the candidate tree paths required for other processors, then exchanged the extracted paths with other processors and inserted it back to the local *FP-tree*. Therefore, the performance deteriorated with large databases or lower thresholds. Moreover, more processors also led to worse load balancing. Therefore, the performance can be improved significantly if the execution time of the exchange stage can be reduced and the workload of the processors can be balanced evenly.

The goal of our algorithm was to reduce the computation and communication cost of the exchange stage. Since extracting the candidate tree paths from an *FP-tree* data structure needs repeated traversing of the entire tree and inserting the tree paths back to the objective tree also requires repeated traversing of the trees, it becomes costly, leading to the *FP-tree* construction procedures being postponed. After creating the Header Table, the necessary information for parallel mining is exchanged in the transaction level of the *DB* instead of in the tree paths of the *FP-tree*.

However, indexing the necessary transactions is costly when the number of processors increases. For example, when there are n processors, processor p_i needs processing $mineSet_i$. $mineSet_i$ is items that block partitioned from header table for processor p_i. Therefore, processor p_0 should scan its database $|mineSet_1| + \cdots + |mineSet_n|$ times and then transfer to corresponding processors, to efficiently index the item in which transactions can speed up the execution processes. For that reason, transaction id (*TID*) was used to index the item. For a transactional database $DB = \{T_1, T_2, \ldots, T_n\}$ and each transaction $T_i \subseteq I$, $I = \{i_1, i_2, \ldots, i_m\}$, $TID(j) = \{k | i_j \cap T_k \neq \varphi,\ k = 1, \ldots, n\}$. After creating *TID*, transactions can be selected directly while the information for mining frequent patterns is exchanged.

3.1. Tidset-based parallel FP-tree (TPFP-tree) algorithm for cluster computing

Since finding all frequent patterns from transactional databases is a computation intensive problem, a parallel and distributed strategy could reduces the execution time and improve the mining performance. Therefore, the first parallel *FP-tree* algorithm based *TID* is developed for Cluster computing. Since a Cluster is homogeneous computing, the proposed algorithm distributes the workload to each processor evenly without considering the difference between processors. The main object is to reduce the execution time of mining information exchange and to shorten the index cost of transaction extraction. There are five primary stages in the Tidset-based Parallel FP-tree (TPFP-tree) algorithm: (1) create Header Table and *Tidset*, (2) distribute mining item set, (3) exchange transactions, (4) *FP-tree* and (5) *FP-growth*.

Firstly, although creating the header table needs only one database scan, when the database size is large, the execution time is still costly. Therefore, the *TPFP-tree* uses block distribution to partition the database and to distribute the divided database to corresponding computing nodes. Moreover, in order to directly select a transaction with corresponding item in subsequent procedures, a local transaction identification set (*Tidset*) is also created in this stage. After processing stage 1, frequent 1-itemset was found with a given threshold. Frequent 1-itemsets were also the mining items of the *TPFP-tree* algorithm. Then the mining items were equally distributed to the participating processors. Each processor was assigned $\left\lfloor \frac{n}{p} \right\rfloor$ items to mine for n frequent 1-itemset and p processors.

In order to build the *FP-tree* structure and to mine the frequent patterns with *FP-growth* on each processor independently, a processor should comprise the transactions which contain the assigned mining items from other processors. In the transaction exchanging stage, processor p_i scans its partial database to gather the transactions containing mining items required by other processors. However, it is costly since p_i must scan its database $p - 1$ times to gather all transactions. Hence, the *Tidset* is used to improve the transaction selecting. *Tidset* is a map between items and transaction, the transactions can be directly chosen from given items with *Tidset*. Since the *Tidset* table can be concurrently created with a frequent 1-itemset, the *Tidset* of each partial database is

Table 1
Execution time for each stage of the PFP-tree (t20.i04.d200k.n100k, threshold = 0.0005) for the different processors.

	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
Data transfer	0.91	0.04	0.06	0.12	0.06	0.05	0.08	0.04
Header table	0.6	0.59	0.6	0.59	0.6	0.6	0.6	0.59
All reduce	1.05	0.17	0.18	0.17	0.17	0.17	0.17	0.17
FP-tree	13.62	12.49	12.49	12.40	12.40	12.42	12.43	12.51
Exchange	98.29	157.11	204.78	233.51	241.07	235.06	223.06	197.02
FP-growth	18.06	26.34	27.09	31.1	24.51	22.44	20.07	12.59
Total	132.53	196.74	245.2	277.89	278.81	270.74	256.41	222.92

created in stage 1. Consequently, selected transactions are transferred to corresponding processors after gathering the transactions. Moreover, with the *Tidset* table, more processors do not lead to a worse performance.

After exchanging the mining items, a processor has the necessary transaction corresponding to its assigned mining items. Therefore, the processor can, independently, build the *FP-tree* and mine the frequent patterns by *FP-growth*. Finally, after completing the mining processes p_1 collects the frequent patterns from the others and merges them into all the frequent patterns. The detailed algorithm of the *TPFP-tree* is given below.

Input: a transaction database $DB = \{T_1, T_2, \ldots, T_n\}$, and each transaction $T_i \subseteq I$, $I = \{i_1, i_2, \ldots, i_m\}$. A given minimum threshold ξ. p is the number of processors. (p_1 is master node (*MN*), and $p_2, p_3, \ldots, p_p$ are salve nodes (*SN*s)).

Output: a complete set of frequent patterns, where $Sup(x_i) \geqslant \xi$, $\forall x_i$.

Method:

1. *MN* equally divides the database *DB* into *p* disjointed partitions $(DB_1, DB_2, \ldots, DB_p,\ \exists DB_1 \cup DB_2 \cup \cdots \cup DB_p = DB)$ and assigns DB_i to p_i.
2. Each processor p_i receives the database DB_i and scans the DB_i to create local header table (HT_i).
3. Each processor creates the local transaction identification set ($Tidset_i$) of DB_i.
4. Processors perform all-reduce of HT_i to get a global header table (GHT).
5. *MN* sorts items in GHT in descending order according to their support and block divides those items into mining set $MS_1, MS_2, \ldots, S_p$ where $MS_1 \cup MS_2 \cup \cdots \cup MS_p = \text{ItemsofGHT}$.
6. MN performs broadcast to distribute mining set information to all processors.
7. In order to create an FP-tree, each processor p_i has to obtain transaction T_{jk} on processor j $(j = 1, \ldots, p,\ j \neq i)$ such that $T_{jk} \cap MS_i \neq \varphi (k = 1, \ldots, |DB_j|)$. Since the mining set MS_i is partitioned statically, each processor knows the mining set of others. Moreover, $Tidset_i$ $(i = 1, \ldots, p)$ helps selecting the transactions directly in the local database. After that, each processor exchanges the transactions required for mining and $NewDB_i = DB_i \cup ReversedTransactions$.
8. Each processor p_i performs the FP-tree constructing procedure of *NewDB*.
9. Each processor p_i performs the FP-growth procedure to mine the given MS_i from their local FP-tree.
10. *MN* performs the MPI *All-Reduce* function to collect the frequent pattern from p_i $(i = 1, \ldots, p)$.

Fig. 1 is an example of a header and *Tidset* table for four processors. Fig. 1a shows the database equally partitioned into four parts with each transaction's local identity (*TID*). Fig. 1b depicts the created *Tidset* table of the database. From $Tidset_1$, item *F* appears in transaction 1 to 4 and item *H* appears in transactions 3 and 4 and so on. Moreover, the local header tables (*HT*) are also created at the same time (Fig. 1c). Finally, the processors performed all-reduce to get a global header table (*GHT*). After that, the master node (*MN*) sorted the *GHT* in descending order according to its support and divided items into mining sets (*MS*) using block distribution. Then *MN* broadcast the *MS*s to all processors.

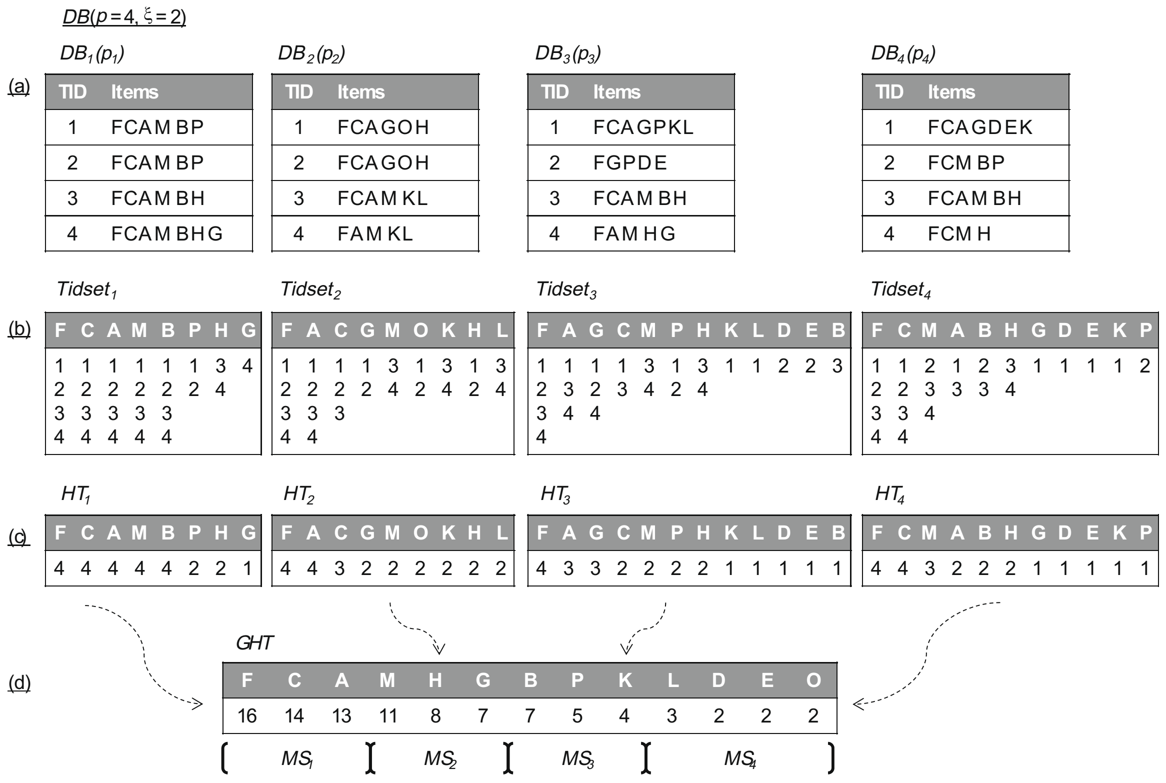
Then, each processor scanned its database to extract the transaction to transfer to the others. Fig. 2 shows the exchanging stage. Fig. 2a is the *MS* of each processor, and from Fig. 2b, p_0 had to prepare three tables which recorded items to be sent for exchange. Since it was costly to scan the database three times to create the table (Fig. 2b), the table using the *Tidset* was created beforehand (Fig. 1b). For example, p_1 sent the transaction containing *M, H, G* to p_2, according to $Tidset_1$, the union of item *M, H, G* was *TID* 1, 2, 3, and 4. Therefore, p_1 sent the transaction 1 to 4 to p_2. By the same process, essential transactions could be efficiently exchanged among processors. Since each processor had the necessary transaction for mining, each one could build an *FP-tree* and use *FP-growth* to find frequent patterns independently. Finally, *MN* gathered the frequent patterns created by each processor to produce the all the frequent patterns.

3.2. Balanced Tidset Parallel FP-tree (BTP-tree) algorithm for grid computing

Since a Grid system consists of heterogeneous computing resources, storage, and network connections, the computation ability and network bandwidth of computing nodes are varied. The proposed *TPFP-tree* designed for Cluster computing, thus executed on a Grid system causes load imbalance and increases idle time. Therefore, a Balanced Tidset Parallel FP-tree (*BTP-tree*) which considers the computation capability of participating nodes and dispatches the mining items to computing nodes according for load balancing is proposed. However, to evaluate the performance of processors according to the detected hardware specification is useless. For example, one processor of 1.8 GHz with Intel Core 2 architecture may be faster than one of 4 GHz with Pentium 4 architecture. The wrong performance index will lead to wrong workload distribution. Moreover, different types of applications require different types of computation resources. Therefore, B*TP-tree* solves this problem by executing a small amount of data on the processors to establish performance indices. The performance indices obtained this way can truly respond to the computation ability of a given application. There are six stages in a *BTP-tree* algorithm: (1) create header table and *Tidset*, (2) evaluate the performance index of computing nodes, (3) distribute mining item set, (4) exchange transactions, (5) create *FP-tree* and (6) *FP-growth*.

Creating a header table requires lots of computing time when the database is large. Therefore, a transaction identification set (*Tidset*) is created at this stage to speed-up the transaction selection for future use. After creating the head table, 1-frequent item set was found. Moreover, these items were also the target mining items in *FP-growth*.

Since the Grid system is a heterogeneous computing system, the processors' capability and memory size are different. Distributing



(a) $DB_1(p_1)$

TID	Items
1	FCAM BP
2	FCAM BP
3	FCAM BH
4	FCAM BHG

$DB_2(p_2)$

TID	Items
1	FCAGOH
2	FCAGOH
3	FCAM KL
4	FAM KL

$DB_3(p_3)$

TID	Items
1	FCAGPKL
2	FGPDE
3	FCAM BH
4	FAM HG

$DB_4(p_4)$

TID	Items
1	FCAGDEK
2	FCM BP
3	FCAM BH
4	FCM H

(b) $Tidset_1$

F	C	A	M	B	P	H	G
1	1	1	1	1	1	3	4
2	2	2	2	2	2	4	
3	3	3	3	3			
4	4	4	4	4			

$Tidset_2$

F	A	C	G	M	O	K	H	L
1	1	1	1	3	1	3	1	3
2	2	2	2	4	2	4	2	4
3	3	3						
4	4							

$Tidset_3$

F	A	G	C	M	P	H	K	L	D	E	B
1	1	1	1	3	1	3	1	1	2	2	3
2	3	2	3	4	2	4					
3	4	4									
4											

$Tidset_4$

F	C	M	A	B	H	G	D	E	K	P
1	1	2	1	2	3	1	1	1	1	2
2	2	3	3	3	4					
3	3	4								
4	4									

(c) HT_1

F	C	A	M	B	P	H	G
4	4	4	4	4	2	2	1

HT_2

F	A	C	G	M	O	K	H	L
4	4	3	2	2	2	2	2	2

HT_3

F	A	G	C	M	P	H	K	L	D	E	B
4	3	3	2	2	2	2	1	1	1	1	1

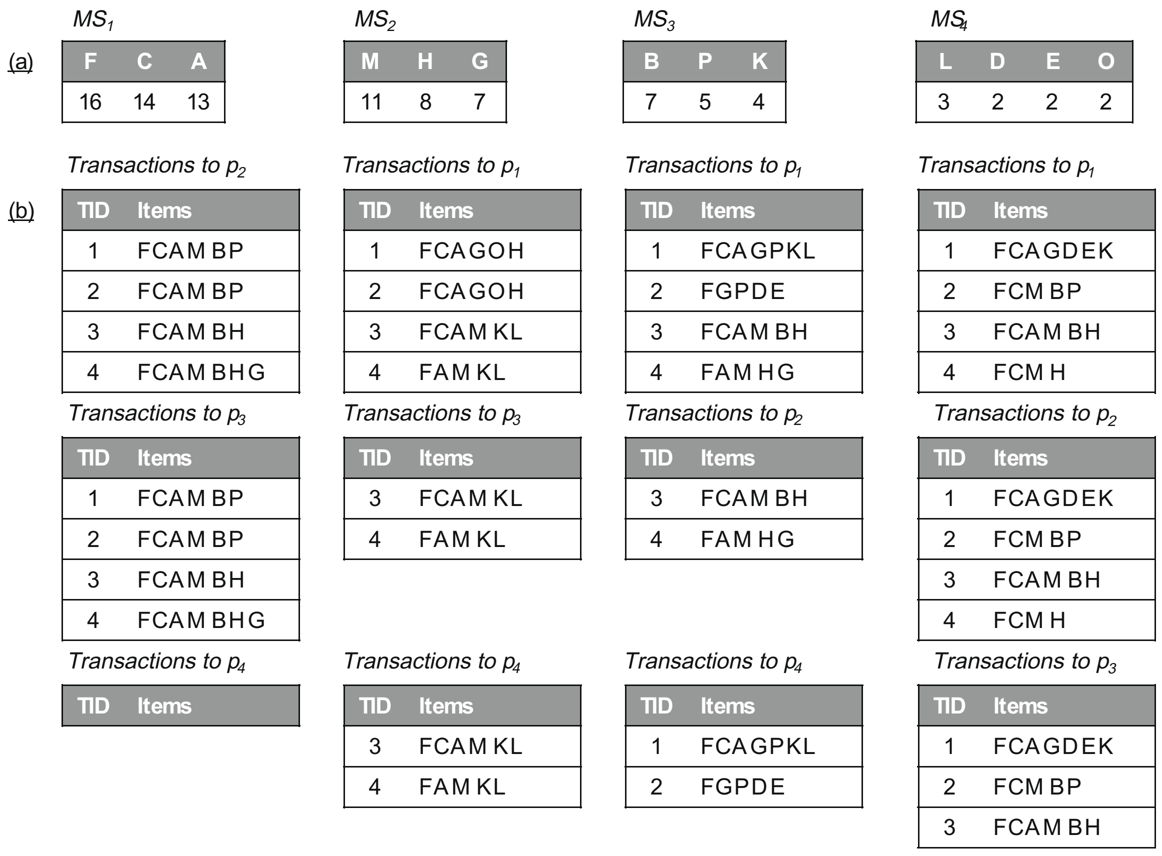
HT_4

F	C	M	A	B	H	G	D	E	K	P
4	4	3	2	2	2	1	1	1	1	1

(d) GHT

F	C	A	M	H	G	B	P	K	L	D	E	O
16	14	13	11	8	7	7	5	4	3	2	2	2

Fig. 1. Example of *DB* partitioning into 4 processors with the given threshold ξ.



(a) MS_1

F	C	A
16	14	13

MS_2

M	H	G
11	8	7

MS_3

B	P	K
7	5	4

MS_4

L	D	E	O
3	2	2	2

(b) Column 1 (MS_1)

Transactions to p_2

TID	Items
1	FCAM BP
2	FCAM BP
3	FCAM BH
4	FCAM BHG

Transactions to p_3

TID	Items
1	FCAM BP
2	FCAM BP
3	FCAM BH
4	FCAM BHG

Transactions to p_4

TID	Items

Column 2 (MS_2)

Transactions to p_1

TID	Items
1	FCAGOH
2	FCAGOH
3	FCAM KL
4	FAM KL

Transactions to p_3

TID	Items
3	FCAM KL
4	FAM KL

Transactions to p_4

TID	Items
3	FCAM KL
4	FAM KL

Column 3 (MS_3)

Transactions to p_1

TID	Items
1	FCAGPKL
2	FGPDE
3	FCAM BH
4	FAM HG

Transactions to p_2

TID	Items
3	FCAM BH
4	FAM HG

Transactions to p_4

TID	Items
1	FCAGPKL
2	FGPDE

Column 4 (MS_4)

Transactions to p_1

TID	Items
1	FCAGDEK
2	FCM BP
3	FCAM BH
4	FCM H

Transactions to p_2

TID	Items
1	FCAGDEK
2	FCM BP
3	FCAM BH
4	FCM H

Transactions to p_3

TID	Items
1	FCAGDEK
2	FCM BP
3	FCAM BH

Fig. 2. Example of the exchange stage of 4 processors.

mining items equally increases the execution time and causes some computing nodes to be idle. In order to solve this problem, the target mining items were partitioned according to the performance index (*PI*). Since mining frequent patterns is a computation intensive job, only some transactions were picked to execute the *FP-tree* and *FP-growth* creation procedure. Later, the *PI* of each processor can be determined and the mining items can be divided according to the index. For example, there are n frequent one-itemset, p processors, and PI_i represents the performance index of processors p_i (higher is better) $\forall i = 1, \ldots, p$. Thus, for processor p_i will be assigned $n \times \frac{PI_i}{\sum_{k=1}^{p} PI_k}$ items to mine.

The next stage is the transaction exchange stage. Each processor scans the local database via *TID* to select the transactions from the local database to send to corresponding computing nodes. After exchange, each processor creates the *FP-tree* and *FP-growth* via the local database.

Finally, the frequent patterns are obtained after completing the *FP-tree* and the *FP-growth*. The detailed algorithm of the *BTP-tree* is given below.

Input: a transaction database $DB = \{T_1, T_2, \ldots, T_n\}$ and each transaction $T_i \subseteq I$, $I = \{i_1, i_2, \ldots, i_m\}$ A given minimum threshold ξ. p is the number of processors. (p_1 is master node (*MN*), and $p_2, p_3, \ldots, p_p$ are salve nodes (*SNs*)).

Output: a complete set of frequent patterns, where $Sup(x_i) \geqslant \xi$, $\forall x_i$.

Method:

1. MN equally divides the database *DB* into p disjointed partitions $(DB_1, DB_2, \ldots, DB_p \exists DB_1 \cup DB_2 \cup \ldots \cup DB_p = DB)$ and assigns DB_i to p_i.
2. Each processor p_i receives the database DB_i and scans it to create the local header table (HT_i).
3. Each processor creates the local transaction identification set ($TIDSET_i$) of DB_i.
4. Processors perform MPI All-Reduce of HT_i to get a global header table (GHT).
5. MN sends first 1000 transactions to each processor, after that p_i executes the FP-tree construction procedure and records the execution time as t_i.
6. Let performance index $PI_i = 1/t_i$.
7. MN gathers all PI and let $PI_{total} = \sum_{i=0}^{p} PI_i$.
8. MN sorts items in GHT in descending order according to their support and these items into mining sets according to performance index $\left(MS_i = \frac{PI_i}{PI_{total}} \times ItemsOfGHT\right)$ where $MS_1 \cup MS_2 \cup \cdots \cup MS_p = ItemsOfGHT$.
9. MN performs broadcast to distribute mining set information to all processors.
10. In order to create an FP-tree, each processor p_i has to obtain transaction T_{jk} on processor j ($j = 1, \ldots, p$, $j \neq i$) such that $T_j \cap MS_i \neq \phi (k = 1, \ldots, |DB_j|)$. Since the mining set MS_i is partitioned statically, each processor knows the mining set of others. Moreover, $TIDSET_i$ ($i = 1, \ldots, p$) helps selecting the transactions directly in the local database. After that, each processor exchanges the transactions required for mining and $NewDB_i = DB_i \cup ReceivedTransactions$.
11. Each processor p_i performs the FP-tree constructing procedure of $NewDB_i$.
12. Each processor p_i performs the FP-growth procedure to mine the given MS_i from their local FP-tree.
13. MN performs the MPI All-Reduce function to collect the frequent pattern from p_i ($i = 1, \ldots, p$)

4. Experimental results

In order to evaluate the performance of the proposed algorithms, the *PFP-tree*, *TPFP-tree* and *BTP-tree* were implemented along with Message Passing Library 2 (MPICH2) on Ubuntu with Linux kernel 2.6. Synthesized datasets generated by IBM's Quest Synthetic Data Generator (Almaden, xxxx) were used to verify the algorithm.

4.1. TPFP-tree on PC Cluster

The program was executed in a PC Cluster with 16 computing nodes. Table 2 gives the hardware and software specifications. As can be seen, each computing node had the same hardware and software environment. To verify the performance of our algorithm, the thresholds 0.001, 0.0005, 0.0001 were used to examine the *TPFP-tree* with different datasets. Table 3 shows the dataset used to verify the performance of the algorithm. The algorithm was evaluated with a different number of items, a different number of average transaction lengths and a different number of transactions. As can be seen from the experimental results, frequent patterns were found for dataset *t10.i04.d100k.n100k* with threshold 0.01. Therefore, the smaller threshold was used to make sure that there were frequent patterns. There were 8112, 53,713, and 234,649 frequent patterns for threshold 0.001, 0.0005, and 0.0001, respectively for *t10.i04.d100k.n100k* dataset.

Figs. 3 and 4 show the execution time of the *TPFP-tree* and the *PFP-tree* with different datasets. From the results in Fig. 3, it can be seen that the proposed algorithm performed better than the *PFP-tree* regardless of the number of processors. Moreover, the results also illustrated that, when the average items per transaction increased, the execution time decreased noticeably. Fig. 4 also

Table 2
Hardware and software specification of PC Cluster.

Hardware specification	
CPU	AMD Athlon XP 2000+
Memory	1 GB DDR RAM
Network	100 Mbps interconnection network
Disk	80 GB IDE H.D.
Software configuration	
OS and compiler	Linux 2.6
	Gcc/G++ 4.03
Message Passing Library	MPICH2 1.0.5
	mpi4py 0.5

Table 3
Statistical characteristics of datasets.

Dataset name	No. of items	Avg. items	No. of trans.
t10.i04.d100k.n100k	100k	10	100k
t10.i04.d200k.n100k	100k	10	200k
t15.i04.d100k.n100k	100k	15	100k
t15.i04.d200k.n100k	100k	15	200k
t20.i04.d050k.n100k	100k	20	50k
t20.i04.d100k.n100k	100k	20	100k

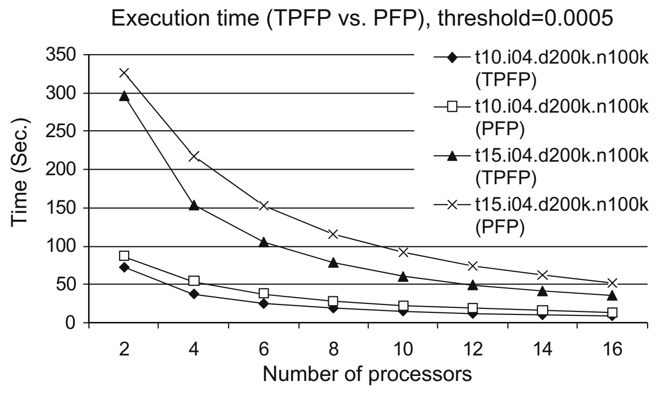


Fig. 3. Execution time (TPFP vs. PFP), threshold = 0.0005.

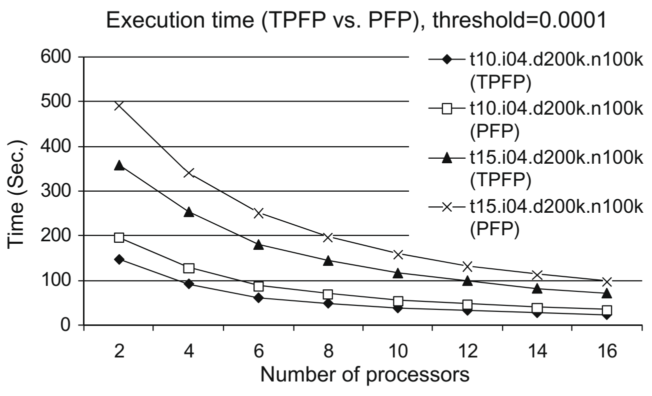


Fig. 4. Execution time (TPFP vs. PFP), threshold = 0.0001.

Table 5
Hardware and software specification of multi-cluster grid.

	Cluster 1	Cluster 2	Cluster 3
	Hardware specification		
Number of Nodes	5	5	3
CPU	Pentium 4 3.2G	AMD XP 2.0G	Pentium 4 3.0G
Memory	512 MB	1024 MB	1024 MB
Network	100 Mbps interconnection network		
			Software configuration
OS and compiler			Linux 2.6 Gcc/G++ 4.03 Globus Toolkit 4.0
Message Passing Library			MPICH-G2 MPICH2 1.0.5 mpi4py 0.5

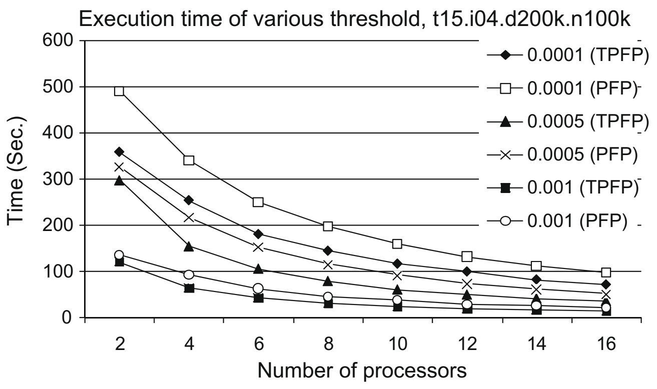


Fig. 5. Execution time of various thresholds.

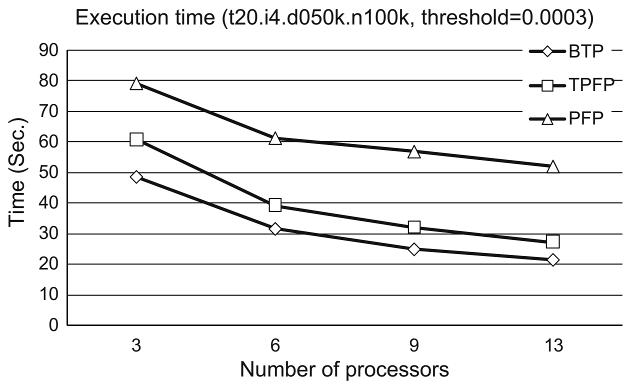


Fig. 6. Execution time of different processors (d50k).

shows that the algorithm saved on execution time compared to the *PFP-tree*. Fig. 5 illustrates the execution time of various thresholds showing that this algorithm reduced the execution time for different given thresholds.

In order to further compare the *TPFP-* and *PFP-tree*, $SU_2(n) = \frac{\text{Execution time of 2 processors}}{\text{Execution time of } n \text{ processors}}$ was defined. Table 4 shows the comparisons of the *two trees* with a given threshold of 0.0005. It can be seen that the ratio SU_2 of the *TPFP-tree* was better than that of the *PFP-tree* for all of the test cases. This means that the *TPFP-tree* had better scalability than the *PFP-tree* with a different number of processors and different datasets. Moreover, it can be observed that the speedup ratio SU_2 improved with a larger number of transactions.

Table 4
Speed-up ratio (SU_2) of TPFP and PFP.

Itemset	Method	Number of processors							
		2	4	6	8	10	12	14	16
t10.i04.d050k.n100k	TPFP	1	1.67	2.53	3.04	3.88	4.07	4.62	4.62
	PFP	1	1.53	2.12	2.7	3.3	3.59	3.95	4.16
t10.i04.d100k.n100k	TPFP	1	1.87	2.7	3.47	4.31	5.11	6.33	6.79
	PFP	1	1.53	2.12	2.7	3.3	3.59	3.95	4.16
t10.i04.d200k.n100k	TPFP	1	1.94	2.94	3.87	4.89	6.23	7.3	8.41
	PFP	1	1.62	2.2	2.95	3.61	4.21	5.25	5.61
t15.i04.d050k.n100k	TPFP	1	1.6	2.29	2.98	3.55	4.12	4.66	5.39
	PFP	1	1.47	2.02	2.52	3.1	3.59	4.14	4.73
t15.i04.d100k.n100k	TPFP	1	1.72	2.48	3.22	4.19	4.82	5.83	6.66
	PFP	1	1.47	2.02	2.59	3.37	4.08	4.8	5.43
t15.i04.d200k.n100k	TPFP	1	1.93	2.81	3.76	4.9	5.98	7.11	8.24
	PFP	1	1.5	2.14	2.83	3.59	4.44	5.29	6.35
t20.i04.d050k.n100k	TPFP	1	1.57	2.12	2.65	3.22	3.71	4.28	4.86
	PFP	1	1.38	1.87	2.32	2.88	3.23	3.79	4.43
t20.i04.d100k.n100k	TPFP	1	1.59	2.22	2.78	3.41	3.97	4.74	5.25
	PFP	1	1.41	1.9	2.44	2.92	3.57	4.13	4.62
t20.i04.d200k.n100k	TPFP	1	1.62	2.27	2.96	3.67	4.42	5.14	5.91
	PFP	1	1.4	1.93	2.5	3.06	3.69	4.46	5.02

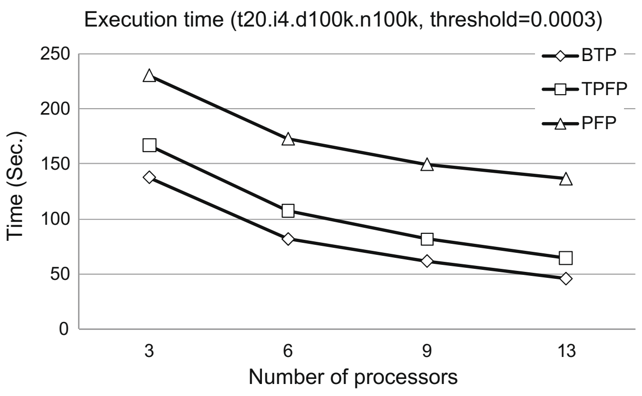


Fig. 7. Execution time of different processors (d100k).

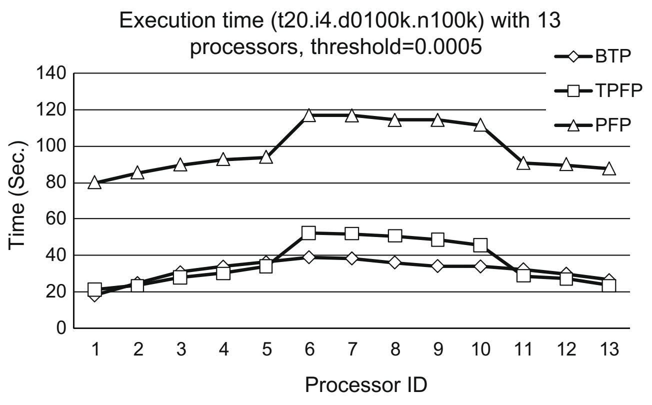


Fig. 10. Execution time of each processor (d050k).

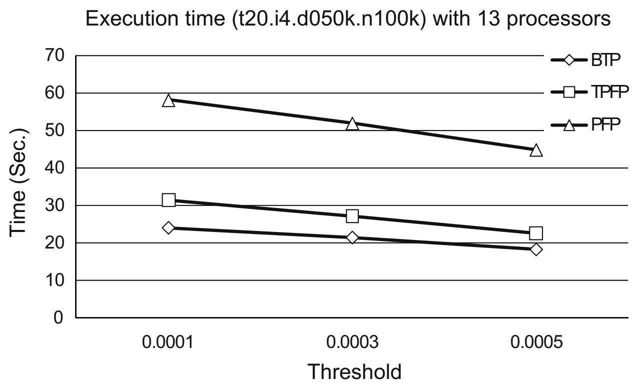


Fig. 8. Execution time of different threshold (d050k).

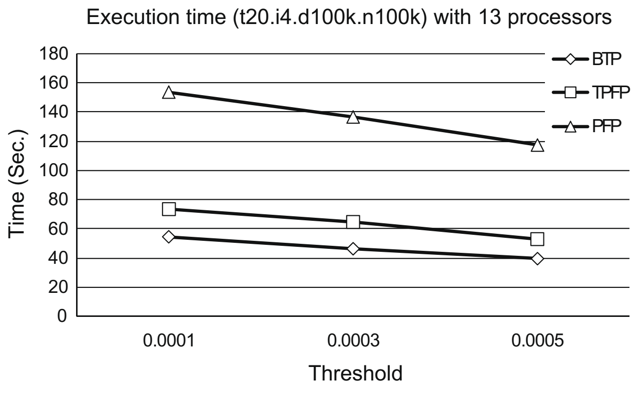


Fig. 9. Execution time of different threshold (d100k).

Table 6
Execution time of BTP, TPFP, and PFP.

Dataset	Method	Number of processors			
		3	6	9	13
Threshold: 0.0005					
t10.d050k	BTP	7.73	5.84	5.68	6.43
	TPFP	9.4	6.66	6.16	6.72
	PFP	12.04	10.21	8.35	8.2
t10.d100k	BTP	15.99	10.95	9.68	9.9
	TPFP	20.05	13.14	11.25	10.72
	PFP	28.6	22	18	17.31
t20.d050k	BTP	38.93	26.03	21	18.27
	TPFP	51.4	33.8	28.03	22.54
	PFP	66.31	52.99	49.37	44.8
t20.d100k	BTP	111.11	65.45	50.42	39.41
	TPFP	138.27	90.11	69.67	52.65
	PFP	192.61	148.21	129.03	117.32
Threshold: 0.0003					
t10.d050k	BTP	10.84	7.89	7.11	7.65
	TPFP	14.57	9.79	9.14	8.98
	PFP	18.59	15.46	12.51	12.65
t10.d100k	BTP	25.04	15.02	12.84	12.29
	TPFP	32.37	19.5	17.3	14.43
	PFP	45.84	33.65	27.09	26.28
t20.d050k	BTP	48.34	31.46	24.85	21.43
	TPFP	60.92	39.18	32.01	27.1
	PFP	79.16	61.19	56.75	51.87
t20.d100k	BTP	138.07	82.14	62.18	46.22
	TPFP	166.78	107.82	81.81	64.66
	PFP	230.14	172.8	149.78	136.69
Threshold: 0.0001					
t10.d050k	BTP	15.86	11.46	10.32	10.89
	TPFP	20.29	13.96	11.92	11.55
	PFP	25.7	21	18.03	17.04
t10.d100k	BTP	36.05	21.91	19.14	16.86
	TPFP	46.49	29.17	23.76	18.9
	PFP	62.45	46.17	41.85	38.62
t20.d050k	BTP	56.47	35.84	29.5	24
	TPFP	68.04	46.88	37.44	31.36
	PFP	87.63	69.98	61.77	58.05
t20.d100k	BTP	162.18	98.38	72.31	54.47
	TPFP	197.38	127.13	96.48	73.8
	PFP	258.85	198.49	169.08	153.85

4.2. BTP-tree on multi-cluster grid

For the *BTP-tree* algorithm, the performance was verified on a multi-cluster grid system consisting of three Linux based PC Clusters. Table 5 depicts the hardware and software specifications showing that each cluster had different computing capability and different memory size. Different thresholds, 0.0005, 0.0003, 0.0001 were used with datasets *t10.i4.d050k.n100k*, *t10.i4.d100k.n100k*, *t20.i4.d050k.n100k*, and *t20.i4.d100k.n100k* to examine the *BTP-tree* in order to make sure that there were frequent patterns.

In order to construct a heterogeneous computing environment for 3 processors, one node from each cluster was selected. For 6 and 9 processors, 2 and 3 nodes were selected from each cluster. Finally, for 13 processors, all processors in Table 5 were used.

Figs. 6 and 7 show the execution time of the *BTP-tree*, *TPFP-tree*, and *PFP-tree* with a different number of processors with 50k and

100k transactions, respectively. From the results of Fig. 6, it is clear that *BTP* performed better than the *TPFP* and *PFP* regardless of how many processors were assigned. Moreover, it can be seen that *BTP* reduced about 20% and 58% on *TPFP* and *PFP*. Fig. 6 also shows that *BTP* saved on execution time compared to *TPFP* and *PFP*.

Figs. 8 and 9 illustrate the execution time of various thresholds with 50k and 100k transactions, respectively. This confirmed that *BTP* can efficiently reduce the execution related to *TPFP* and *PFP*. Moreover, it can be seen that in a heterogeneous computing environment (three types of CPU in this case) balancing the workload can reduce the execution time.

Fig. 10 shows that *BTP* could balance the workload to save execution time. The experimental results showed *BTP* had better balancing capability and it could save about 25% and 66% on the execution time needed with *TPFP* and *PFP*.

Table 6 shows the complete execution time for dataset *t10.i4.d050k.n100k*, *t10.i4.d100k.n100k*, *t20.i4.d050k.n100k*, and *t20.i4.d100k.n100k* with threshold 0.0005, 0.0003, and 0.0001. Here it can be seen that *BTP-tree* performed better than the *TPFP-tree* and *PFP-tree* for all test cases. Moreover, the *BTP-tree* shortened execution time with more processors with different thresholds and datasets. Consequently, with a smaller threshold *BTP* performed significantly better than the others.

5. Conclusions

Mining frequent patterns from a transaction-oriented database is important in data mining research. Many methods have been proposed to solve this problem, and some of them have been developed for a parallel-distributed computing system. However, the execution time increases significantly with an increase in database size and a decrease in the given threshold. In this paper, two parallel algorithms Tidset-based Parallel FP-tree (*TPFP-tree*) and Balanced Tidset-based Parallel FP-tree (*BTP-tree*) were developed to solve frequent pattern mining problems. *TPFP-tree* is based on *FP-tree* data structure for a Cluster system, it exchanges necessary information for mining before tree construction to improve the performance. Moreover, it also uses a *TID* set to select transactions directly instead of scanning the database repeatedly. Furthermore, *BTP-tree* was proposed for solving mining problems on Grid systems. *BTP-tree* calculates the performance index (*PI*) of each computing node, then dispatches mining items to a computing node according to its *PI*. The experimental results show that *TPFP-tree* performed better than *PFP-tree* on a Cluster system judged upon a variety of computing nodes, database sizes or different given thresholds. Additionally, the results also show that *BTP-tree* had better load balancing ability and it performed better than *TPFP-tree* and *PFP-tree* on a Grid system. Moreover, it also reduced both execution and idle time.

References

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules in large database. In *Proceedings of the 20th international conference on very large data base* (pp. 487–499).

Almaden. Quest synthetic data generation code. <http://www.almaden.ibm.com/cs/quest/syndata.html>.

Cannataro, M., Talia, D., & Trunfio, P. (2002). Distributed data mining on the grid. *Future Generation Computer Systems, 18*(8), 1101–1112.

Chen, M.-C., Huang, C.-L., Chen, K.-Y., & Wu, H.-P. (2005). Aggregation of orders in distribution centers using data mining. *Expert System with Applications, 28*(3), 453–460.

Ciglaric, M., Pancur, M., Ster, B., & Dobnikar, A. (2005). Data mining in grid environment. *Adaptive and Natural Computing Algorithm*, 522–525.

Coenen, F., Leng, P., & Ahmed, S. (2004). Data structure for association rule mining: T-trees and P-trees. *IEEE Transactions on Knowledge and Data Engineering, 16*(6), 774–778.

Foster, I., & Kesselman, C. (1998). *The grid: blueprint for a new computing infrastructure*. Morgan Kaufmann.

Gorodetsky, V., Karasaeyv, O., & Samoilov, V. (2003). Multi-agent technology for distributed data mining and classification. In *Proceedings of the IEEE/WIC international conference on intelligent agent technology* (pp. 438–441).

Han, J., Pei, J., Yin, Y., & Mao, R. (2004). Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Journal of Data Mining and Knowledge Discovery, 8*(1), 53–87.

Holt, J. D., & Chung, S. M. (2004). Parallel mining of association rules from text databases on a cluster of workstations. In *Proceedings of 18th international symposium on parallel and distributed processing* (pp. 86).

Hong, T.-P., Lin, C.-W., & Wu, Y.-L. (2008). Incrementally fast updated frequent pattern trees next term. *Expert System with Applications, 34*(4), 2424–2435.

Javed, A., & Khokhar, A. (2004). Frequent pattern mining on message passing multiprocessor systems. *Distributed and Parallel Database, 16*(3), 321–334.

Jiang, W.-S., & Yu, J.-H. (2005). Distributed data mining on the grid. In *Proceedings of the fourth international conference on machine learning and cybernetics* (pp. 18–21).

Lazcorreta, E., Botella, F., & Fernández-Caballero, A. (2008). Towards personalized recommendation by two-step modified Apriori data mining algorithm. *Expert System with applications, 35*(3), 1422–1429.

Li, T., Zhu, S., & Ogihara, M. (2003). A new distributed data mining model based on similarity. In *Symposium on applied computing* (pp. 432–436).

Lin, C.-R., Lee, C.-H., Chen, M.-S., & Yu, P. S. (2002). Distributed data mining in a chain store database of short transactions. In *Conference on knowledge discovery in data* (pp. 576–581).

Lin, C.-W., Hong, T.-P., & Lu, W.-H. (2009). The Pre-FUFP algorithm for incremental mining. *Expert System with Applications, 36*(5), 9498–9505.

Park, J. S., Chen, M.-S., Yu, P. S. (1995). An effective hash-based algorithm for mining association rules. In *Proceedings of the ACM SIGMOD* (pp. 175–186).

Pramudiono, I., & Kitsuregawa, M. (2003). Shared nothing parallel execution of FP-growth. *DBSJ Letters, 2*(1), 43–46.

Tang, P., & Turkia, M. P. (2006). Parallelizing frequent itemset mining with FP-Trees. In *Proceeding of the 21st international conference on computers and their applications*. (pp. 30–35).

Yan, X., Zhang, C., & Zhang, S. (2009). Genetic algorithm-based strategy for identifying association rules without specifying actual minimum support. *Expert System with Applications, 36*(2), 3066–3076.

Zhou, J., & Yu, K.-M. (2008). Tidset-based parallel FP-tree algorithm for the frequent pattern mining problem on PC clusters. In *Proceeding of 3rd international conference on grid and pervasive computing* (pp. 18–28).

Parallel Branch-and-Bound Approach with MPI Technology in Inferring Chemical Compounds with Path Frequency

Kun-Ming Yu
Department of Computer Science and Information Engineering, Chung Hua University
yu@chu.edu.tw

Hui-Yuan Wang
Department of Computer Science and Information Engineering, Chung Hua University
wang@ pdlab.csie.chu.edu.tw

Jiayi Zhou
Institute of Engineering and Science, Chung Hua University
jyzhou@ pdlab.csie.chu.edu.tw

Chun-Yuan Lin
Department of Computer Science and Information Engineering, Chang Gung University
cyulin@mail.cgu.edu.tw

Chuan Yi Tang
Department of Computer Science, National Tsing Hua University
cytang@cs.nthu.edu.tw

Abstract

Drug design is the approach of finding drugs by design using computational tools. When designing a new drug, the structure of the drug molecule can be modeled by classification of potential chemical compounds. Kernel Methods have been successfully used in classifying potential chemical compounds. Frequency of labeled paths has been proposed to map compounds into feature in order to classify the characteristics of target compounds. In this study, we proposed an algorithm based on Kernel method via parallel computing technology to reduce computation time. This less constrain of timing allows us to aim at back tracking a full scheme of all of the possible pre-images, regardless of their difference in molecular structure, only if they shared with the same feature vector. Our method is modified on BB-CIPF and used MPI to reduce the computation time. The experimental results show that our algorithms can reduce the computation time effectively for chemical compound inference problem.

Keyword: chemical compound inference, parallel branch-and-bound, MPI

1. Introduction

Drug design is a very valuable issue in the chemogenomics [1]. To classify appropriately characteristic, classification of drug are important in designing new drug. Quantitative structure activity relationship was used to classify the chemical compounds by many researchers. Support Vector Machines (*SVM*s) [2, 3] of Kernel methods [4, 5, 6, 7] have been widely used in various classification problems of chemogenomics. Kernel method is usually required to develop a mapping from the set of objects in the target problem to a *feature space* and a kernel function is defined as an inner product between two *feature vectors*. In order to apply kernel methods, it is usually required to develop a mapping from the set of objects in the target problem to a feature space. An object will be defined as a feature vector in the feature space by Kernel methods, and then *SVM*s can be employed to learn the classification rules. Feature vectors have been successfully used based on *frequency of labeled paths* [8, 9] or *frequency of small fragments* [4, 5].

A desired object is computed as a point in the feature space using suitable function and then the point is mapped back to the input space, where this mapped back object is called a *pre-image*.

Let φ be a function of mapping from an input space G to a feature space F. The pre-image problem [18] is, given a point y in F, to find x in G such that $y = \varphi(x)$, through a proper function, if the feature vector can be mapped backward to an object from y such as $y = \varphi(x)$, where such x is called a pre-image.

For example, if we want to infer a graph from numbers of occurrences of vertex-labeled paths [10,11]. In [10], a feature vector g is a multiple set of strings of labels with length at most K which represents the path frequency. Given a feature vector g, they considered the problem of finding a vertex-labeled graph G that attains a one-to-one correspondence between g and the set of sequences of labels along all paths of length at most K in G. In this study, we took into compound structure of bigger size. This will spend more time in inferring a pre-image from path frequency of g.

Parallel computing makes more computing resources than a single processor [13]. In science and engineering, some applications (the complex challenge problems) are computationally bounded. Also, the new application areas where large amounts of computation can be put to profitable by using parallel computing, such as data mining and optimization.

Processing computations in a parallel way is natural and intuition, because the real world is naturally parallel. Parallel computing has allowed complex problems to be solved and high-performance applications to be implemented in science and engineering area, or in new application areas [13].

In this study, we want to reduce the computing time in inferring chemical compounds with path frequency. We developed a parallel computing method by modifying the algorithm published by Akutsu and Fukagawa[12]. The main concept is to assign independent tasks to different computing nodes expect for reducing computing time.

The rest of this paper is organized as follows. Section II introduces the background about problem and definition. Next, we present our algorithm and description in section III. In section IV, we show the experiment results. Finally we conclude this paper in section VI.

2. Related Work

2.1 The BB-CIPF algorithm

Terms of compound characteristics such as Frequency of labeled paths [8, 9] or frequency of small fragments [4, 5] are used by some researchers to classify compound. Extending from inferring a tree from walks [14] or graphic reconstruction problem [15], *Kernel Principal Component Analysis and Regression* [4] and *stochastic search algorithms* [5] are used to find pre-images. However, in previous cases, the obtained results and performance of these algorithms were not thoroughly verifies against more complex compound cases.

Figure 1 shows that giving a target x to find $\varphi(x)$ with kernel method and then inferring the compound. Inferring a chemical structure from a feature vector based on frequency of labeled-paths and small fragments, *Branch-and-Bound Chemical compound Inference from Path Frequency* (*BB-CIPF*) [12] is used to infer chemical compounds of tree-like structures. BB-CIPF algorithm extends from [10, 11]. BB-CIPF uses tree-like structures and infers chemical compound. Chemical compounds are assigned with a feature vector by the algorithm based on frequency of small fragments. Moderate size chemical compounds are inferred.

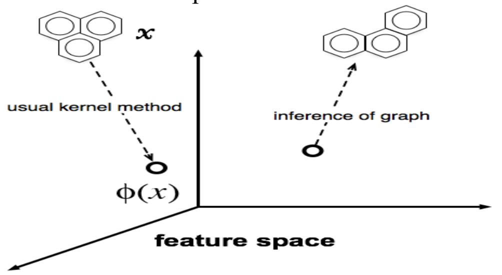
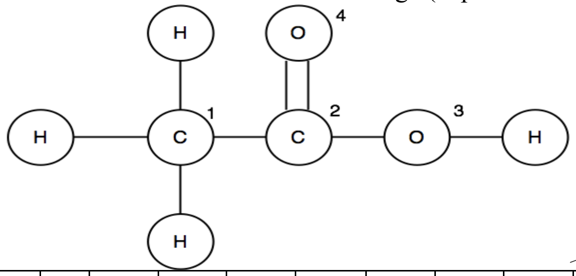


Figure 1: Inferring a Chemical Structure from a Feature Vector

The pre-image problem was defined as follows. Let be an alphabet and $_K$ be the set of strings with length K over . For a string t and a graph G, $occ(t, G)$ denotes the number of occurrences of substring t in G. Then, the feature vector $f_K(G)$ of level K for G is a — $_K$—-dimensional integer vector such that the coordinate indexed by $t \in$ $_K$ is $occ(t, G)$. That is, $f_K(G)$ is defined by $f_K(G) = (occ(t, G))_{t \in \Sigma K}$. For example, consider a compound $C_2H_4O_2$ (see figure2) over $\Sigma = \{C, O, H\}$ and the K value is 1. Then, $f_K(G) = (2,2,4,2,2,3,2,0,1,3,1,0)$ because occ(C,G) = 2, occ(O,G) = 2, occ(H,G) = 4, occ(CC,G) = 2, occ(CO,G) = 2, and so on. If K is large, the number of dimensions of a feature vector will be large (exponential of K).



C	O	H	CO	CH	OC	OO	OH	HC	HO	HH
2	2	4	2	3	2	0	1	3	1	0

Figure 2: An illustration of a multitree G and its feature vector

Give a target compound T^{target} and T^{cur} is inferred to T^{target}. T^{cur} insert a node n becoming T^{next}. f^{target} is the feature vector of T^{target} and f^{next} is the feature vector of T^{next}. After inserting a node compare T^{target} and T^{next}. If the feature vector f^{next} of T^{next} does not match with the feature vector f^{target} of T^{target}, the T^{next} will be discarded and T^{next} will not continue to carry out the evolution of T^{next}. T^{cur} may be re-inserted into another node for comparison with T^{target}.

The concept of branch-and-bound chemical compound inference from path frequency algorithm is inferring tree-like structures of chemical compounds. Back tracking a full scheme of all of the possible pre-images, regardless of their difference in molecular structure, if they shared with the same feature vector to come close to the reality of the development of algorithms for drug design.

The BB-CIPF algorithm will track back pre-images as the solution of partial results. For example, giving a target compound, if there are three objects having the same feature vector, then those are the partial results.

Our method is based on BB-CIPF. We will find all possible compounds with same feature vector. To reduce the computing time, the method we proposed used MPI to solve the chemical compound inference problem.

2.2 Message Passing Interface

Message Passing Interface has already used in solving chemistry problems [16]. MPI (Message Passing Interface) is a specification for a standard library for message passing that was defined by the MPI Forum, a broadly based group of parallel computer vendors, library writers, and applications specialists. Multiple implementations of MPI have been developed [17].

The message-passing model of parallel computation has emerged as an expressive, efficient, and well-understood paradigm for parallel programming. Until recently, the syntax and precise semantics of each message-passing library implementation were different from the others, although many of the general semantics were similar. The proliferation of message-passing library designs from both vendors and users was appropriate for a while, but eventually it was seen that enough consensus on requirements and general semantics for message-passing had been reached that an attempt at standardization might usefully be undertaken. MPI is a message-passing application programmer interface, which including protocol and semantic specifications for behaving implementation in the feature (such as a message buffering and message delivery progress requirement).

3. Parallel BB-CIPF (PB-CIPF)

In the previous section, we have described that the feature vector g in feature space has been mapped from a compound *c* thought a function *φ*, we want to find all possible *c'*=*φ* (*g*). Figure 3 shows that we are interested in inferring compounds which have same feature vector of the target compound. If a compound structure is bigger, the solution space will bigger. It leads to larger computation time to find the answer. In other word, consistent with the feature vector g are also less, it needs more time that mapped back to c' from g, this will be a substantial increase in the amount of time spent.

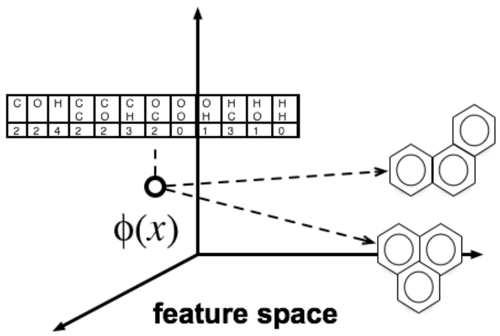


Figure 3: Inferring all possible *c'*=*φ* (*g*) of a graph from a feature vector.

In this study, we develop a parallel algorithm to decrease the computing time and wish to find all possible compound structure having the same feature vector with the target compound. The task will be separated into several parts and appropriate to distribute for computing node.

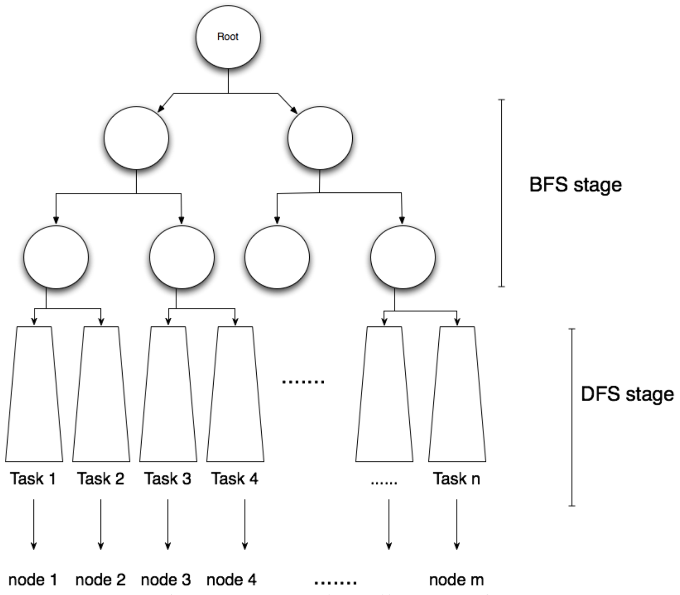


Figure 4: Procedure diagram of PB-CIPF

Figure 4 depicts the idea of our approach. Our proposed method has two stages. Firstly, a master node builds several candidate compounds using BFS. Dispatch candidate compounds to each participated computing nodes according to block distribution. Then each computing node will infer the *c'*=*φ*(*g*) using a DFS approach. The BFS and DFS adopted branch-and-bound approach.

The branch-and-bound approach has 3 stages:

(1) Branching Stage: Insert a new node to selected candidate compound.
(2) Bounding Stage: If addition of a leaf with atom label violates the condition on the numbers of occurrences of atoms.
(3) Terminating Stage: If all candidate compounds are complete the computation.

Each task will spend different computing time. A bigger structure compound means having larger solution space and required more computing time. For example, if we had four computing node C_1, C_2, C_3 and C_4. The master computing node C_1 analyzed the target compound calculating feature vector and established four tasks T_1, T_2, T_3, T_4, and assigned tasks to four computing nodes to execute. It will stop counting after the last task finished.

At BFS stage, the first step, master node loads the target compound and computes the feature vector. The master computing node employs the Breadth-First-Search (BFS) approach to obtain the candidate compounds. Figure 5 shows that each job is initiated based on the atoms that exist in the target compound. However, H atoms are not included. A master node will build several candidate compounds using BFS approach and obtain its path frequency for distributing jobs. Tasks will be putted into a block with appropriate amount then each block will be assigned one by one to computing nodes.

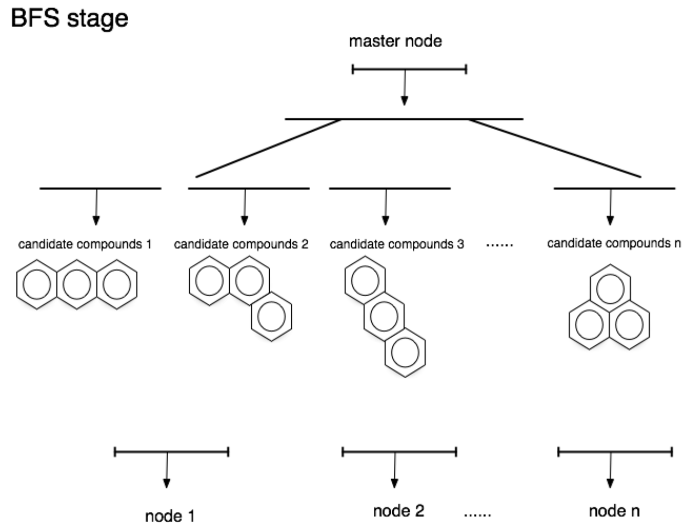
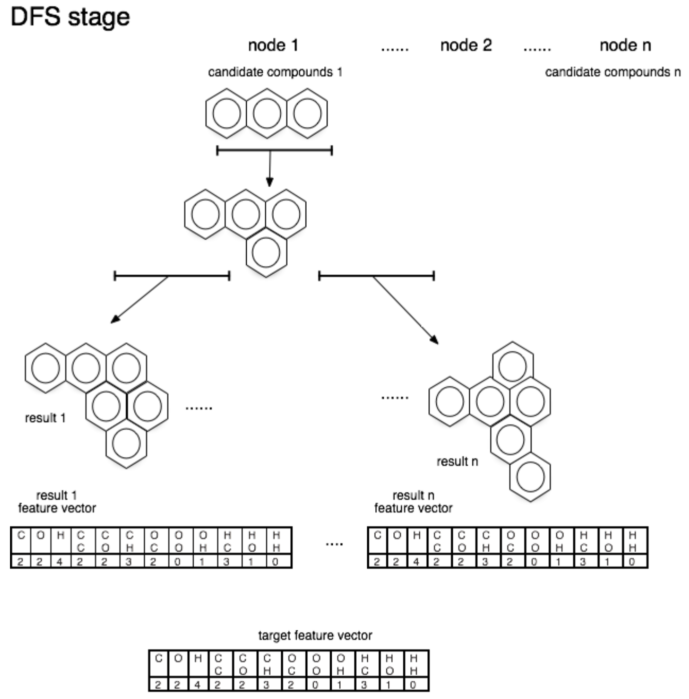


Figure5: An example of BFS stage of PB-CIPF

After the tasks were assigned by the master computing node, each slave computing node used Depth-First-Search (DFS) approach to insert an atom into a candidate compound. After inserted an atom, the candidate compound will be compared feature vector with target compound. If the feature vector of candidate compound and target compound has the same feature vector with parts of target compound structure, then the atom will be kept. If the candidate feature vector is different from target compound structure, then the atom will be dropped and continue to apply DFS approach to insert another atom into the candidate compound. If the candidate compounds have the same feature vector with the target structure, then it got a solution. Do the same thing until all nodes completed its candidate compounds. Figure 6 shows the idea of this stage.



result 1 feature vector

C	O	H	C C	C O	C H	O C	O O	O H	H C	H O	H H
2	2	4	2	2	3	2	0	1	3	1	0

result n feature vector

C	O	H	C C	C O	C H	O C	O O	O H	H C	H O	H H
2	2	4	2	2	3	2	0	1	3	1	0

target feature vector

C	O	H	C C	C O	C H	O C	O O	O H	H C	H O	H H
2	2	4	2	2	3	2	0	1	3	1	0

Figure 6: An example of DFS stage of PB-CIPF

3.1 The procedure of PB-CIPF

The pseudo code of PB-CIPF is shown below; the implemented code has more details and will be presented later.

```
Procedure PB-CIPF($T^{temp}_i$, $f^{temp}_i$, $T^{target}$, $f^{target}$)
  GET processors numbers:n
  Let first computing node = master computing node

  **Master computing node:**
    **step 1:** Produce candidate compounds. Run BFS ($T^{target}$).
            Store all candidate compounds computed in $T^{queue}$
            which is a queue that stores all candidate
            compounds.

            The procedure of BFS is shown below:
            **Procedure BFS** ($T^{target}$):
              **for** (a = all atoms exist in $T^{target}$ ) **do**
                Let $T^{temp}$ be a temporary candidate compound
                Initial $T^{temp}$
                Insert $a$ as a new node into $T^{temp}$
                Add $T^{temp}$ to $T^{queue}$
                Process the next atom in $T^{target}$
              **end**
    **step 2:** Gather candidate compounds as tasks. Block tasks
            by computing nodes number.
    **step 3:** For each block of tasks will be assigned to slave
            computing node.


  **Slave computing node:**
    **step 1:** Receive tasks.
    **step 2**: Run DFS approach.
            The procedure of DFS is shown below:
             **Procedure DFS**($T^{temp}_i$, $f^{temp}_i$, $T^{target}$, $f^{target}$)
              **if** $f^{temp}_i = f^{target}$
                **then** output a solution $T^{temp}_i$;
              **return** true;
              **else return** false;
              **for** ($a$ = all atoms exist in $T^{target}$ )**do**
                **if** atom = H **continue;**
                (Hydrogen atoms will be added at the last
              stage**)**
                **if** $\{l(u)|u\ V(T^{temp}) \cup \{a\}\ atomset(f^{erget})$
                *(set means multiset here)*
                  **then continue;**
                **for** all $w \in V(T^{temp})$ **do**
                  Let $T^{next}$ be a tree got by connecting new
                leaf node $u$
                  **if** $w$ does not satisfy the valence constraint
                    **then continue**;
                    Compute $f^{next}$ from $T^{next}$ and $f^{temp}$;
                    **if** DFS($T^{next}$, $f^{next}$, $T^{target}$, $f^{target}$)=true
                      **then return** true;
                **end**
              **end**
              **return false**;
  **step 3:** Send result to a queue stored all matched result.
  **step 4:** If still needed to process go to step 2. Else end.
```

The followings are the detail part:
(i) Hydrogen atoms will be added at the last stage of the inferring procedure. Hydrogen atoms will be added only if the frequencies of the other atoms are same as those in the target feature vector.
(ii) When calculating $\boldsymbol{f}^{next}$ from $\boldsymbol{T}^{next}$ and $\boldsymbol{f}^{cur}$, paths beginning from and ending at a new node are only computed.
(iii) Benzene rings can be added as if these were leaves, where structural information on benzene is utilized for calculating feature vectors.
(iv) A benzene ring will be given as an initial structure when a compound is small and contains a benzene ring.

4. Experimental results

In this section, experimental results of our algorithm were shown in term of the comparisons of single node, 2 nodes, and 4 nodes.

We used a PC cluster with AMD Athlon(TM) XP 2000+ CPU and 1 GB Ram which worked on the Linux to verify the performance of our algorithm. PB-CIPF was implemented using C language and the MPI version with MPICH2. The test data were obtained from KEGG LIGAND Compound Database.

We increase the computing node to verify that increasing computing node can reduce the computing time.

The makespan defined as following:

In our experiment, it has 4 computing nodes: C_0, C_1, C_2 and C_3. Then the finish time of four nodes is: t_0, t_1, t_2, t_3. If t_2 is the longest finish time, then the makespan will be t_2.

We choose several compounds form KEGG LIGAND Database. We tested and verified our algorithm with K = 1, 2, 3, 4, where K is the length of sequence labels of feature vectors. Bigger value of K means more constrains for target compound. Figure 7 to 10 show the makespans from different target compound with different size. Compound size of C11108 with H atom is 13 and 9 without H atom. Size of C11109 with H atom is 14 and 7 without H atom. Size of C00097 with H atom is 15 and 9 without H atom. Size of C15987 with H atom is 19 and 8 without H atom. The less value of K makes bigger amount of permutation with small length of path frequency. The more amount of permutation with path frequency will make more results which match the path frequency then cost more computing time. Therefore, figure 11 shows that when computing C15987 with K=1, the computing time is much bigger than it of other value of K, so makespan of C15987 in figure 10 is excluded the result of K =1.We can find that the makespan was reduced when the number of nodes increased. For example, in Figure7, when the value of K is 4, the makespan was reduced from 399.546 seconds to 279.82 second by 4 nodes.

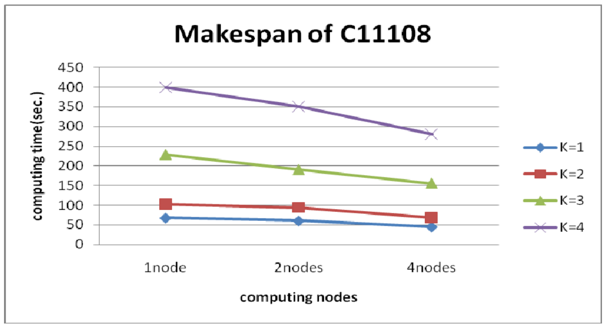


Figure 7: Makespan of C11108

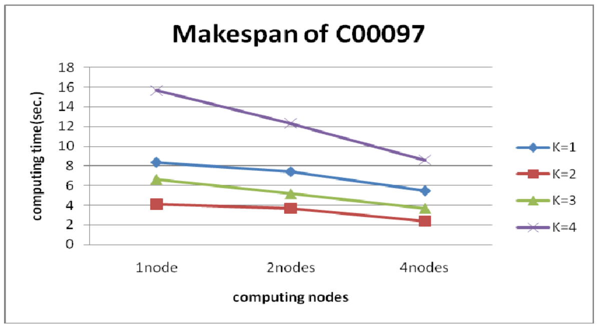


Figure8: Makespan of C00097

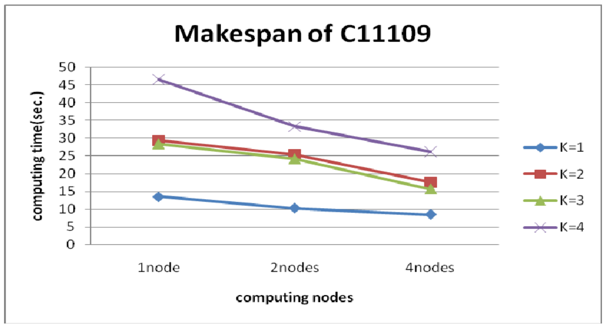


Figure9: Makespan of C11109.

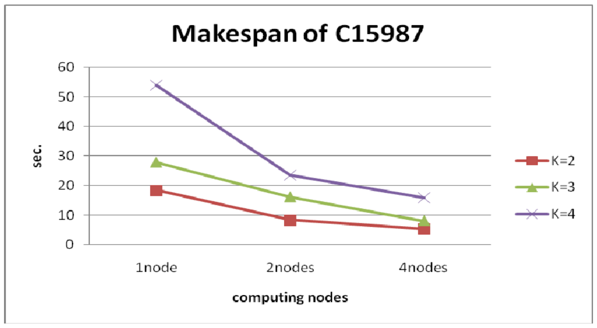


Figure 10: Makespan of C15987

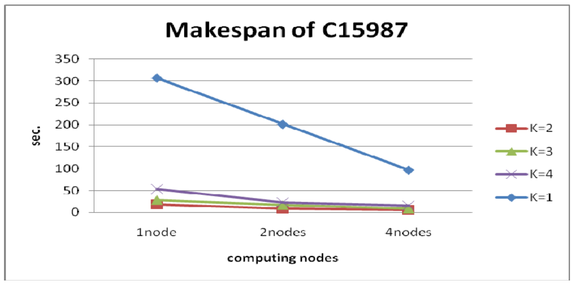


Figure 11: Makespan of C15987 with K=1

When the K = 1 and size of atoms is bigger than 19, the solution space is larger, so it needs more computing time to finish. We use several compounds with different size of atoms to verify the performance of our algorithm. To look over performance of our algorithm, we compute the speedup ratio of each testing case. The speedup ratio is defined as follows:

If the computing time of single node is t_0 and the computing time of 2 nodes is t_1. Then the speedup ratio of 2 computing nodes will be t_0 / t_1.

Figure12 and 13 show the speedup of our algorithm. Increasing the computing nodes to 4 nodes, we can find out that the average speedup ratio is about 1.9. According to our experiment, it verified that our proposed algorithm can reduce the computing time.

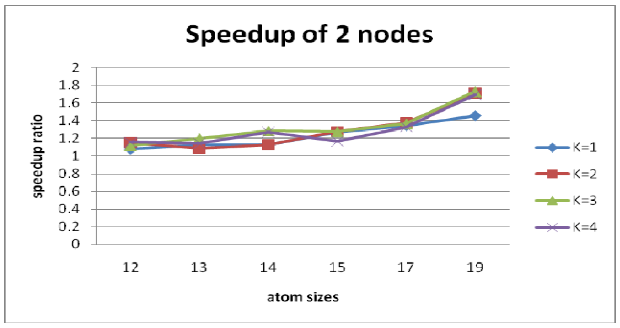


Figure 12: Speedup ratio of 2 nodes

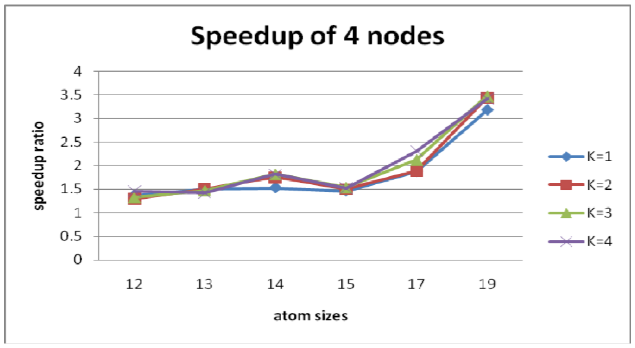


Figure 13: Speedup ratio of 4 nodes

5. Conclusion

In this paper, we proposed a parallel algorithm for the problem of chemical compound inference from path frequency. Our approach has two stages. First, a master node will build several candidate compounds using BFS approach. Then distribute the candidate compounds to participated computing nodes according to block distribution. Then each computing node will infer the $c'=\varphi\ (g)$ using a DFS approach. The BFS and DFS adopted branch-and-bound approach. The experimental results show that our algorithm can reduce the computing time. When using 4 nodes to compute, the average speedup is 1.820136 when K=1, 1.891199 when K=2, 1.954588 when K=3 and 1.995495 when K=4.

References

[1] C. J. Harris, A. P. Stevens, "Chemogenomics: structuring the drug discovery process to gene families", Drug Discov Today11, 2006, 880-888.

[2] C. Cortes, V. Vapnik, "Support vector networks". Machine Learning 20, 2005, 273-297.

[3] N. Cristianini, J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*, Cambridge Univ. Press, 2000.

[4] E. Byvatov, U. Fechner, J. Sadowski, and G. Schneider, "Comparison of support vector machine and artificial neural network systems for drug/nondrug classification" *Journal of Chemical Information and Computer Sciences 43*, 2003, 1882–1889.

[5] M. Deshpande, M. Kuramochi, N. Wale, and G. Karypis, " Frequent substructure-based approaches for classifying chemical compounds", *IEEE Trans. Knowledge and Data Engineering17*, 2005, 1036–1050.

[6] A. Ben-Hur, W. Noble, "Kernel methods for predicting protein-protein interactions," *Bioinformatics, vol. 21*, 2005, 38-46.

[7] C. J. C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition" *Data Mining and Knowledge Discovery, vol. 2*, 1998, 121-167.

[8] H. Kashima, K. Tsuda, and A. Inokuchi, " Marginalized kernels between labeled graphs." *Machine Learning 20*, 2005, 321–328.

[9] P. Mahé, N. Ueda, T. Akutsu, J.-L. Perret, and J.-P. Vert, " Graph kernels for molecular structure-activity relationship analysis with support vector machines", *Journal of Chemical Information and Modeling 45,* 2005, 939–951.

[10] T. Akutsu, D. Fukagawa, "Inferring a graph from path frequency", *Combinatorial Pattern Matching 16, Volume 2527 of Lecture Notes in Computer Science*, 2005, 371-392.

[11] T. Akutsu, D. Fukagawa, "On inference of a chemical structure from path frequency", *BIOINFO*, 2005.

[12] T. Akutsu, D. Fukagawa," Inferring a Chemical Structure from a Feature Vector Based on Frequency of Labeled Paths and Small Fragments", *Asia-Pacific bioinformatics conference 5th*, 2007, 165-174.

[13] D. B. Skillicorn, D. Talia, " Models and Languages for Parallel Computation", *ACM Computing Surveys Vol. 30 No. 2*, 1998, 123-169.

[14] O. Maruyama, S. Miyano, "Inferring a tree from walks" *Theoretical Computer Science 161*, 1996, 289–300.

[15] J. Lauri, R. Scapellato, *Topics in Graph Automorphisms and Reconstruction*, Cambridge Univ. Press, 2003.

[16] J. J. Vincent, K. M. M. Jr, "A highly portable parallel implementation of AMBER4 using the message passing interface standard", *Journal of Computational Chemistry Volume 16 Issue 11*, 1995, 1420-1427.

[17] W. Gropp, E. Lusk, N. Doss, and A. Skjellum , "A high-performance, portable implementation of the MPI message passing interface standard" , *Parallel Computing 22*, 1996, 789-828.

[18] G. H. Bakimathr, A. Zien, and K. Tsuda, "Learning to find graph pre-images ", *The 26th DAGM Symposium, Volume 3175 of Lecture Notes in Computer Science*, 2004, 253-261.

Chemical Compounds with Path Frequency Using Multi-Core Technology

Kun-Ming Yu[1], Yi-Yan Chang[2], Jiayi Zhou[3], Chun-Yuan Huang[1], Whei-meih Chang[1], Chun-Yuan Lin[4], Chuan Yi Tang[5]

[1]Department of Bioinformatics, Chung Hua University
[2]Department of Information Management, Chung Hua University
[3]Institute of Engineering and Science, Chung Hua University
[4]Department of Computer Science and Information Engineering, Chang Gung University
[5]Department of Computer Science, National Tsing Hua University

yu@chu.edu.tw, {frank38, jyzhou}@pdlab.csie.chu.edu.tw, {chunyuan.huang, wmchang}@chu.edu.tw, cyulin@mail.cgu.edu.tw, cytang@cs.nthu.edu.tw

Abstract. Drug design is the approach of finding drugs by design using computational tools. When designing a new drug, the structure of the drug molecule can be modeled by classification of potential chemical compounds. Kernel Methods have been successfully used in classifying chemical compounds, within which the most popular one is Support Vector Machine (SVM). In order to classify the characteristics of chemical compounds, methods such as frequency of labeled paths have been proposed to map compounds into feature vectors. In this study, we analyze the path frequencies computed from chemical compounds, and reconstruct all possible compounds that share the same path frequency with the original ones, but differ in their molecular structures. Since the computation time for reconstructing such compounds increase greatly along with the size increase of the compounds, we propose an efficient algorithm based on multi-core processing technology. We report here that our algorithm can infer chemical compounds from path frequency while effectively reduce computation time and obtained high speed up.

Keywords: Chemical compound, feature space, Multi-Core Processing, Branch-and-Bound, OpenMP

1 Introduction

In recent years, many researchers have worked on the drug design problem in order to develop new drugs based on computation methods. When designing a new drug, the structure of the drug molecule can be modeled by classifying candidate chemical compounds using Kernel Methods [4, 5, 6, 7], within which the most popular one is Support Vector Machine (SVM) [10]. Kernel method is a type of pattern analysis, the task of which is to discover the relationships, such as clusters, rankings, classifications, in the data (such as sequences, vectors, sets of points, images, etc). Kernel methods approach the problem by first mapping the data into a highdimensional feature space. Recently, it has also been applied to the classification of chemical compounds [4, 5, 6, 7]. In these approaches, chemical compounds are mapped to feature vectors and then SVMs [9, 10] are employed to learn the rules for classifying these feature vectors. Several mapping methods for feature vectors have been proposed; among them, the mapping of feature vectors based on the *frequency of labeled paths* [6, 7] or the *frequency of small fragments in chemical compounds* [4, 5] are widely used.

In kernel methods, an object in the input space can be mapped into a point (or feature vector) in a space called feature space. Through a suitable function $\emptyset$, a given point y in the feature space can be mapped back into an object in the input space. Such object is called pre-image. The problem exists when mapping a given y in feature space back into an object in the input space such that $y=\emptyset(x)$ is satisfied, as x may not exist.

In [1], a feature vector g is a multiple set of strings of labels with length at most K which represents path frequency. Given a feature vector g, they considered the problem of finding a vertex-labeled graph G that attains a one-to-one correspondence between g and the set of sequences of labels along all paths of length at most K in G.

In previous works [1, 2], a graph can be inferred from the numbers of occurrences of vertex-labeled paths. In [1], they showed that this problem can be solved in polynomial time of the size of an output graph if graphs are trees of bounded degree and the lengths of given paths are bounded, by a constant, whereas this problem is strongly NP-hard even for planar graphs of bounded degree.

In this study, we have taken into account the situation when chemical compounds become increasingly complex, the computation time required to infer pre-images from the feature vectors of these compounds increase at a much faster rate. We resort to parallel computing, in which the computation tasks are assigned to multiple cores appropriately to reduce the overall computation time. We extend the algorithms in [3], and therefore the modified algorithms can support multi-core processing technology.

The rest of this paper is organized as follows. Section 2 introduces the background about problem and definition. Next we describe our proposed algorithms in section 3. In section 4, we show the experimental result. Finally we conclude this paper in section 5.

2 Related Work

For classification of the characteristics of chemical compounds to work, chemical compounds are often mapped into feature vectors. Several methods for converting chemical compounds into feature vectors have been proposed. Among them, methods such as frequency of labeled paths [6, 7] or frequency of small fragments [4, 5] are popular. Recently, the pre-image methods have been proposed. In [4], pre-images were found in a general setting by using Kernel Principal Component Analysis and regression. In [8], stochastic search algorithm is used to find pre-images for graphs. However, these pre-image methods are not derived from a computational viewpoint.

In [4], the obtained results and performance of the algorithm was unclear because it was applied only to a few similar cases. Other related pre-image studies include inferring a tree from walks in [12], as well as inferring by graphic reconstruction [13].

In [3], chemical structures are modeled as trees or tree-like structures. They extend algorithms in [1, 2] so that constraints on valences of atoms are taken into account. They proposed an algorithm, *Branch-and-Bound Chemical compound Inference from Path Frequency* (BB-CIPF), which can infer tree from related chemical structures. BB-CIPF works within a few or a few tens of seconds for inferring moderate size of chemical compounds (e.g., the number of carbon atoms are less than 20) with tree or tree-like structures, and can be modified for inferring more general classes of chemical compounds and/or for feature vectors based on frequency of small fragments.

In BB-CIPF, given a tree T^{cur} to be inferred to a target tree T^{target}, T^{cur} is first inserted into a node n to become T^{next}. If the feature vector f^{next} of T^{next} does not comply with the feature vector f^{target} of T^{target}, the T^{next} will be discarded and then the T^{cur} will be re-inserted into another node and be compared to T^{target}.

The advantage of BB-CIPF algorithm is to effectively reduce the computation time, as it terminates the computation process immediately and displays the results once it obtains a solution; this also means that there is only one solution [3]. For example, if there are three objects, a, b and c, which all correspond to the same feature vectors v. Through BB-CIPF algorithm, only one of the objects a, b, c can be inferred from v, so the inferred solution is not necessarily be the most useful one in practice. Therefore, how to produce all possible compounds that are mapped back from the same feature vector but differ in their molecular structures is an important issue in the problem. Moreover, when a compound structure is more complex, it will require more computation time for inference of its solutions.

Parallel computing is a suitable technique in shortening the inference procedure. Parallel computing is a form of computation in which several calculations are carried out simultaneously [11], operated on the principle that large problems can often be divided into smaller ones, and then solved concurrently to provide the solution in a shorter time. While clusters, Massive parallel processing (MPP), and Grids use multiple computers to work on the same task, multi-core and multi-processor computers employ multiple processing elements to work on the same task.

A multi-core processor (or chip-level multiprocessor) combines two or more independent cores (normally a CPU) into a single package that consisted of a single integrated circuit. A dual-core processor contains two cores, and a quad-core processor contains four cores. A multi-core microprocessor implements several processing units in a single physical package. In general, programming is required to orchestrate processes in several cores in order to solve problems.

The OpenMP (Open Multi-Processing) standard allows programmers to take advantage of the new shared-memory, multiprocessor programming systems from vendors like Compaq, Sun, HP, and SGI. Aimed at the researcher working with C/C++ or Fortran programming languages, OpenMP explains both what this standard is and how to use it to create software that takes full advantage of parallel computing. OpenMP support Sun compiler, GNU compiler and Intel compiler.

In this paper, we extend the inference algorithm [3] to obtaining all possible compounds that are mapped back from the same feature vector but differ in their molecular structures. We used the Branch-and-Bound concept to derive the trees or tree-like structures of chemical compounds. Our algorithm is committed to obtain all possible compounds that can be inferred from the same feature vector but differ in their molecular structures. We develop our algorithm based upon the algorithm in [3] so that the computation process will not terminate on the first obtained solution, but will continue to search for all possible solutions. However, in order to output more chemical compounds, it also means that the algorithm will consume more computation time. Therefore, we also propose adopting the multi-core computing technology to reduce the computation time in our proposed algorithm. We hope that by providing more thorough and practical solutions to the inference problem, we can improve on the development of drug design.

3 Multi-Core Chemical Compound Inference from Path Frequency (MC-CIPF)

In the previous section, we have described that when a compound structure is more complex, it will require more computation time for inference of its solutions. That is to say, if the feature vector v in feature space has been mapped from a compound *c* thought a function ∅, and we want to find *c'* where *c'*= ∅(*v*). If a compound is more complex in structure, its feature vector in feature space is also more complex, and it will require substantially more computation time to map back to *c'* from *v*. Therefore, in this paper, we divide computation tasks into several smaller tasks and distribute these tasks appropriately among several processing cores for computation. We propose the Multi-Core Chemical Compound Inference from Path Frequency (MC-CIPF) to obtain all possible compounds.

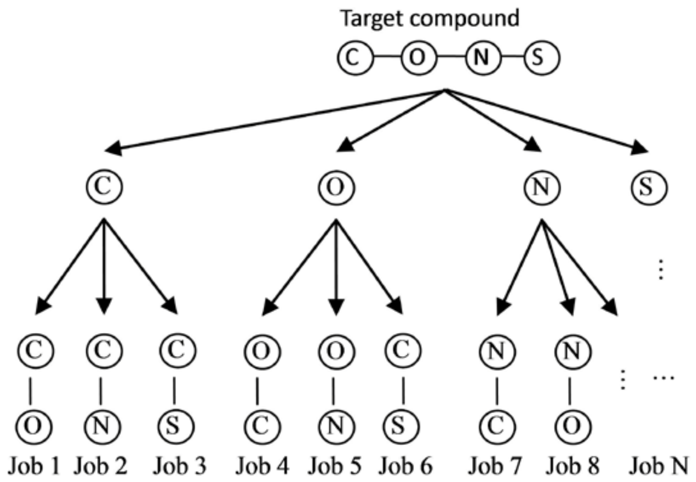


Fig. 1. Each job is initiated based on the atoms that existed in the target compound.

In the first step of MC-CIPF, the algorithm loads into the master core a target compound for inference of all other chemical compounds that share the same feature vector. The master core employs the Breadth-First-Search (BFS) algorithm to analyze the target compound and obtain its path frequency for distributing jobs later. Each job is initiated based on the atoms that exist in the target compound (Fig. 1). However, H atoms are not included in this step.

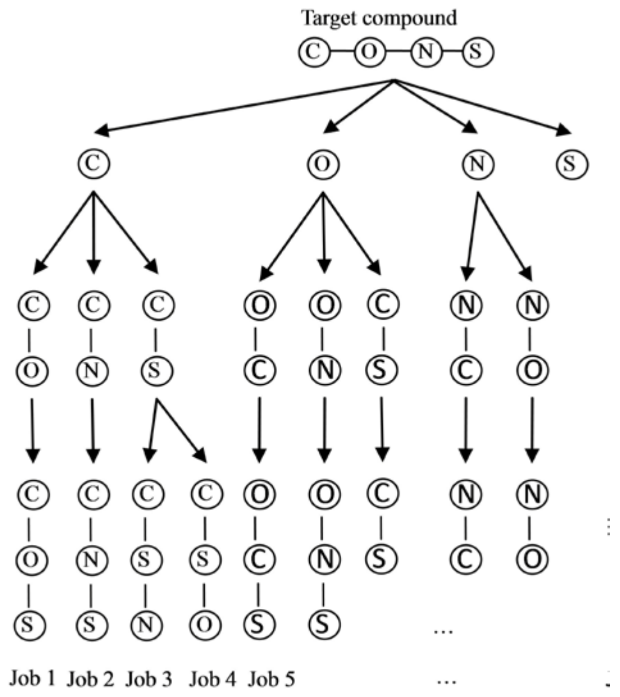


Fig. 2. An example of balancing the load in each core in MC-CIPF.

Each job requires different amount of time for computation, and a more complex one will require more time. For example, if there are four cores C_1, C_2, C_3 and C_4, the master core will analyze the target compound, initiate four jobs T_1, T_2, T_3, T_4, and distribute them among four corresponding cores for execution. If T_1, T_2 and T_3 have completed their jobs while T_4 is still in process, T_1, T_2 and T_3 cores will be in idle as there are no more jobs to allocate to these cores. Therefore, we balance the overall computation loads by increasing the number of jobs that can be allocated by the master core and reducing the computation time demanded for each job. The scenario is depicted in Fig. 2.

Each core applies the Depth-First-Search (DFS) algorithm to insert an atom into a candidate compound. After inserted an atom, the candidate compound will be compared with target compound, and if the feature vector of candidate compound has the same structure as parts of target compound structure, the inserted atom will be kept; otherwise, the inserted atom will be dropped, and the algorithm continues on applying DFS to insert the next atom in queue into the candidate compound. If the resulted structure of the candidate compounds is in line with the target structure, it is output as one of the solution. The algorithm iterates until all cores have completed all candidate compounds in their queues.

```
Procedure BFS ($T^{targe}$)
  Let $T^{queue}$ be a queue that stores all candidate compounds;
  for all $a \in$ all atoms exist in $T^{target}$ do
    Let $T^{temp}$ be a temporary compound
    $T^{temp} \leftarrow \varnothing$ ;
    Insert $a$ into $T^{temp}$;
    if $T^{temp} \in T^{target}$ then
      Add $T^{temp}$ to $T^{queue}$;
    else
      continue (examine the next atom in $T^{target}$);
    end
  end
  while $T^{queue}$ is not empty do
    for each core compute a compound from $T^{queue}$ per time do
      Compute feature vector $f^{temp}{}_{i}$ from $T^{temp}{}_{i}$ in $T^{queue}$;
      if MC-CIPF($T^{temp}{}_{i}$, $f^{temp}{}_{i}$, $T^{target}$, $f^{target}$)=false then
        output "no solution";
      end
    end

Procedure MC-CIPF($T^{temp}{}_{i}$, $f^{temp}{}_{i}$, $T^{target}$, $f^{target}$)
  if $f^{temp}{}_{i} = f^{target}$ then output $T^{temp}{}_{i}$;
    popup $T^{temp}{}_{i}$ from $T^{temp}$;
    return true;
  else
    return false;
  for all $a \in$ all atoms exist in $T^{target}$ do
    $L \leftarrow \varnothing$ ;
    if $\{ L(u) | u \in V(T^{temp})\} \cup \{a\} \not\subseteq atomset(f^{target})$ then
      continue;
    for all $w \in V(T^{temp})$ do
      Let $T^{next}$ be a tree gotten by connecting new leaf $u$ with
label $a$ to $w$ by bond $b$;
      if $w$ does not satisfy the valence constraint then
        continue;
      Compute $f^{next}$ from $T^{next}$ and $f^{temp}$;
      if MC-CIPF($T^{next}$, $f^{next}$, $T^{target}$, $f^{target}$)=true then
        return true;
      end
    end
  return false;
```

4 Experimental Results

For verifying the effectiveness of MC-CIPF, we implemented the proposed algorithm and compared the performance when using single core, dual core and quad core for computation. The simulation environment is built by using a personal computer equipped with Intel Core 2 Quad Q6600 CPU and 4 GB RAM and installed with the operating system of Windows Vista with Service Pack 1. MC-CIPF was implemented using C language and experimental datasets are retrieved from KEGG LIGAND Database.

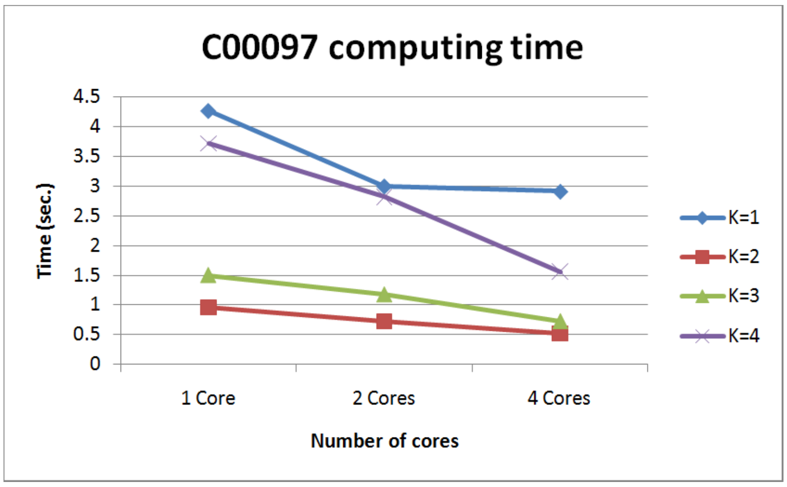


Fig. 3(a). Computing time of C00097.

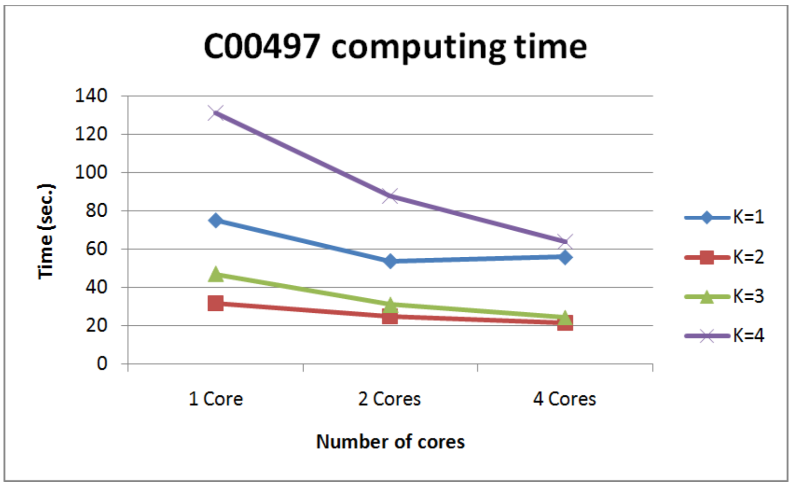


Fig. 3(b). Computing time of C00497.

In the experiment, we randomly chosen 5 chemical compounds (C00097, C00497, C11109, C14601, and C15987; the number of atoms of compound size with hydrogen are 14, 15, 16, 15 and 19, respectively) from KEGG LIGAND Database and examined them with $K = 1, 2, 3, 4$, where K is the length of sequence label of feature vectors in MC-CIPF. Larger K means more constraints for target compound, which leads to less variation for its molecular structure. Fig. 3(a)-(e) are the computing time for each chemical compound. In each case, we have found that the computing time was reduced as the number of cores was increased. For example, in Fig. 3(c), when K was equal to 4, the computing time was reduced from 11.9337 second with 1 core to 5.542821 second with 4 cores.

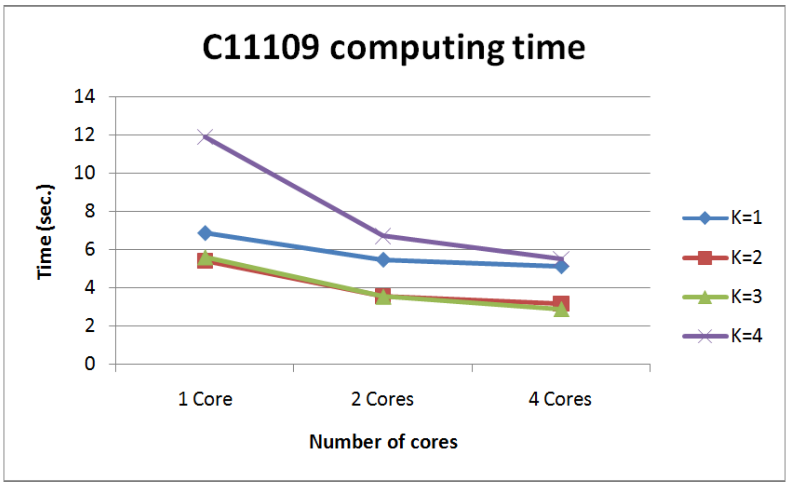


Fig. 3(c). Computing time of C11109.

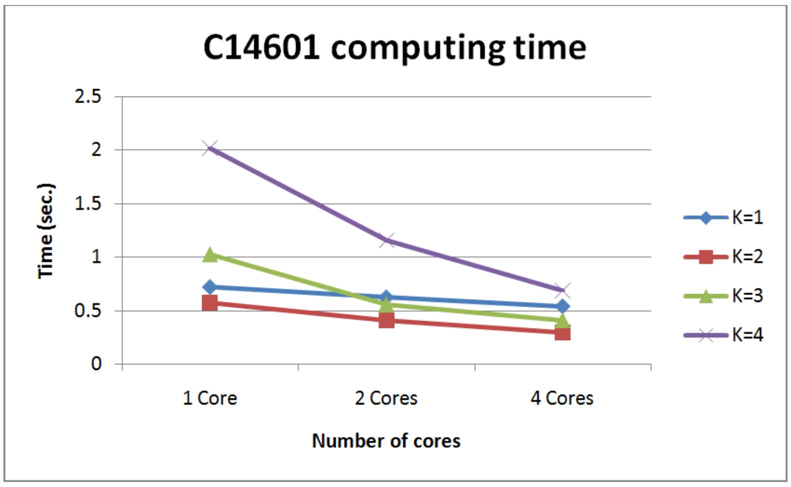


Fig. 3(d). Computing time of C14601.

When the *K* value increases, the constraints of the feature vectors increase accordingly. As a result, MC-CIPF spends less computing time in searching for the combinations of target compound, as there is less permitted variations to compute for. However, the path frequency will be longer, so MC-CIPF needs to spend more execution time in re-computing path frequency. Consequently, the shortest computing time occurs when *K* is equal to 2 in the experiment, since the number of constraints has not increased too much and the length of path frequency is not too long. Details of the computing time are shown in Table 1.

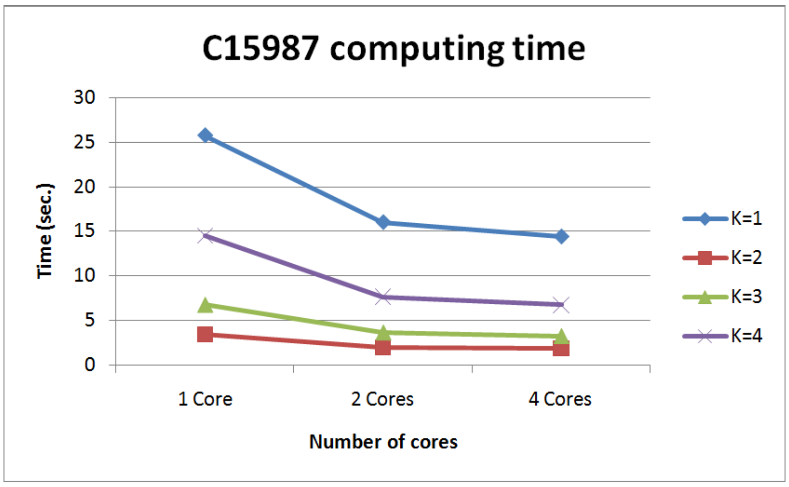


Fig. 3(e). Computing time of C15987.

Table 1. Computing time of MC-CIPF for various chemical compounds.

Instances	Cores detail		CPU time (sec.)			
			K=1	K=2	K=3	K=4
C00097	1 Core		4.27	0.96	1.49	3.71
	2 Cores	Core 1	2.99	0.26	1.17	2.81
		Core 2	3.00	0.72	1.17	2.81
		Max	3.00	0.72	1.17	2.81
	4 Cores	Core 1	2.91	0.20	0.72	1.56
		Core 2	2.67	0.36	0.46	0.98
		Core 3	0.77	0.35	0.50	0.97
		Core 4	2.91	0.51	0.72	1.56
		Max	2.91	0.51	0.72	1.56
C00497	1 Core		75.44	31.95	47.23	131.42
	2 Cores	Core 1	28.82	8.78	31.26	35.33
		Core 2	54.10	25.12	31.27	88.09
		Max	54.10	25.12	31.27	88.09
	4 Cores	Core 1	13.76	3.93	24.43	11.05
		Core 2	31.23	7.77	12.25	28.88
		Core 3	12.47	7.88	12.26	30.69
		Core 4	56.14	21.67	24.43	64.10
		Max	56.14	21.67	24.43	64.10
C11109	1 Core		6.88	5.42	5.63	11.93
	2 Cores	Core 1	5.47	3.59	3.55	6.73
		Core 2	5.48	3.59	3.56	6.73
		Max	5.48	3.59	3.561	6.73
	4 Cores	Core 1	5.13	3.18	2.90	5.53
		Core 2	1.83	2.06	1.91	3.53
		Core 3	2.48	2.05	1.18	1.93
		Core 4	5.13	3.18	2.90	5.54
		Max	5.13	3.18	2.90	5.54
C14601	1 Core		0.72	0.57	1.02	2.02
	2 Cores	Core 1	0.62	0.40	0.51	0.83
		Core 2	0.62	0.40	0.55	1.16
		Total	0.62	0.40	0.55	1.16
	4 Cores	Core 1	0.54	0.27	0.40	0.37
		Core 2	0.13	0.23	0.38	0.55
		Core 3	0.21	0.25	0.40	0.68

		Core 4	0.54	0.29	0.35	0.58
		Max	0.54	0.29	0.40	0.68
C15987	1 Core		25.73	3.49	6.77	14.52
	2 Cores	Core 1	16.02	2.00	0.35	4.95
		Core 2	11.43	2.00	3.67	7.67
		Max	16.02	2.00	3.67	7.67
	4 Cores	Core 1	14.41	1.87	1.28	2.03
		Core 2	5.78	1.48	1.65	3.16
		Core 3	12.85	1.87	3.25	6.76
		Core 4	1.42	0.72	0.55	0.68
		Max	14.41	1.87	3.25	6.76

More importantly, we want to compare the speedup ratios of MC-CIPF with respect to the core number used in the experiments. Fig. 4(a)-(e) are the speedup ratios of C00097, C00497, C11109, C14601, and C15987. In these figures, the speedup ratios are increased from 1 core to 4 cores, with the best speedup ratio close to 3 (Fig. 4(c)). Interestingly, when the *K* value is increased, the speedup ratio is raised accordingly.

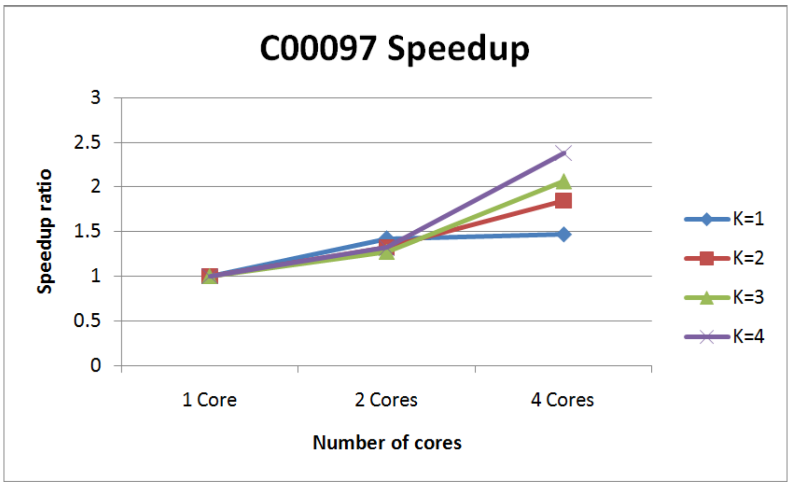


Fig. 5(a). Speedup ratio of C00097

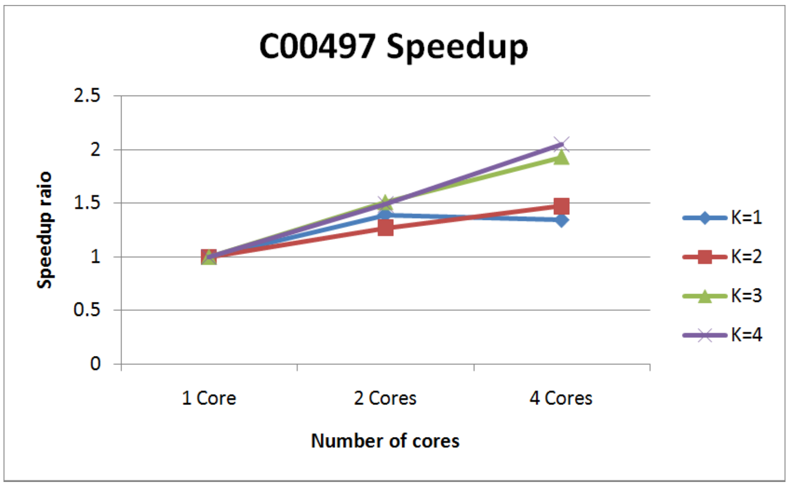


Fig. 4(b). Speedup ratio of C00497.

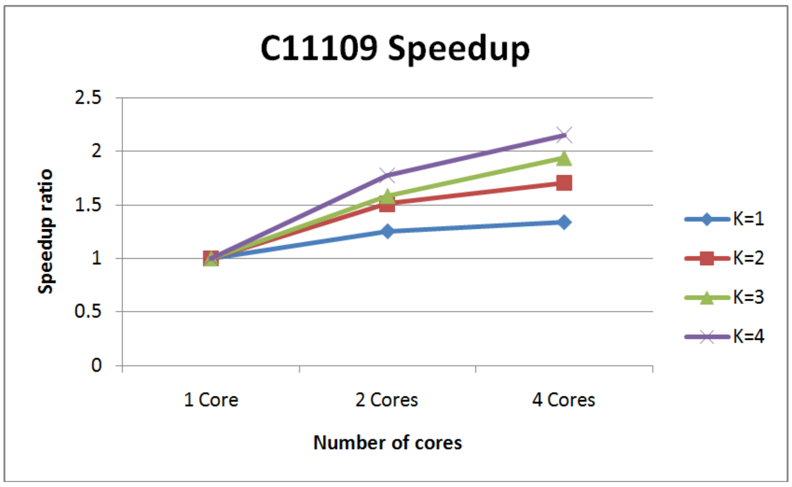


Fig. 4(c). Speedup ratio of C11109.

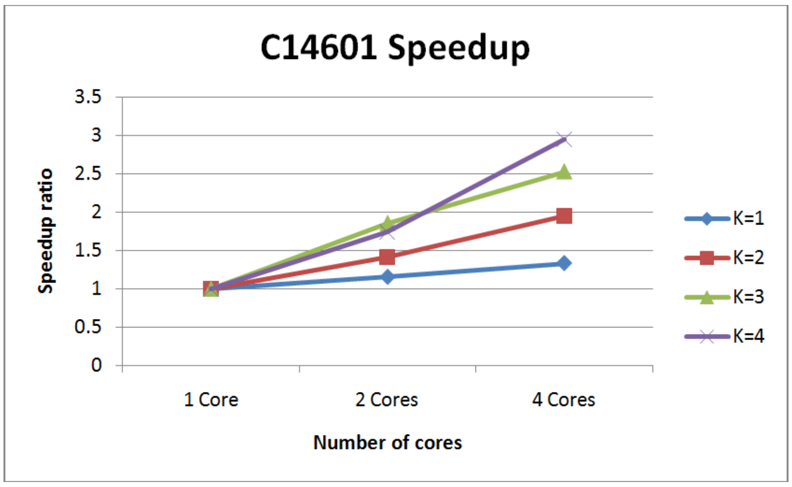


Fig. 4(d). Speedup ratio of C14601.

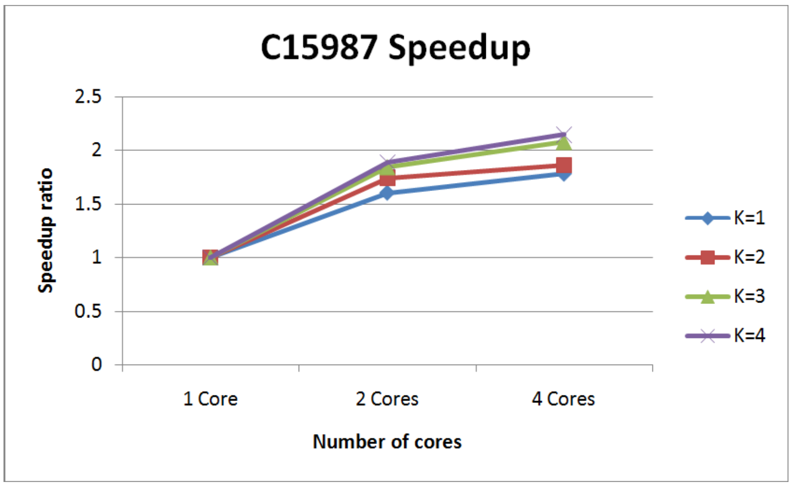


Fig. 4(e). Speedup ratio of C15987.

5 Conclusions

In this research, we proposed a multi-core algorithm for solving Chemical Compound Inference from Path Frequency problem. We adopted the Branch-and-Bound concept to evolve the tree-like structures of chemical compounds in the paper. The experimental results show that our algorithm can practically reduce computing time, with the best speedup ratio close to 3 folds while using 4 cores in the experiment. Therefore, our proposed algorithm can infer chemical compounds from path frequency effectively and reduce computation time by employing the multi-core technology.

References

1. Tatsuya, A., Daiji, F.: Inferring a graph from path frequency, In Proc. 16th Symp. Combinatorial Pattern Matching. Lecture Notes in Computer Science, Springer. 2527, 371--392 (2005)
2. Tatsuya, A., Daiji, F.: On inference of a chemical structure from path frequency. Proc. 2005 International Joint Conference of InCoB, AASBi, and KSBI, pp. 96--100 (2005).
3. Tatsuya, A., Daiji, F.: Inferring a Chemical Structure from a Feature Vector Based on Frequency of Labeled Paths and Small Fragments. APBC, pp. 165--174 (2007)
4. Byvatov, E., Fechner, U., Sadowski, J., Schneider, G.: Comparison of support vector machine and artficial neural network systems for drug/nondrug classification. Journal of Chemical Information and Computer Sciences. 43, 1882--1889 (2003)
5. Deshpande, M., Kuramochi, M., Wale, N., Karypis, G.: Frequent substructure-based approaches for classifying chemical compounds. IEEE Trans. Knowledge and Data Engineering. 17, 1036--1050 (2005)
6. Kashima, H., Tsuda, K., Inokuchi, A.: Marginalized kernels between labeled graphs. In Proc. 20th Int. Conf. Machine Learning, pp. 321--328 (2003)
7. Mahé, P., Ueda, N., Tatsuya, A., Perret, J-L., Vert, J-P.: Graph kernels for molecular structure-activity relationship analysis with support vector machines. Journal of Chemical Information and Modeling. 45, 939--951 (2005)
8. Bakir, G.H., Zien, A., Tsuda, K.: Learning to find graph pre-images. In Proc. The 26th DAGM Symposium. Lecture Notes in Computer Science, Springer. 3175, 253--261 (2004)
9. Cortes, C., Vapnik, V.: Support vector networks. Machine Learning. 20, 273--297 (1995)
10. Cristianini, N., Shawe-Taylor, J.: An Introduction to Support Vector Machines and Other Kernel-based Learning Methods. Cambridge Univ. Press (2000)
11. Almasi, G.S. and A. Gottlieb. Highly Parallel Computing. Benjamin-Cummings publishers, Redwood City, CA. (1989)
12. Maruyama, O., Miyano, S.: Inferring a tree from walks. Theoretical Computer Science. 161, 289--300 (1996)
13. Lauri, J., Scapellato, R.: Topics in Graph Automorphisms and Reconstruction. Cambridge Univ. Press (2003)

行政院國家科學委員會補助國內專家學者出席國際學術會議報告

97 年 9 月 10 日

報告人姓名	游坤明	服務機構及職稱	中華大學 資工系副教授
時間 會議地點	97/8/26 –97/8/28 Hangzhou, China	本會核定 補助文號	
會議 名稱	(中文) 2008 IEEE 粒度計算國際研討會 (英文) 2008 IEEE International Conference on Granular Computing		
發表 論文 題目	(中文) 一個運用權重負載平衡技術進行平行化 Apriori 關連式法則探勘 (英文) A Weighted Load-Balancing Parallel Apriori Algorithm for Association Rule Mining		

一、　參加會議經過

會議的開幕典禮由大會主席致簡單的歡迎詞，並由會議的委員會主席說明本屆的投稿與錄取篇數及本屆會議的特色與會議重點後，隨即展開，本屆會議分別於第一天及第二天的會議議程中各安排了五、六個場次之粒度計算與生物資料處理暨應用領域之最新趨勢之專題報告: (1). Intelligence and Language How Could Human Being Have Language?, (2). What is Granular Computing?: Highlights and Fallacies, (3). Security and Integrity in Outsourcing of Data Mining, (4). Cloud Computing, (5). Cloud Computing Panel (6). Challenges Techniques for Mining Real Clonical Data, (7). Granular Computing:Based on Fuzzy and Tolerance Relation, (8). Granular Computing: Toward inner logic in decision systems, (9). Mechanism Approach to the Research of Atificial Intelligence, (10). Nonstructure Information Retrieval Tolerant Granular Space Model, 以及(11). Clustering Algorithms with Automatic Selection of Cluster Number，分別由 Setsuo Ohsuga、Tsau Young Lin、 David Wai-lok. Cheung、Dennis Quan、 Meng Ye、 Wesley W. Chu、Bo Zhang and Ling Zhang、 Lech Polkowski、Yixin Zhong、ZhongZhi Shi 以及 Michel Ng 作精彩的專題報告。正式之論文報告分別於專題報告後進行，本人之論文『A Weighted Load-Balancing Parallel Apriori Algorithm for Association Rule Mining』被安排在第二天的 –"Intelligent Data Analysis and Applications"之場次發表。

二、　與會心得

GrC 2008 是一個在粒度計算(Granular computing)研究領域中具有指標性的最重要國際會議，此次會議共有近四百篇的論文投稿，但僅錄取了一百七十餘篇優秀的粒度計算資訊系統領域的論文，由會議的進行過程中可以看出主辦單位對會議的流程安排相當用心，參與此次會議之篇學者可藉著此次會議，在粒度計算資訊系統與生物資訊運用之各研究領域互相作深入的討論，而且可藉此機會認識在此領域中之權威研究學者，對於此後從事此領域之研究有相當之助益。

三、　考察參觀活動(無是項活動者省略)

無。

四、　建議

無。

五、攜回資料名稱及內容

1. 大會議程
2. The Proceeding of 2008 IEEE International Conference on Granular Computing 研討會論文集。

六、　其他

A Weighted Load-Balancing Parallel Apriori Algorithm for Association Rule Mining

Kun-Ming Yu[1], Jia-Ling Zhou[2]
[1]*Department of Computer Science and Information Engineering, Chung Hua University*
[2]*Department of Information Management, Chung Hua University*
[1]*yu@ chu.edu.tw*, [2]*jlzhou@pdlab.csie.chu.edu.tw*

Abstract

Because of the exponential growth in worldwide information, companies have to deal with an ever growing amount of digital information. One of the most important challenges for data mining is quickly and correctly finding the relationship between data. The Apriori algorithm is the most popular technique in association rules mining; however, when applying this method, a database has to be scanned many times and many candidate itemsets are generated. Parallel computing is an effective strategy for accelerating the mining process. In this paper, the Weighted Distributed Parallel Apriori algorithm (WDPA) is presented as a solution to this problem. In the proposed method, metadata are stored in TID forms, thus only a single scan to the database is needed. The TID counts are also taken into consideration, and therefore better load-balancing as well as reducing idle time for processors can be achieved. According to the experimental results, WDPA outperforms other algorithms while having lower minimum support.

1. Introduction

With the rapid development of information technology, companies have been working on digitizing all areas of business to improve efficiency and thus competitiveness. However, the consequences of full-digitization are that tremendous amounts of data are generated. It is important to extract meaningful information from scattered data, and data mining techniques are developed for that purpose. There are many techniques being used for data mining, for example, Classification, Regression, Time Series, Clustering, Association Rules and Sequence. Association rule [1, 2] is one of the most useful techniques in data mining. Generally, it takes long to find the association rules between datasets when a database contains a large number of transactions. By applying parallel-distributed data mining techniques, the mining process can be effectively speeded up. With parallel-distributed data mining the calculation is done in a distributed environment [3, 7, 8, 9, 12], but most of the time, irregular and imbalanced computation loads are allocated between processors and thus the overall performance is degraded.

In this paper the Weighted Distributed Parallel Apriori algorithm (WDPA) is presented as a solution for this problem. In the proposed method, a database has only to be scanned once because metadata are stored in TID tables. This approach also takes the TID count into consideration. Therefore, WDPA improves load-balancing as well as reduces idle time of processors.

The experimental results in this study showed that the running time of WDPA was significantly faster than that of previous methods. In some cases, WDPA only used about 2% of the time used in previous methods. This can be achieved because WDPA successfully reduced the number of scan iterations to databases and was able to evenly distribute workloads among processors.

The paper is organized as follows: In section 2, association rule and parallel distributed algorithms are explained. The WDPA algorithm is proposed in section 3. Section 4 gives the experimental results. Finally, the conclusion is given in section 5.

2. Related Work

Frequent pattern mining problem is defined as follows. Let DB = $\{T_1, T_2, \ldots, T_k\}$ be a database of transactions, where each transaction T_e consists of I, I = $\{i_1, i_2, \ldots, i_m\}$ be a set of all items. Assuming A, B are itemsets, $A, B \subseteq I$, $A \cap B = \emptyset$, $A \rightarrow B$ denotes there is an association rule between A and B. Each association rule has *support* and *confidence* to confirm the validity of the rule. *Support* denotes the occurrence rate of an itemset in a DB. *Confidence* denotes the proportion of data items containing Y in all items containing X in a DB. When the *support* and *confidence* are greater than or equal to the pre-defined *minimum support* and *minimum confidence*, the association rule is considered to be a valid rule.

The Apriori algorithm was proposed by R. Agrawal and R. Srikant in 1994 A.D. [2]. The Apriori algorithm is one of the most representative algorithms in mining association rules. It is based on the assumption that subsets of low-frequency itemsets must be low-frequency as well. Even though the Apriori algorithm takes lots of time to calculate combination of itemsets, the design of the data structure makes it easy for the algorithm to be parallelized. Therefore, some scholars propose that parallel- distributed Apriori algorithms be used [3, 7, 8, 9, 10, 11, 12, 13, 14]. For example, CD, DD, FDM, FPM, DMA etc. Recently, Ye [12] proposed a parallel-distributed algorithm using Trie Structure [6]. Ye's algorithm distributes computing workload using the Trie Structure to speed up the computation, however, this causes significant variance between the sizes of candidate itemsets distributed among processors. Moreover, this method also requires a database to be scanned many times. The problems related to multiple-scan and load-imbalance gets worse when dealing with large databases and huge itemsets. Therefore, a Weighted Distributed Parallel Apriori algorithm (WDPA) is proposed. By storing the TIDs of itemsets and precisely calculating and distributing computation workloads, WDPA is able to effectively accelerate the computation of itemsets and reduce the required scan iterations to a database and balancing the load, thus significantly reduces processor idle time.

3. Weighted Distributed Parallel Apriori (WDPA) Algorithm

To avoid the problems associated with load-imbalance and multiple-scan, the WDPA algorithm is proposed so that a database only needs to be scanned once while maintaining load balancing among processors. In the algorithm, each transaction has a *Transaction IDentification*, called TID. By using hash functions to store TID in table structure, the number of itemsets can be quickly calculated without the need of rescanning the database.

In the WDPA parallel-distributed processing algorithm, the number of combinations for items, called a lattice, is first calculated. Lattice is the number of combinations calculated from candidate (k+1)-itemset counts by frequent k-itemsets. Equation (1) is the required count of itemset combinations of I_i, Equation (2) represents the total number of k-itemsets. Using block division to do frequent itemset distribution after calculating the number of combinations, is called Block_Lattice (BL) (Figure 1). From Figures 1 and 2 we can see that the Lattice count decreases gradually at the lower-left part of the matrix, thus if itemsets are distributed sequentially, the load-imbalance distribution will occur. Therefore by cyclic partitioning of lattice, itemsets are distributed to the processors cyclically to balance the distribution of itemsets. This is called $Cyclic\_Lattice(CL)$.

$$Cnt\_Lattice(I_i) = [(len(freq_{k-1}) - 1) - i] \tag{1}$$

$$TotalCnt\_Lattice = \sum_{i=0}^{len(freq_{k-1})-1} Cnt\_Lattice(I_i) \tag{2}$$

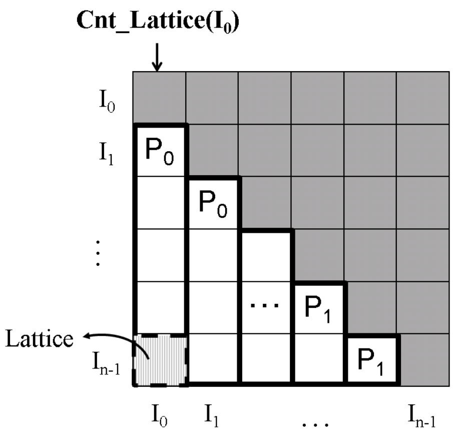


Figure 1. Block partitioning

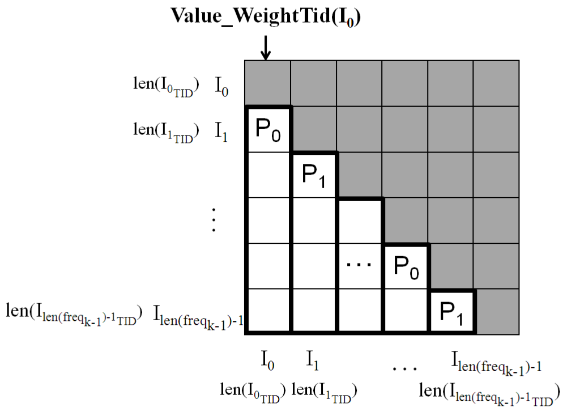


Figure 2. Cyclic partitioning

By only calculating the Lattice number and ignoring the length of the itemsets' TID, an uneven distribution of workload occurs. Therefore, this algorithm also takes TID length into consideration and regards it as a weight value, which makes the distribution of itemsets more accurate and more even. Equation (3) calculates the weighted value of itemset

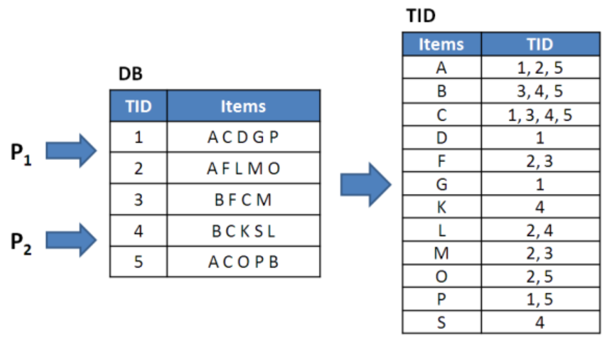
Ii. Equation (4) represents the total weight value of k-itemsets.

$$Value\_WeightTid(I_i) = \sum_{j=i+1}^{len(freq_{k-1})-1} len(I_{i_{TID}}) \times len(I_{j_{TID}}) \tag{3}$$

$$TotalValue\_WeightTid = \sum_{i=0}^{len(freq_{k=1})-1} \sum_{j=i+1}^{len(freq_{k-1})-1} len(I_{i_{TID}}) \times len(I_{j_{TID}}) \tag{4}$$

There are two methods of partitioning weighted TID, the *Block_WeightTid(BWT)* partitions and distributes TID by block, the *Cyclic_WeightTid(CWT)* partitions and distributes TID in cyclic.

An example of the algorithms is given below: For step1 and step2, P_1 (MP), P_2 (SP) read and scan database, then build level-1 candidate itemsets. (Figure 3)

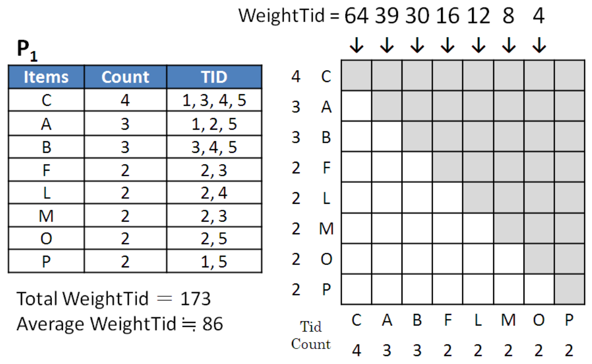


TID	Items
1	A C D G P
2	A F L M O
3	B F C M
4	B C K S L
5	A C O P B

Items	TID
A	1, 2, 5
B	3, 4, 5
C	1, 3, 4, 5
D	1
F	2, 3
G	1
K	4
L	2, 4
M	2, 3
O	2, 5
P	1, 5
S	4

Figure 3. Scan database and creating TID forms

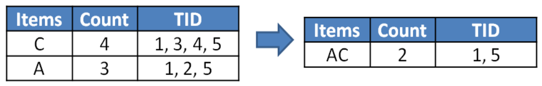
Figure 4 shows that P_1 collects itemsets that match given support into frequent 1-itemsets, then uses CWT to calculate and distribute the itemsets on P_1. P_1: {C, B}, P_2: {A, F, L, M, O}. (Step 3 and Step 4)



Items	Count	TID
C	4	1, 3, 4, 5
A	3	1, 2, 5
B	3	3, 4, 5
F	2	2, 3
L	2	2, 4
M	2	2, 3
O	2	2, 5
P	2	1, 5

Figure 4. Distributing frequent 1-itemsets on P_1

Figure 5 describes P_1 and P_2 combining level-2 candidate itemsets and calculating itemset counts according to the TID table. Figure 6 represents level-2 candidate itemset counts on P_1 and P_2. (Take level-2 candidate A and C for example, the intersection of A and C on TID, [1, 5], is the resulting set, AC) (Step 5 and Step 6)



Items	Count	TID
C	4	1, 3, 4, 5
A	3	1, 2, 5

Items	Count	TID
AC	2	1, 5

Figure 5. Itemsets are calculated by counting the TID forms

P_1			P_2		
Itemsets	Count	TID	Itemsets	Count	TID
AC	2	1, 5	AB	1	5
BC	3	3, 4, 5	AF	1	2
CF	1	3	AL	1	2
CL	1	3	AM	1	2
CM	1	3	AO	2	2, 5
CO	1	5	AP	2	1, 5
CP	2	1, 5	FL	1	2
BF	1	3	FM	2	2, 3
BL	1	4	FO	1	2
BM	1	3	FP	0	(null)
BO	1	5	LM	1	2
BP	1	5	LO	1	2
			LP	0	(null)
			MO	1	2
			MP	0	(null)

Figure 6. P_1 and P_2 level-2 candidate itemsets

Select the itemsets that match the given support value, and save them as frequent 2-itemsets. Because the frequent 1-itemsets are larger, candidate itemsets that required combination computation will be larger, too. In this case, distributing the itemsets in Cyclic will produce better results. On the other hand, if there are frequent itemsets above level 1, candidate itemsets that required combination computation will be smaller, too. In this case, distributing the itemsets in Block will produce better results.

P_1 receives P_2 itemsets, and repeats execution step 4 to step 9 until there are no more frequent itemsets. Figure 7 illustrates the use of BWT to calculate and distribute the itemsets on P_1. P_1: {BC, AC}, P_2: {AO, AP, CP}. (Step 3 and step 4)

Figure 8 represents P_1 combined level-3 candidate itemsets matching given support value into frequent 3-itemset.

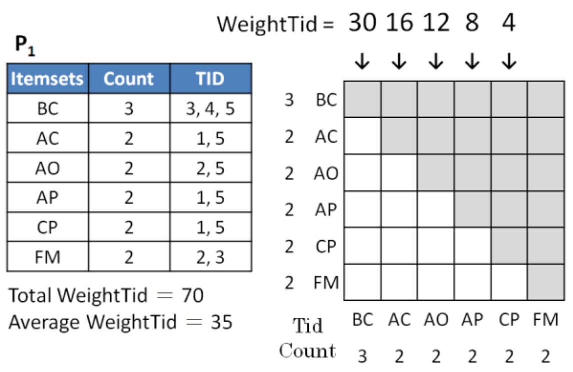


Figure 7. Distributing frequent 2-itemsets on P_1

P_1

Itemsets	Count	TID
ACP	2	1, 5

Figure 8. The resulting frequent 3-itemsets

Because the resulting frequent 3-itemset contains only one itemset, no more combination operation can be made, the mining process ends here. The number of resulting frequent itemsets are: level-1: Eight, level-2: Six, level-3: One.

The algorithms are described in detail below:

Input: a transaction database DB = $\{T_1, T_2, ..., T_n\}$, and each transaction $T_i \subseteq I$, $I = \{i_1, i_2, ..., i_m\}$. A given minimum support s. P is the number of processors. (p_1 is master processor (MP), and $p_2, p_3, ..., p_p$ are salve processors (SPs))
Output: All frequent itemsets.
Method:
Step 1. Each processor reads the database DB.
Step 2.Each processor scans DB and creates the transaction identification set (TID).
Step 3. Each processor calculates candidate k-itemset counts, when the count is greater than s, let freqk be frequent k-itemsets.
Step 4. MN equally divides the $freq_k$ into p disjointed partitions and assigns $itemsets_i$ to p_i. $Itemsets_i$ denote that SPs were assigned to the itemsets from MN. The frequent pattern dividing method:
(1) Block_Lattice (BL)
(2) Cyclic_Lattice (CL)
(3) Block_WeightTid (BWT)
(4) Cyclic_WeightTid (CWT)
Step 5. Each processor receives the $itemsets_i$ and the combination candidate (k+1)-itemsets.
Step 6. Each processor candidate itemsets is calculated by counting the TID forms.
Step 7. When itemset count is greater than s then it is a frequent (k+1)-itemset, and itemset appeared in transaction id is saved to (k+1)-TID.
Step 8. SPs send frequent itemsets to MN.
Step_9.MN receives SPs itemsets, and repeats execution step4 to setp9 until there are no more frequent itemsets.

4. Experiments

In order to evaluate the performance of the proposed algorithm, the WDPA was implemented along with the algorithm proposed by Ye [12]. The program was executed in a PC cluster with 16 computing processors. Table 1 gives the hardware and software specifications. Synthesized datasets generated by IBM's Quest Synthetic Data Generator [4] were used to verify the algorithm. Moreover, the database T10I4D50N100K, T10I4D100N100K, T10I4D200N100K was used to examine the WDPA. From the experimental results, our proposed method balances the workload among processors and saves on processor idle time because of the way CWT distributes itemsets. Therefore, the following experiments are calculated based on the CWT method.

Table 1. Hardware and Software Specifications

Hardware Environment	
CPU	AMD Athlon Processor 2200+
Memory	1GB DDR Ram
Network	100 Mbps interconnection network
Disk	80GB IDE H.D.
Software Environment	
O.S.	ReadHat Linux 7.3
Library	MPICH2 1.0.3

Figure 9 shows the speed up of four WDPA methods. From the results, it can be seen that the speeded up of the four methods is similar, but the CWT itemsets distributed used weighted TID and cyclic partition, therefore the CWT have more accurately parallel-distributed itemsets. According to the experiment, the CWT, regardless of processor numbers of 1,2,4,8,16, achieves better results than the other methods.

Figure 10 shows the execution time of the WDPA and Ye's algorithm on different processors. WDPA(8) denotes that the WDPA algorithm used eight processors. Because Ye's algorithm needs to repeatedly scan the database, the loads are imbalanced between processors. Therefore, from Figure 10, WDPA is nearly 120 times faster than Ye's algoritm in the 16 processors case. The use of TID form in WDPA accurately parallelize the workloads, hence it effectively reduced the database scanning and saved on processor idle time.

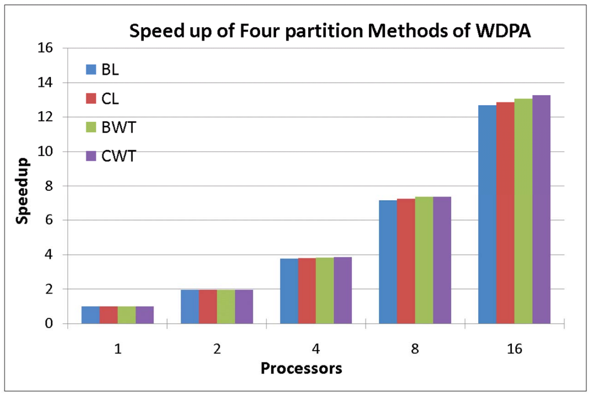


Figure 9. Speedup of Four partition Methods of WDPA (T10I4D100KN100K, minsup: 0.2%)

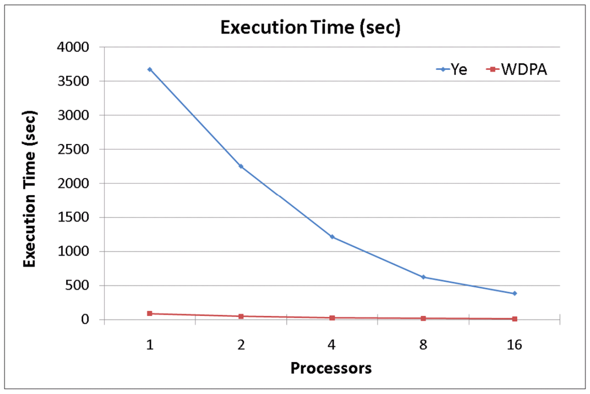


Figure 11. Execution Time (WDPA vs. Ye's algorithm)(T10I4D100KN100K, minsup: 0.2%)

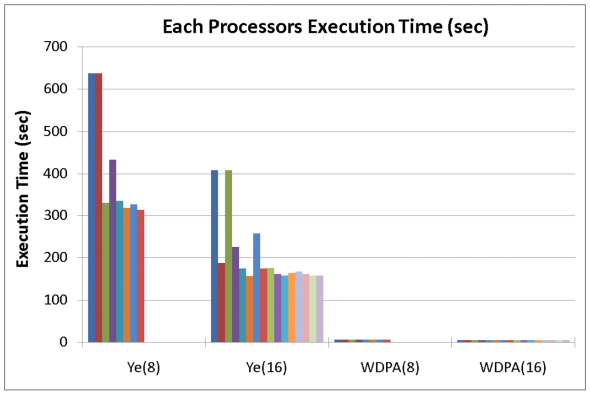


Figure 10. Each Processors Execution Time (T10I4D50KN100K, minsup: 0.2%)

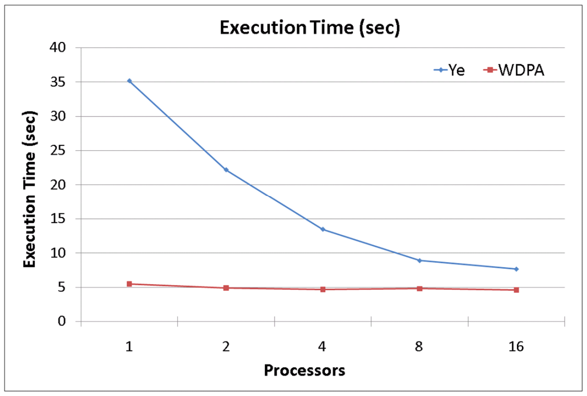


Figure 12. Execution Time (WDPA vs. Ye's algorithm)(T10I4D100KN100K, minsup: 0.3%)

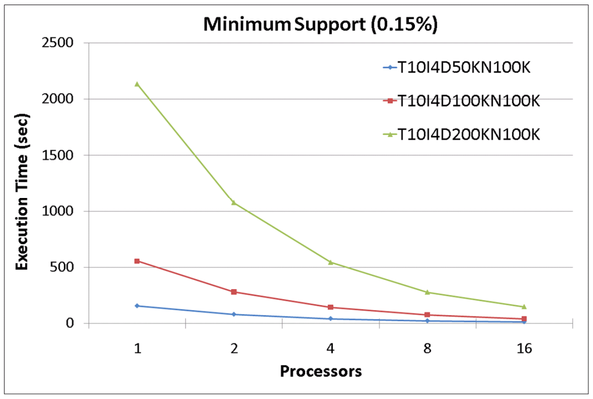


Figure 13. WDPA Execution Time, minsup: 0.15%

Figure 11 and 12 show the execution time and speedup under different given supports. Ye's algorithm requires that a database being re-scanned for every itemset to be counted during the mining process, so when there is lower support, Ye's algorithm takes longer to re-scan the database. On the other hand, using the TID table with precise distribution of itemsets, the WDPA scans the database once only. This greatly reduced the time spent on database scanning and balanced the computation workload among processors. Thus, there is an obvious performance advantage of the WDPA algorithm over Ye's algorithm.

Figures 13 and 14 give the execution time and speed up with different databases. With the increased size of the database, the length of the TID of itemsets in the table will increase. Therefore, when the size of the database increased, the execution took longer. Moreover, by parallel-distributing the processing, large databases can be more effectively mined and itemsets will be allocated to different processors to perform the calculation, this significantly speeding up the mining process.

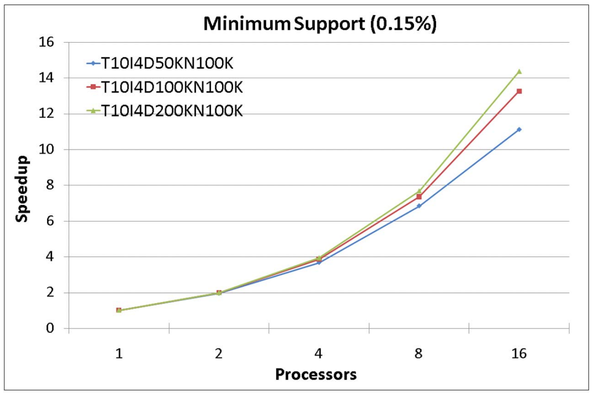


Figure 14. WDPA Speedup, minsup: 0.15%

Figure 15 illustrates the execution time of various minimum supports. The number of frequent itemsets as well as their length increased with a lower support in WDPA. Therefore, by using this method of calculation WDPA effectively accelerated the computation. Thus WDPA can achieve better speedup when there was lower minimum support.

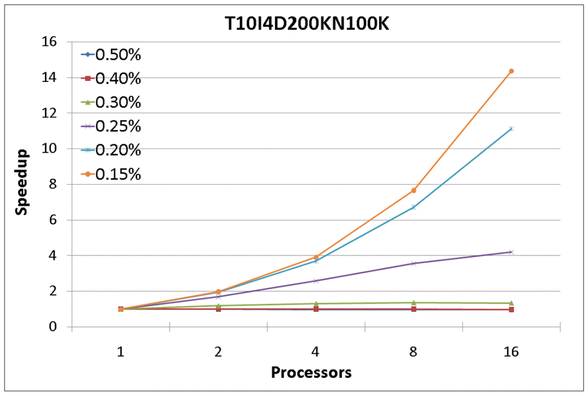


Figure 15. WDPA Speedup

5. Conclusion

Determining the association between items in a huge database, is a worthwhile research topic. However, the process of generating itemsets and confirming them is time consuming. Parallel-distributed computation strategies provide workable solutions to this problem. In this paper, a Weighted Distributed Parallel Apriori algorithm (WDPA) is proposed, in which the TID of itemsets is stored in a table to compute their occurrence. WDPA effectively reduced the required scan iterations to a database as well as accelerated the calculation of itemsets. By taking the factor of itemset counts into consideration, this approach effectively balanced workloads among processors and reduced processor idle time. Experimental results show that WDPA achieved higher speedups than pervious works in the case of high data volume and low support.

References

[1] Agawal, R., Imilinski, T., and Swami, A. “Mining Association Rules between Sets of Items in Large Databases,” *Procceeding of the 1993 ACM SIGMOD International Conference on Management of Data*, Vol. 22, Issue 2, June 1993, pp. 207-216.

[2] Agrawal, R. and Srikant, R. “Fast Algorithms for Mining Association Rules,” *Proceeding of 20th International conference on Very Large Database*, 1994, pp. 487-499.

[3] Agrawal, R. and Shafer, J.C. “Parallel Mining of Association Rules,” *IEEE Transaction On Knowledge And Data Engineering*, Vol. 8, Issue 6, December 1996, pp. 962-969.

[4] Almaden, “I. Quest synthetic data generation code,” http://www.almaden.ibm.com/cs/quest/syndata.html.

[5] Apte, C. and Weiss, S.M. “Data Mining with Decision Trees and Decision Rules,” *Future Generation Computer Systems*, Vol. 13, Issue 2-3, November 1997, pp. 197-210.

[6] Bodon, F. “A fast Apriori implementation,” *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations*, 2003.

[7] Cheung, D.W., Han, J., Ng, V.T., Fu, A.W., and Fu, Y. “A fast distributed algorithm for mining association rules,” *Fourth International Conference on Parallel and Distributed Information Systems*, December 1996, pp. 31-42.

[8] Cheung, D.W., Lee, S.D., and Xiao, Y. “Effect of Data Skewness and Workload Balance in Parallel Data Mining,” *IEEE Transactions On Knowledge And Data Engineering*, Vol. 14, Issue 3, 2002, pp. 498-514.

[9] Cheung, D.W., Ng, V.T., and Fu, A.W. “Efficient Mining of Association Rules in Distributed Databases,” *IEEE Transaction On Knowledge And Data Engineering*, Vol. 8, Issue 6, December 1996, pp. 911-922.

[10] Einakian, S. and Ghanbari, M. “Parallel Implementation of Association Rule in Data Mining,” *Proceedings of the 38th Southeastern Symposium on System Theory*, 2006, pp. 21-26.

[11] Parthasarathy, S., Zaki, M.J., Ogihara, M., and Li, W. “Parallel Data Mining for Association Rules on Shared-Memory Systems,” *Knowledge and Information Systems*, Vol. 3, Issue 1, 2001, pp.1-29.

[12] Ye, Y. and Chiang, C.C. “A Parallel Apriori Algorithm for Frequent Itemsets Mining,” *Proceedings of the Fourth International Conference on Software Engineering Research, Management and Applications*, 2006, pp. 87-94.

[13] Zaki, M.J., Ogihara, M., Parthasarathy, S., and Li, W. “Parallel Data Mining for Association Rules on Shared-memory Multi-processors,” *Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM)*, 1996.

[14] Zaki, M.J., Parthasarathy, S., Ogihara, M., and Li, W. “Parallel Algorithms for Discovery of Association Rules,” *Data Mining and Knowledge Discovery*, Vol. 1, Issue 4, 1997, pp. 343-373.