BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ TP. HCM

CÔNG MINH HIẾU

KHAI THÁC SONG SONG TẬP PHỔ BIẾN DỰA TRÊN MẢNG SYSTOLIC

LUẬN VĂN THẠC SĨ

Chuyên ngành: Công nghệ thông tin

Mã số ngành: 60480201

TP. HỒ CHÍ MINH, tháng 04 năm 2016

BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ TP. HCM

CÔNG MINH HIẾU

KHAI THÁC SONG SONG TẬP PHỔ BIẾN DỰA TRÊN MẢNG SYSTOLIC

LUẬN VĂN THẠC SĨ

Chuyên ngành: Công nghệ thông tin

Mã số ngành: 60480201

CÁN BỘ HƯỚNG DẪN KHOA HỌC: TS. BÙI ĐỨC MINH

TP. HỒ CHÍ MINH, tháng 04 năm 2016

CÔNG TRÌNH ĐƯỢC HOÀN THÀNH TẠI

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ TP. HCM

Cán bộ hướng dẫn khoa học : TS. Bùi Đức Minh

Luận văn Thạc sĩ được bảo vệ tại Trường Đại học Công nghệ TP. HCM ngày 10 tháng 09 năm 2016

Thành phần Hội đồng đánh giá Luận văn Thạc sĩ gồm:

(Ghi rõ họ, tên, học hàm, học vị của Hội đồng chấm bảo vệ Luận văn Thạc sĩ)

TT	Họ và tên	Chức danh Hội đồng
1	TS. Trần Đức Khánh	Chủ tịch
2	PGS.TS. Võ Đình Bảy	Phản biện 1
3	TS. Lư Nhật Vinh	Phản biện 2
4	TS. Cao Tùng Anh	Ủy viên
5	Ts. Nguyễn Thị Thúy Loan	Ủy viên, Thư ký

Xác nhận của Chủ tịch Hội đồng đánh giá Luận văn sau khi Luận văn đã được sửa chữa (nếu có).

Chủ tịch Hội đồng đánh giá LV

TS. Trần Đức Khánh

TRƯỜNG ĐH CÔNG NGHỆ TP. HCM
PHÒNG QLKH – ĐTSĐH

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập – Tự do – Hạnh phúc

TP. HCM, ngày 15 *tháng 04 năm 20*16

NHIỆM VỤ LUẬN VĂN THẠC SĨ

Họ tên học viên: Công Minh Hiếu Giới tính: Nam
Ngày, tháng, năm sinh: 29/10/1991 Nơi sinh: Phú Yên
Chuyên ngành: Công nghệ thông tin MSHV: 1441860046

I- Tên đề tài:

Khai thác song song tập phổ biến dựa trên mảng Systolic.

II- Nhiệm vụ và nội dung:

- Tổng hợp phân tích các nghiên cứu về khai thác song song tập phổ biến.
- Nghiên cứu về khai thác tập phổ biến sử dụng mảng Systolic để khai thác song song tập phổ biến trên một máy tính thay vì trên hệ thống song song.

III- Ngày giao nhiệm vụ: 30/09/2015

IV- Ngày hoàn thành nhiệm vụ: 15/04/2016

V- Cán bộ hướng dẫn: TS. Bùi Đức Minh

CÁN BỘ HƯỚNG DẪN **KHOA QUẢN LÝ CHUYÊN NGÀNH**

TS. Bùi Đức Minh

LỜI CAM ĐOAN

Tôi xin cam đoan đây là công trình nghiên cứu của riêng tôi. Các số liệu, kết quả nêu trong Luận văn là trung thực và chưa từng được ai công bố trong bất kỳ công trình nào khác.

Tôi xin cam đoan rằng mọi sự giúp đỡ cho việc thực hiện Luận văn này đã được cảm ơn và các thông tin trích dẫn trong Luận văn đã được chỉ rõ nguồn gốc.

Học viên thực hiện Luận văn

Công Minh Hiếu

LỜI CÁM ƠN

Trước tiên tôi xin chân thành cảm ơn thầy TS. Bùi Đức Minh đã tận tình giúp đỡ, chỉ dạy tôi trong suốt quá trình nghiên cứu và thực hiện luận văn này.

Tôi cũng xin cảm ơn quý thầy cô Khoa Công Nghệ Thông Tin – Trường Đại Học Công Nghệ Thành Phố Hồ Chí Minh đã tận tình chỉ dạy, truyền đạt các kiến thức bổ ích qua các môn học trong chương trình đào tạo.

Cuối cùng tôi xin chân thành cảm ơn gia đình, bạn bè, các anh chị trong lớp cao học đã giúp đỡ, tạo điều kiện cho tôi trong suốt quá trình học tập.

TP.Hồ Chí Minh, 2016

Công Minh Hiếu

TÓM TẮT

Khai thác tập phổ biến là bài toán cơ bản khi khai thác dữ liệu. Một trong những vấn đề lớn của khai thác tập phổ biến là thời gian thực thi của thuật toán khai thác quá dài, chi phí cho hệ thống phần cứng lớn. Để giải quyết vấn đề này, luận văn đã tìm hiểu, nghiên cứu một phương pháp khai thác song song tập phổ biến dựa trên mảng Systolic với mục tiêu hướng đến là để giảm thời gian tính toán. Ưu điểm chính của thuật toán này là có thể chạy song song nhiều luồng dữ liệu cùng một lúc trên một máy tính. Qua đó, chi phí thiết kế phần cứng và thời gian khai thác dữ liệu được giảm đi đáng kể. Để đánh giá thuật toán, tôi đã thí nghiệm trên các cơ sở dữ liệu giao dịch chuẩn dành cho khai thác tập phổ biến. Hiệu quả cuộc thuật toán được chứng minh bằng cách so sánh thời gian khai thác so với các thuật toán song song khác.

ABSTRACT

Frequent itemsets mining is one of the most important concepts in data mining. One of the major problems in frequent itemset mining is the long execution time of extraction algorithm, the high cost of hardware systems. To solve this problem, the thesis has researched a method to parallel frequent itemset mining based on Systolic array. The target of this method is to reduce the execute time of mining. The main advantage of this algorithm is able to run in parallel multiple data streams simultaneously on one computer. Therefore, the cost of hardware system and data extraction time are reduced significantly. To evaluate the algorithm, experimentations are done on the database transaction set for mining frequent itemset. The algorithm efficiency is proven by comparing extraction time to other parallel algorithms.

MỤC LỤC

DANH MỤC CÁC TỪ VIẾT TẮT

Từ viết tắt	Nghĩa tiếng Anh	Nghĩa tiếng Việt
KTDL	Data mining	Khai thác dữ liệu
CSDL	Database	Cơ sở dữ liệu
CNTT	Information technology	Công nghệ thông tin
CPU	Central Processing Unit	Bộ vi xử lý trung tâm
PE	Processing Unit	Đơn vị xử lý
GPU	Graphic Processing Unit	Bộ xử lý đồ họa
BXL	Cell	Bộ xử lý
Sup	(Support)	Độ phổ biến
Conf	Confidence	Độ tin cậy
Minsup	Minimum support	Độ phổ biến tối thiểu
Minconf	Minimum confidence	Độ tin cậy tối thiểu
SA	Systolic array	Mảng Systolic
SAP	Systolic array processor	Bộ xử lý trên mảng Systolic
SABMA	Systolic array based mining algorithm	Thuật toán khai thác dựa trên mảng Systolic
FP-Growth	FP-Growth algorithm	Thuật toán FP-growth
Eclat	Eclat algorithm	Thuật toán Eclat
FP-Tree	Frequent Pattern tree	Cây FP-Tree
SISD	Single Instruction Single Data	Đơn dòng lệnh đơn luồng dữ liệu
SIMD	Single Instruction Multiple Data	Đơn dòng lệnh đa luồng dữ

		liệu
MISD	Multiple Instruction Single Data	Đa dòng lệnh đơn luồng dữ liệu
MIMD	Multiple Instruction Multiple Data	Đa dòng lệnh đa luồng dữ liệu
VLSI	Very large scale integration	
SQL	Structured query language	Ngôn ngữ truy vấn có cấu trúc
ACID	Atomicity, Consistency, Isolation, Durability	tính nguyên tố, tính nhất quán, tính tách biệt, tính bền vững

DANH MỤC CÁC BẢNG

DANH MỤC CÁC BIỂU ĐỒ, SƠ ĐỒ, HÌNH ẢNH

MỞ ĐẦU

Thế giới đang ở trong thời đại mà thông tin có giá trị rất lớn. Trong kinh doanh ai có nhiều thông tin hơn người đó sẽ làm chủ thị trường, trong nghiên cứu ai càng nhiều thông tin thì người đó càng có nhiều cơ hội thành công hơn. Với sự phát triển và ứng dụng của công nghệ thông tin vào nhiều lĩnh vực, đặc biệt là Internet thì lượng thông tin đó càng ngày càng trở nên khổng lồ. Để thu thập những thông tin có ích từ những nguồn dữ liệu khổng lồ cần những kỹ thuật, công cụ mới để chuyển đổi nguồn dữ liệu này thành tri thức có ích. Mặt khác, trong môi trường cạnh tranh hiện nay thì người ta cần thông tin với tốc độ nhanh để giúp việc ra quyết định được hiệu quả hơn. Việc sử dụng các kỹ thuật khai thác dữ liệu để thu thập và trích xuất những thông tin hữu ích tiềm ẩn trong các cơ sở dữ liệu lớn là quá trình phát hiện tri thức trong cơ sở dữ liệu. Khai thác dữ liệu hiện nay là lĩnh vực mang tính cấp thiết của nền công nghệ thông tin. Nhiều tổ chức, công ty lớn trên thế giới đã áp dụng khai thác dữ liệu vào các hoạt động của mình và thu được nhiều lợi ích.

Các bước khai thác dữ liệu (KTDL) được chia làm nhiều giai đoạn và sử dụng nhiều kỹ thuật như: phân lớp (classification), gom cụm (clustering), tổng hợp (summarization), khai thác tập phổ biến (frequent itemset) để phát hiện luật kết hợp (association rules)… Trong đó khai thác tập phổ biến là một vấn đề kinh điển trong KTDL. Nó là một quá trình để tìm kiếm các tập phổ biến được ẩn trong một khối lượng lớn dữ liệu để tóm gọn hoặc mô hình hóa cơ sở dữ liệu, từ đó phát hiện ra các luật kết hợp.

Nhiều thuật toán đã được nghiên cứu và phát triển để giúp cho việc khai thác tập phổ biến trở nên hiệu quả như: Apriori, FP-Tree, FP-Growth, Eclat… Các thuật toán này đều giúp hỗ trợ tăng tốc độ xử lý CSDL, giảm thời gian và chi phí KTDL. Một số thuật toán được cài đặt để xử lý song song CSDL trên nhiều hệ thống máy tính nên thời gian khai thác tập phổ biến được giảm đi đáng kể. Mặc dù vậy, vẫn còn tồn tại nhiều vấn đề như: các kết quả thu được quá nhiều làm cho thời gian chạy của thuật toán quá lớn, đặc biệt là ở các CSDL có chiều cao lớn. Cách hữu ích và thích hợp để giải quyết vấn đề này là sử dụng kỹ thuật song song và phân tán. Khai

thác song song dữ liệu giúp tăng đáng kể khả năng phân tích dữ liệu vì sức mạnh tính toán được tăng lên. Tuy nhiên, nó cũng gặp phải một số thách thức như cần lượng giao tiếp lớn, yêu cầu về thiết bị phần cứng cao, cần phải đồng bộ hóa và cân đối các số liệu. Do đó, cần tìm ra thuật toán hiệu quả để tận dụng tối đa năng suất làm việc của các thiết bị phần cứng giúp giảm chi phí và tăng hiệu suất KTDL.

Vì vậy, luận văn chọn đề tài về thuật toán "Khai thác song song tập phổ biến dựa trên mảng Systolic" để tìm hiểu, nghiên cứu nhằm xây dựng một thuật toán khai thác song song tập phổ biến tận dụng được tối đa sức mạnh của phần cứng máy tính và tiết kiệm chi phi giao tiếp giữa các hệ thống song song.

Mục tiêu đề tài:

Tìm hiểu, nghiên cứu phương pháp khai thác song song tập phổ biến bằng cách xử lý đồng thời nhiều luồng dữ liệu cùng một lúc để thời gian tính toán có thể ở mức chấp nhận được. Sử dụng kiến trúc mảng Systolic (Systolic array) để có thể điều khiển một cách đều đặn và tận dụng được khả năng song song của tất cả các phần tử tham gia xử lý dữ liệu. Qua đó, có thể giảm thiểu chi phí về cơ sở hạ tầng của thiết bị phần cứng và nâng cao hiệu quả khai thác, thời gian cần thiết để KTDL cũng sẽ giảm đi.

Cài đặt chương trình khai thác CSDL giao dịch dựa trên thuật toán đã tìm hiểu để so sánh với các thuật toán song song đã được phát triển trước đó nhằm tìm ra ưu, nhược điểm của thuật toán được nghiên cứu.

Nội dung nghiên cứu:

- Tìm hiểu các khái niệm về KTDL, khai thác tập phổ biến và phát hiện luật kết hợp.
- Thu thập, nghiên cứu các CSDL giao dịch và các khái niệm, kỹ thuật liên quan.
- Các thuật toán khai thác song song tập phổ biến đã và đang được phát triển.
- Phương pháp khai thác song song sử dụng mảng Systolic và ưu, nhược điểm của thuật toán.

- Xây dựng chương trình để kiểm chứng và so sánh thuật toán với các thuật toán song song khác.

Bố cục luận văn:

Chương 1: Cơ sở lý thuyết

Chương 2: Một số phương pháp khai thác tập phổ biến

Chương 3: Thuật toán khai thác song song tập phổ biến dựa trên mảng Systolic

Chương 4: Xây dựng chương trình thử nghiệm

- Tìm kiếm và thu thập dữ liệu
- Cài đặt chương trình thực nghiệm
- Kết quả và đánh giá

CHƯƠNG 1. CƠ SỞ LÝ THUYẾT

1.1. Tổng quan về khai thác dữ liệu

1.1.1. Mục tiêu của khai thác dữ liệu

Khai thác dữ liệu là một khái niệm ra đời vào những năm cuối của thập kỷ 80, nó là quá trình tìm kiếm, khám phá dưới nhiều góc độ khác nhau nhằm phát hiện các mối liên hệ, quan hệ giữa các dữ liệu, đối tượng bên trong CSDL để rút trích và tìm ra được các mẫu, các mô hình hay thông tin mới, tri thức tiềm ẩn có trong CSDL chủ yếu phục vụ cho mô tả và dự đoán. Đây là giai đoạn quan trọng nhất trong quá trình phát hiện tri thức từ CSDL, hỗ trợ cho việc ra quyết định, điều hành trong khoa học và kinh doanh.

Khai thác dữ liệu là tiến trình khám phá tri thức tiềm ẩn trong các CSDL, cụ thể hơn, đó là tiến trình lọc, sản sinh những tri thức hoặc các mẫu tiềm ẩn, những thông tin hữu ích từ các CSDL lớn.

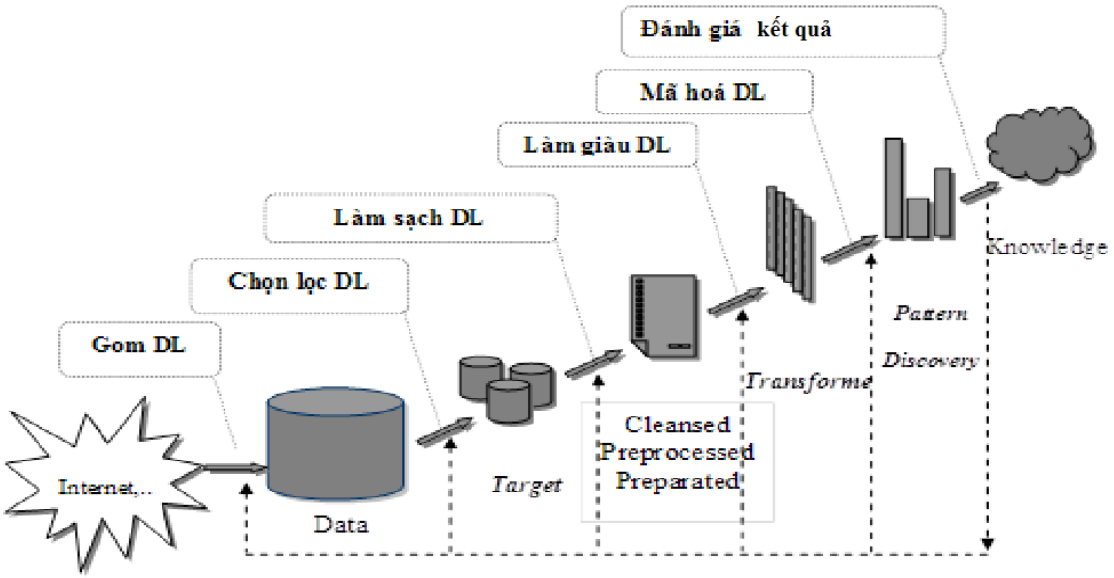
Yếu tố thành công trong mọi hoạt động kinh doanh ngày nay là việc biết sử dụng thông tin có hiệu quả, có nghĩa là từ các dữ liệu có sẵn phải tìm ra những thông tin tiềm ẩn mà trước đó chưa được phát hiện, tìm ra những xu hướng phát triển và những yếu tố tác động lên chúng. Thực hiện công việc này chính là quá trình phát hiện tri thức trong CSDL mà trong đó các kỹ thuật cho phép ta lấy được các tri thức chính là kỹ thuật KTDL.

Nếu quan niệm tri thức là mối quan hệ của các mẫu giữa các phần tử dữ liệu thì quá trình phát hiện tri thức chỉ toàn bộ quá trình trích xuất tri thức từ CSDL, trong đó trải qua nhiều giai đoạn khác nhau như: Tìm hiểu và phát hiện vấn đề, thu thập và tiền xử lý dữ liệu, phát hiện tri thức, minh hoạ, đánh giá tri thức được phát hiện và đưa kết quả vào thực tế.

1.1.2. Quá trình phát hiện tri thức từ cơ sở dữ liệu

Phát hiện tri thức từ CSDL là quá trình sử dụng nhiều phương pháp và công cụ tin học nhưng vẫn là quá trình mà trong đó con người làm trung tâm. Do đó, nó

không phải là một hệ thống phân tích tự động mà là một hệ thống bao gồm nhiều hoạt động tương tác thường xuyên giữa con người với CSDL được hỗ trợ bởi nhiều hệ thống, công cụ [2].



Hình 1.1. Quá trình phát hiện tri thức từ CSDL [3]

Hình 1.1 cho thấy quá trình phát hiện tri thức từ CSDL. Bắt đầu của quá trình là kho dữ liệu thô và kết thúc với tri thức được chiết xuất ra. Về lý thuyết thì có vẻ đơn giản nhưng thực sự đây là một quá trình rất khó khăn, gặp phải nhiều vướng mắc như: quản lý các tập dữ liệu, lặp đi lặp lại toàn bộ quá trình…

- Gom dữ liệu: Tập hợp dữ liệu là bước đầu tiên trong quá trình KTDL. Đây là bước được khai thác trong một cơ sở dữ liệu, một kho dữ liệu và thậm chí các dữ liệu từ các nguồn ứng dụng Web.
- Lựa chọn dữ liệu: Ở giai đoạn này dữ liệu được lựa chọn hoặc phân chia theo một số tiêu chuẩn nào đó, ví dụ chọn tất cả những người có tuổi đời từ 25–35 và có trình độ đại học.
- Làm sạch, tiền xử lý và chuẩn bị trước dữ liệu: Giai đoạn này là giai đoạn rất quan trọng trong quá trình KTDL nhưng thường bị đánh giá thấp. Một số lỗi thường mắc phải trong khi gom dữ liệu là tính không đủ chặt chẽ, logic. Vì vậy, dữ liệu thường chứa các giá trị vô nghĩa và không có khả năng kết nối dữ liệu.

Ví dụ: tuổi của người = 200 (quá lớn so với thực tế). Giai đoạn này sẽ tiến hành xử lý những dạng dữ liệu không chặt chẽ nói trên. Những dữ liệu dạng này được xem như thông tin dư thừa, không có giá trị. Bởi vậy, đây là một quá trình rất quan trọng vì dữ liệu này nếu không được "làm sạch-tiền xử lý-chuẩn bị trước" thì sẽ gây nên những kết quả sai lệch nghiêm trọng.

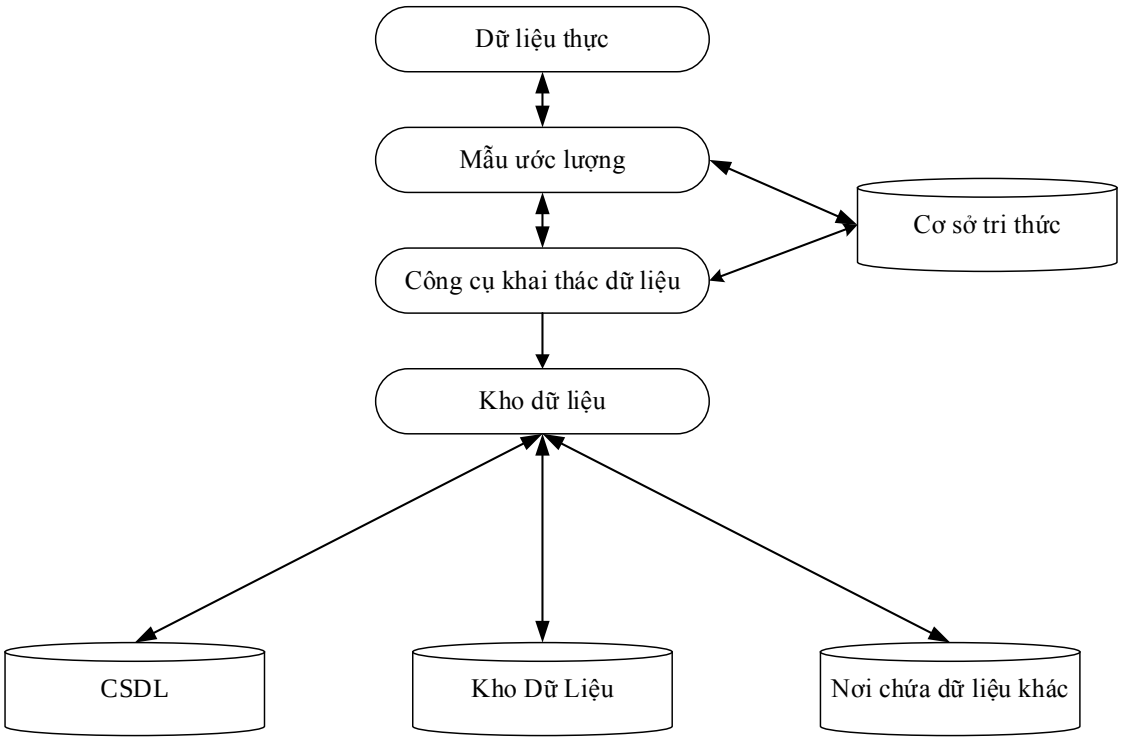
- Chuyển đổi dữ liệu: Trong giai đoạn này, dữ liệu đưa ra có thể sử dụng và điều khiển được bởi việc tổ chức lại nó. Dữ liệu đã được chuyển đổi phù hợp với mục đích khai thác.
- Phát hiện và trích mẫu dữ liệu: Đây là bước mang tính tư duy trong KTDL. Ở giai đoạn này nhiều thuật toán khác nhau đã được sử dụng để trích ra các mẫu từ dữ liệu. Thuật toán thường dùng là nguyên tắc phân loại, nguyên tắc kết hợp hoặc các mô hình dữ liệu tuần tự…
- Đánh giá kết quả: Đây là giai đoạn cuối trong quá trình KTDL. Ở giai đoạn này, các mẫu dữ liệu được chiết xuất ra bởi phần KTDL. Không phải bất cứ mẫu dữ liệu nào cũng đều hữu ích, đôi khi nó còn bị sai lệch. Vì vậy, cần phải ưu tiên những tiêu chuẩn đánh giá để chiết xuất ra các tri thức.

Trong quá trình KTDL với sáu giai đoạ trên, trong đó giai đoạn thứ năm là giai đoạn được quan tâm nhiều nhất, rất nhiều các nghiên cứu, thuật toán được giới thiệu và phát triển nhằm tối ưu hóa quá trình KTDL được thực hiện ở giai đoạn này.

1.1.3. Kiến trúc của một hệ thống khai thác dữ liệu

KTDL là quá trình rút trích thông tin bổ ích từ những kho dữ liệu lớn. KTDL là quá trình chính trong khai thác tri thức từ CSDL.

Kiến trúc của một hệ KTDL có các thành phần như hình 1.2:



Hình 1.2. Khám phá tri thức trong CSDL điển hình

CSDL, kho dữ liệu hoặc lưu trữ thông tin khác: Đây là một hay các tập CSDL, các trang tính hay các dạng khác của thông tin được lưu trữ. Các kỹ thuật làm sạch dữ liệu và tích hợp dữ liệu có thể được thực hiện.

Máy chủ CSDL: Máy chủ có trách nhiệm lấy những dữ liệu thích hợp dựa trên những yêu cầu khám phá của người dùng.

Cơ sở tri thức: Đây là miền tri thức dùng để tìm kiếm hay đánh giá độ quan trọng của các mẫu kết quả thu được. Tri thức này có thể bao gồm một sự phân cấp khái niệm dùng để tổ chức các thuộc tính hay các giá trị thuộc tính ở các mức trừu tượng khác nhau.

Máy khai thác dữ liệu: là một hệ thống KTDL cần phải có một tập các mô đun chức năng để thực hiện công việc, chẳng hạn như kết hợp, phân lớp, gom cụm.

Mô đun đánh giá mẫu: Bộ phận tương tác với các mô đun KTDL để tập trung vào việc duyệt tìm các mẫu hữu ích. Nó có thể dùng các ngưỡng về độ phổ biến để lọc mẫu đã khám phá được. Cũng có thể mô đun đánh giá mẫu được tích hợp vào mô đun KTDL, tùy theo cách cài đặt của phương pháp KTDL được dùng.

Giao diện đồ họa cho người dùng: Bộ phận này cho phép người dùng giao tiếp với hệ thống KTDL. Thông qua giao diện này người dùng tương tác với hệ thống bằng cách đặc tả một yêu cầu khai thác hay một nhiệm vụ, cung cấp thông tin trợ giúp cho việc tìm kiếm và thực hiện khai thác thăm dò trên các kết quả khai thác trung gian. Ngoài ra bộ phận này còn cho phép người dùng xem các lược đồ CSDL, lược đồ kho dữ liệu, các đánh giá mẫu và hiển thị các mẫu trong các khuôn dạng khác nhau.

1.1.4. Các phương pháp khai thác dữ liệu

Có rất nhiều các phương pháp KTDL, mỗi phương pháp có đặc điểm riêng về biểu diễn mô hình, đánh giá mô hình và cách tìm kiếm, phù hợp với một lớp các bài toán với các dạng dữ liệu và miền dữ liệu nhất định. Dưới đây là một số phương pháp phổ biến thường được dùng:

- Phương pháp quy nạp.
- Cây quyết định và luật.
- Phát hiện luật kết hợp.
- Các phương pháp phân lớp và hồi phi tuyến tính.
- Gom cụm.
- Các phương pháp dựa trên mẫu.
- KTDL văn bản.
- Mạng nơron.
- Thuật toán di truyền.
- Mô hình phụ thuộc dựa trên đồ thị xác suất.
- Mô hình học quan hệ.

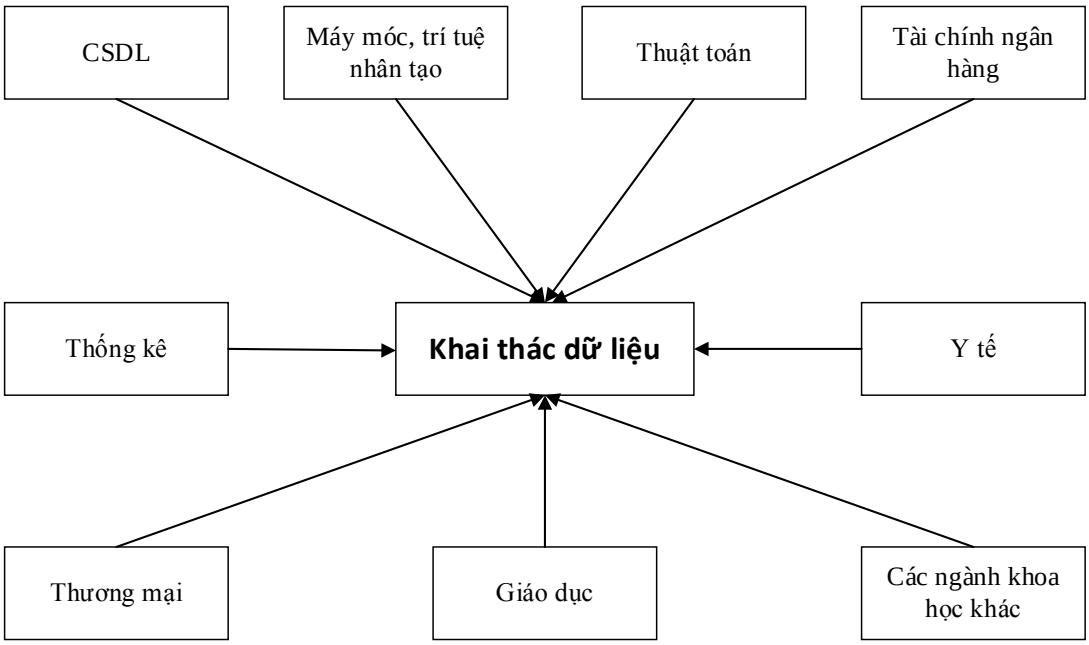
Các thuật toán KTDL tự động vẫn mới chỉ ở giai đoạn phát triển ban đầu. Người ta vẫn chưa đưa ra được một tiêu chuẩn nào trong việc quyết định sử dụng phương pháp nào và trong trường hợp nào thì có hiệu quả.

Để so sánh các kỹ thuật cần phải có một tập lớn các quy tắc và các phương pháp thực nghiệm tốt. Thường thì quy tắc này không được sử dụng khi đánh giá các kỹ

thuật mới nhất. Vì vậy mà những yêu cầu cải thiện độ chính xác không phải lúc nào cũng thực hiện được.

1.1.5. Ứng dụng của khai thác dữ liệu

KTDL là một lĩnh vực liên quan tới nhiều ngành học khác như: Hệ CSDL, thống kê, trực quan hoá,v.v. (hình 1.3). Hơn nữa, tuỳ vào cách tiếp cận được sử dụng, KTDL còn có thể áp dụng một số kỹ thuật như mạng nơron, lý thuyết tập thô, tập mờ, biểu diễn tri thức, v.v.



Hình 1.3. Một số lĩnh vực liên quan đến khai thác dữ liệu

Phát hiện tri thức và KTDL được coi là trung tâm của nhiều ngành khoa học, nó liên quan đến rất nhiều ngành, nhiều lĩnh vực khác nhau như tài chính, ngân hàng, thương mại, y tế, giáo dục, thống kê, máy móc, trí tuệ nhân tạo, CSDL, thuật toán học, tính toán song song, thu nhận tri thức trong các hệ chuyên gia, quan sát dữ liệu.

Lĩnh vực học máy và nhận dạng mẫu là giống nhau trong KTDL nghiên cứu các lý thuyết và thuật toán của hệ thống trích ra các mẫu và mô hình dữ liệu. KTDL tập trung vào việc mở rộng các lý thuyết và thuật toán cho các vấn đề về tìm ra các mẫu

đặc biệt, đây được coi là những mẫu hữu ích hoặc tri thức quan trọng tập dữ liệu lớn.

Đặc biệt, phát hiện tri thức và KTDL rất gần gũi với lĩnh vực thống kê, sử dụng các phương pháp thống kê để mô hình hoá dữ liệu và phát hiện các mẫu, luật,… kho dữ liệu và các công cụ xử lý trực tuyến tập trung vào phân tích dữ liệu đa chiều, tốt hơn ngôn ngữ truy vấn CSDL trong tính toán và phân tích thống kê đa chiều cũng liên quan chặt chẽ đến KTDL.

Đặc trưng của hệ thống KTDL là nhờ vào các phương pháp thuật toán và kỹ thuật từ những lĩnh vực khác nhau, nhằm mục đích cuối cùng là trích ra tri thức từ dữ liệu trong CSDL khổng lồ.

KTDL hiện đang được áp dụng một cách rộng rãi trong nhiều lĩnh vực kinh doanh và thu được những lợi ích to lớn. Một số lĩnh vực được áp dụng rộng rãi như:

- Phân tích dữ liệu và hỗ trợ ra quyết định.
- Điều trị trong y học: mối liên hệ giữa triệu chứng, chuẩn đoán và phương pháp điều trị (chế độ dinh dưỡng, dược phẩm).
- Phân lớp văn bản, tóm tắt văn bản và phân lớp các trang Web.
- Tin sinh học: Tìm kiếm, đối sánh các hệ gen và thông tin di truyền, mối liên hệ giữa một số hệ gen và một số bệnh di truyền.
- Nhận dạng.
- Tài chính và thị trường chứng khoán: Phân tích tình hình tài chính và dự đoán giá cổ phiếu.
- Bảo hiểm.
- Giáo dục.

1.1.6. Một số khó khăn trong việc khai thác dữ liệu

Cơ sở dữ liệu lớn: Các tập dữ liệu cần xử lý trong KTDL thường có kích thước cực kỳ lớn về cả số lượng các bản ghi và số lượng các thuộc tính. Trong thực tế, kích thước của các tập dữ liệu trong KTDL thường ở mức tera-byte. Với kích thước như thế, thời gian xử lý thường cực kỳ dài. Mặc dù kích thước bộ nhớ trong của

máy tính đã gia tăng đáng kể trong thời gian gần đây, việc gia tăng này cũng không thể đáp ứng kịp với việc tăng kích thước dữ liệu. Vì vậy, việc vận dụng các kỹ thuật xác suất, lấy mẫu, đệm, song song…vào các giải thuật để tạo ra các phiên bản phù hợp với yêu cầu của KTDL trở nên ngày càng quan trọng.

Dữ liệu thiếu và nhiễu: Mức độ nhiễu cao trong dữ liệu điều này dẫn đến việc dự đoán thiếu chính xác.

Vấn đề "quá phù hợp": Khi thuật toán khai thác tìm kiếm với các tham số tốt nhất cho một mô hình đặc biệt và một giới hạn của tập dữ liệu. Mô hình đó có thể "Quá phù hợp" trên tập dữ liệu đó nhưng lại thi hành không chính xác trên tập dữ liệu kiểm tra.

Sự thay đổi của dữ liệu và tri thức: Dữ liệu là không tĩnh, dữ liệu thay đổi nhanh chóng có thể dẫn đến những tri thức đã khai thác trước đây trở nên không còn phù hợp thậm chí là vô giá trị.

Đánh giá các mẫu dữ liệu tìm được: Nhiều mẫu phát hiện không thực sự hữu ích với người sử dụng và thách thức với các hệ KTDL.

Làm việc với các dữ liệu quan hệ phức tạp: Do các hệ cơ sở dữ liệu quan hệ được sử dụng rộng rãi nên vấn đề làm tốt với các hệ cơ sở dữ liệu này là vấn đề cần quan tâm đối với các hệ KTDL.

Khai thác thông tin trong các hệ cơ sở dữ liệu hỗn hợp và hệ thống thông tin toàn cầu: Với sự ra đời của mạng máy tính, dữ liệu có thể được thu thập từ nhiều nguồn khác nhau với định dạng khác nhau với số lượng rất lớn. Việc phát hiện tri thức từ các dạng dữ liệu hỗn hợp này là một thách thức đối với KTDL.

Vì lượng thông tin cần khai thác ngày càng lớn và phức tạp, các thuật toán khai thác tuần tự đã không đáp ứng được nhu cầu khai thác thông tin ngày càng tăng. Các giải pháp tính toán song song có thể xử lý lượng lớn tính toán trên các dữ liệu khổng lồ, vì vậy tính toán song song là công cụ tốt để KTDL.

1.2. Tổng quan về khai thác dữ liệu song song

Hầu hết các thuật toán KTDL đều đòi hỏi số lượng tính toán lớn, chúng cần có khả năng mở rộng được để xử lý lượng dữ liệu lớn hơn và phức tạp hơn. Một số CSDL liên quan tới Internet nên đã sẵn có tính phân tán. Các hệ thống tính toán song song với các bộ đa xử lý tốc độ cao, ghép các hệ sẵn có với nhiều cách khác nhau bằng các thiết bị kết nối đã được nghiên cứu và phát triển để KTDL. Phần này sẽ trình bày về cấu trúc hệ thống song song, các loại hệ thống song song và một số chiến lược KTDL song song.

1.2.1. Cấu trúc hệ thống song song

1.2.1.1. Định nghĩa xử lý song song

Xử lý song song là quá trình xử lý gồm nhiều tiến trình được kích hoạt đồng thời và cùng giải quyết một bài toán. Nói chung, xử lý song song được thực hiện trên những hệ thống đa bộ xử lý.

1.2.1.2. Phân biệt xử lý song song và xử lý tuần tự

Trong tính toán tuần tự với một bộ xử lý thì tại mỗi thời điểm chỉ được thực hiện một phép toán.

Trong tính toán song song thì nhiều bộ xử lý cùng kết hợp với nhau để giải quyết cùng một bài toán cho nên giảm được thời gian xử lý vì mỗi thời điểm có thể thực hiện đồng thời nhiều phép toán.

1.2.1.3. Mục đích của xử lý song song

Là thực hiện tính toán nhanh trên cơ sở sử dụng nhiều bộ xử lý đồng thời. Cùng với tốc độ xử lý nhanh, việc xử lý song song cũng sẽ giải được những bài toán phức tạp yêu cầu khối lượng tính toán lớn.

1.2.1.4. Ba yếu tố chính dẫn đến việc xây dựng các hệ thống xử lý song song

Tốc độ xử lý của các bộ xử lý theo kiểu Von Neumann đã dần tiến tới giới hạn, không thể cải tiến thêm được, do vậy dẫn tới đòi hỏi phải thực hiện xử lý song song.

Hiện nay giá thành của bộ xử lý (CPU) giảm mạnh, tạo điều kiện để xây dựng những hệ thống có nhiều bộ xử lý với giá thành hợp lý.

Sự phát triển của công nghệ mạch tích hợp (VLSI–very large scale integration) cho phép tạo ra những hệ thống có hàng triệu transistor trên một chip.

Hệ thống tính song song: Là một tập các bộ xử lý (thường cùng một loại) kết nối với nhau theo một kiến trúc nào đó để có thể hợp tác với nhau trong hoạt động và trao đổi dữ liệu được với nhau.

Vấn đề xử lý song song: Liên quan trực tiếp đến kiến trúc máy tính, phần mềm hệ thống (hệ điều hành), thuật toán và ngôn ngữ lập trình, v.v…

Độ phức tạp của xử lý song song sẽ lớn hơn xử lý tuần tự rất nhiều, và tập trung chủ yếu ở phương diện trao đổi dữ liệu và đồng bộ các tiến trình.

1.2.2. Phân loại các kiến trúc song song

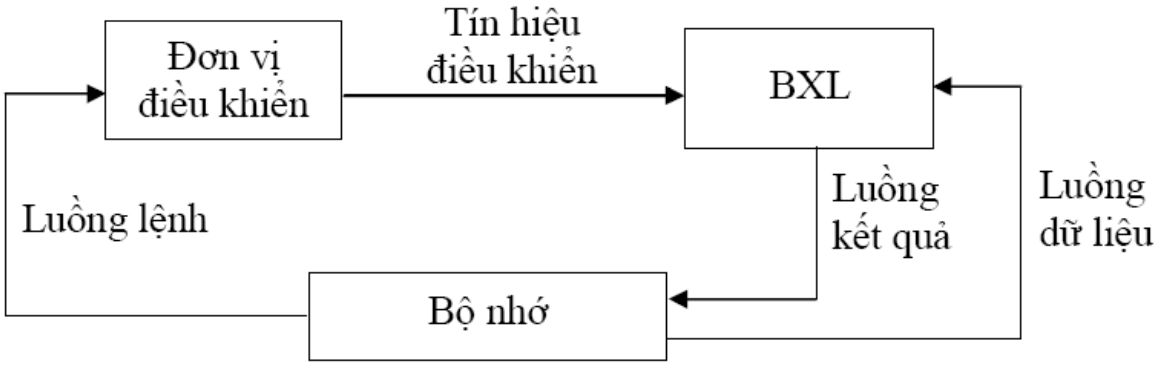
Một trong những phân loại kiến trúc máy tính song song được biết đến nhiều nhất là phân loại của Flynn, được sử dụng từ năm 1966. Michael Flynn dựa vào đặc tính về số lượng bộ xử lý, số chương trình thực hiện, cấu trúc bộ nhớ, v.v… để phân máy tính thành bốn loại dựa trên sự biểu hiện của cặp khái niệm: dòng lệnh và dòng dữ liệu, mỗi loại nằm trong một trong hai trạng thái đơn hoặc đa [2] được thể hiện trong Bảng 1.1:

Bảng 1.1. Mô tả phân loại kiến trúc của Flynn

Dòng lệnh (instruction stream)	**Dòng dữ liệu (data stream)**	**Loại kiến trúc**
Trạng thái đơn (single)	Trạng thái đơn (single)	SISD Single Instruction Single Data
Trạng thái đơn (single)	Trạng thái đa (multiple)	SIMD Single Instruction Multiple Data
Trạng thái đa (multiple)	Trạng thái đơn (single)	MISD Multiple Instruction Single Data
Trạng thái đa (multiple)	Trạng thái đa (multiple)	MIMD Multiple Instruction Multiple Data

1.2.2.1. Kiến trúc đơn dòng lệnh đơn luồng dữ liệu (SISD)

Máy tính SISD chỉ có một CPU, ở mỗi thời điểm thực hiện một luồng lệnh và chỉ đọc, ghi một mục dữ liệu. Tất cả các máy tính SISD chỉ có một thanh ghi (register) được gọi là bộ đếm chương trình, được sử dụng để nạp địa chỉ của lệnh tiếp theo và kết quả là thực hiện theo một thứ tự xác định của các câu lệnh. Kiến trúc máy SISD có mô hình hoạt động theo Hình 1.4:

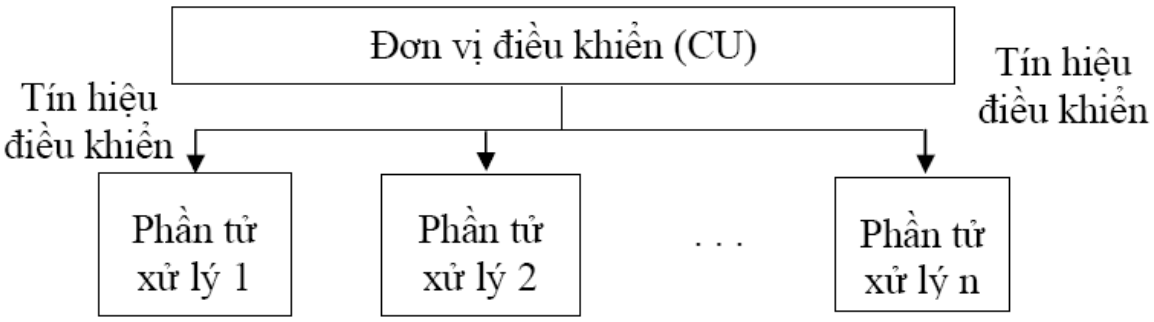


Hình 1.4. Mô hình kiến trúc máy SISD

1.2.2.2. Kiến trúc đơn dòng lệnh đa luồng dữ liệu (SIMD)

Máy tính SIMD có một đơn vị điều khiển để điều khiển nhiều đơn vị xử lý thực hiện theo một luồng các câu lệnh. CPU phát sinh tín hiệu điều khiển tới tất cả các

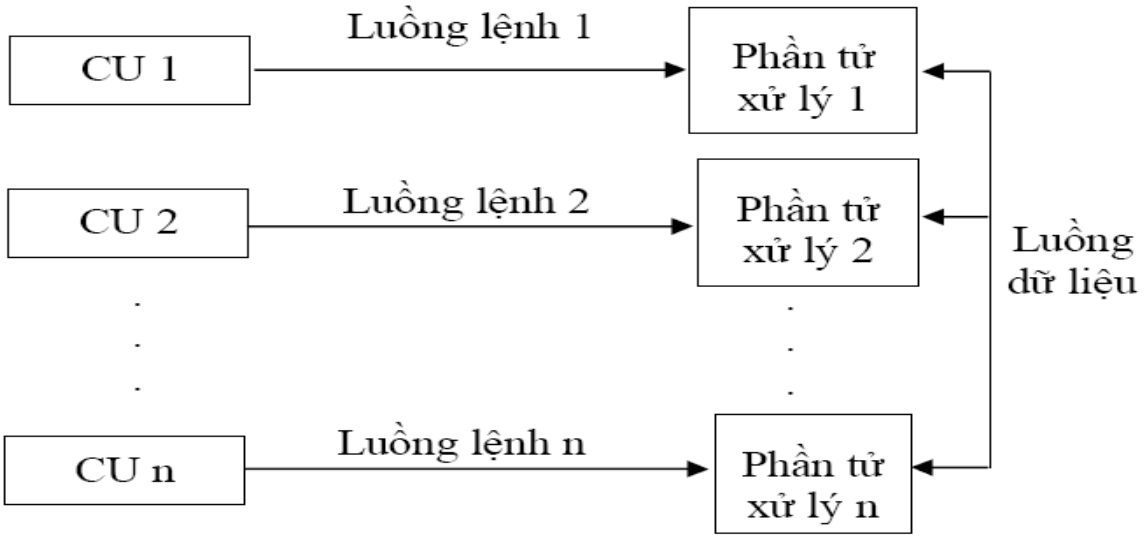
phần xử lý, những bộ xử lý này *cùng thực hiện một phép toán trên các mục dữ liệu khác nhau*, nghĩa là mỗi bộ xử lý có luồng dữ liệu riêng. Kiến trúc máy SIMD có mô hình hoạt động theo Hình 1.5:



Hình 1.5. Mô hình kiến trúc máy SIMD

1.2.2.3. Kiến trúc đa dòng lệnh đơn dữ liệu (MISD)

Máy tính loại MISD có thể thực hiện nhiều chương trình (nhiều lệnh) trên cùng một mục dữ liệu (ngược với máy tính loại SIMD). Kiến trúc máy MISD có mô hình hoạt động mô tả theo Hình 1.6:



Hình 1.6. Mô hình kiến trúc máy MISD

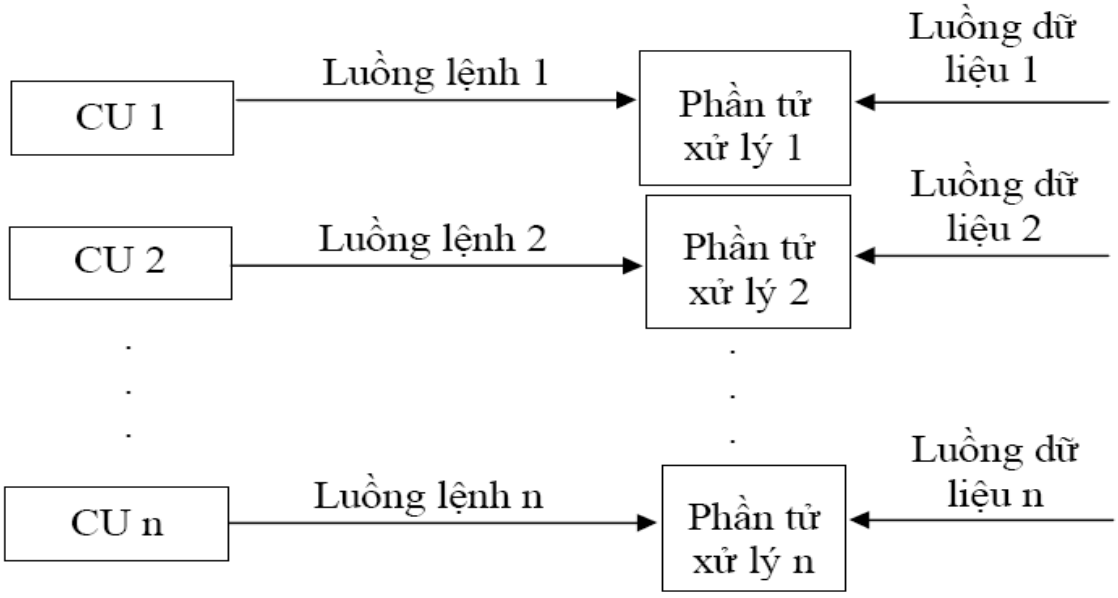
Kiến trúc kiểu này có thể chia thành hai nhóm:

- Các máy tính yêu cầu mỗi đơn vị xử lý (PU) nhận những lệnh khác nhau để thực hiện trên cùng một mục dữ liệu.

- Các máy tính có các luồng dữ liệu được chuyển tuần tự theo dãy các CPU liên tiếp-gọi là kiến trúc hình ống-xử lý theo vector thông qua một dãy các bước, trong đó mỗi bước thực hiện một chức năng và sau đó chuyển kết quả cho CPU thực hiện bước tiếp theo.

1.2.2.4. Kiến trúc đa dòng lệnh đa luồng dữ liệu (MIMD)

Máy tính loại MIMD gọi là đa bộ xử lý, trong đó mỗi bộ xử lý có thể thực hiện những luồng lệnh (chương trình) khác nhau trên các luồng dữ liệu riêng. Hầu hết các hệ thống MIMD đều có bộ nhớ riêng và cũng có thể truy cập vào bộ nhớ chung khi cần, do vậy giảm thiểu được thời gian trao đổi dữ liệu giữa các bộ xử lý trong hệ thống. Đây là loại kiến trúc phức tạp nhất, nhưng nó là mô hình hỗ trợ xử lý song song cao nhất và đã có nhiều máy tính được thiết kế theo kiến trúc này, ví dụ: BBN Butterfly, Alliant FX, iSPC của Intel… Kiến trúc máy MIMD có mô hình hoạt động theo Hình 1.7:



Hình 1.7. Mô hình kiến trúc máy MIMD

1.2.3. Các chiến lược khai thác dữ liệu song song

Khai thác dữ liệu song song đòi hỏi phân chia công việc để các bộ xử lý có thể thực hiện phần công việc của mình, nhằm đạt kết quả cuối cùng một cách nhanh nhất. Vấn đề là ở chỗ phân chia công việc như thế nào, ta có thể phân chia việc tính toán, cũng có thể phân chia việc truy cập tới dữ liệu, và giảm thiểu sự truyền thông giữa các bộ xử lý trong khi thực hiện. Trong các ứng dụng KTDL, ta cần giảm thiểu nguồn tài nguyên được dùng để sinh các khái niệm có vẻ có giá trị địa phương, dựa trên lượng hạn chế dữ liệu có sẵn tại mỗi bộ xử lý, nhưng không có giá trị toàn phần. Có ba chiến lược để song song hóa các thuật toán KTDL, đó là:

Tìm kiếm độc lập: Mỗi bộ xử lý được truy cập tới toàn bộ tập dữ liệu, nhưng chỉ tập trung vào một phần không gian tìm kiếm, bắt đầu từ một điểm được chọn ngẫu nhiên.

Cách này phù hợp với các bài toán mà kết quả cần tìm là một giải pháp tối ưu, tuy nhiên nó đòi hỏi mỗi bộ xử lý phải truy cập toàn bộ tập dữ liệu, khiến tốc độ bị chậm lại.

Song song hóa một thuật toán khai thác dữ liệu tuần tự: Tập các khái niệm được phân chia giữa các bộ xử lý, mỗi bộ xử lý kiểm tra toàn bộ tập dữ liệu để kiểm tra xem các khái niệm cục bộ của nó có đúng trên phạm vi toàn cục không. Do việc tạo khái niệm mới thường đòi hỏi phải biết các khái niệm nhỏ hơn hay đơn giản hơn nào là đúng, các bộ xử lý phải thường xuyên trao đổi thông tin về các khái niệm của chúng. Một cách khác là tập dữ liệu được phân chia theo các cột, mỗi bộ xử lý tìm ra các khái niệm trên các cột mà chúng giữ.

Theo cách này cũng cần thường xuyên trao đổi thông tin để xác định xem các khái niệm cục bộ nào có thể ghép lại thành khái niệm đúng trên toàn cục.

Lặp lại một thuật toán khai thác dữ liệu tuần tự: Mỗi bộ xử lý làm việc trên một phần của tập dữ liệu (theo hàng), và thực hiện thuật toán tuần tự. Do chỉ có một phần thông tin, nó tạo nên các khái niệm đúng cục bộ, nhưng có thể không đúng trên toàn cục - các khái niệm xấp xỉ. Các bộ xử lý trao đổi các khái niệm xấp xỉ này, hoặc các số liệu về chúng, để kiểm tra xem chúng có đúng trên toàn cục không. Khi

làm như vậy mỗi bộ xử lý học được về những phần dữ liệu mà chúng không nhìn thấy.

Cách này có hai ưu điểm đáng chú ý: tập dữ liệu được phân chia giúp cho chi phí truy cập được chia đều cho các bộ xử lý, và dữ liệu cần được trao đổi giữa các pha thường nhỏ hơn rất nhiều so với bản thân các khái niệm, vì thế không tốn nhiều chi phí truyền dữ liệu.

Để thuận lợi cho việc khai thác và quản lý thông tin trong CSDL, các CSDL thường được sắp xếp lại sao cho phù hợp. Nhất là trong các CSDL giao dịch được sử dụng nhiều trong KTDL, việc sắp xếp lại dữ liệu nhằm tối ưu hóa CSDL để thuật toán đạt hiệu quả cao hơn là rất quan trọng. Phần tiếp theo sẽ trình bày các khái niệm cơ bản về cơ sở dữ liệu giao dịch, tập phổ biến, các tính chất của tập phổ biến…

1.3. Một số khái niệm về cơ sở dữ liệu giao dịch và tập phổ biến

1.3.1. Cơ sở dữ liệu giao dịch

Cơ sở dữ liệu giao dịch (*database transaction*) là đơn vị tương tác của một CSDL hoặc các hệ tương tự, mỗi giao dịch được xử lý một cách nhất quán và tin cậy mà không phụ thuộc vào các giao dịch khác.

Một bảng của CSDL giao dịch gồm:

- Các cột là các thuốc tính hay còn gọi là các mục dữ liệu.
- Các hàng là các giao tác được thực hiện.

Ví dụ về bảng dữ liệu của một CSDL giao dịch:

Cho một CSDL mẫu với số lượng giao dịch là 6 Tập mục {A, C, D, T, W}

Bảng 1.2. Cơ sở dữ liệu mẫu

Mã giao dịch	Nội dung giao dịch
1	A, C, T, W
2	C, D, W
3	A, C, T, W
4	A, C, D, W
5	A, C, D, T, W
6	C, D, T

1.3.2. Khái niệm về tập phổ biến

Luật kết hợp được giới thiệu đầu tiên vào năm 1993, được sử dụng để xác định các mối quan hệ giữa các tập thuộc tính trong CSDL. Việc xác định các quan hệ này không được dựa vào các đặc tính dữ liệu vốn có của chúng (như các phụ thuộc hàm), mà nên dựa vào sự xuất hiện cùng lúc của các thuộc tính dữ liệu.

Bài toán khai thác luật kết hợp trên một CSDL được chia thành hai giai đoạn. Giai đoạn thứ nhất là tìm tất cả các tập mục dữ liệu có độ phổ biến thỏa ngưỡng tối thiểu cho trước, gọi là các tập phổ biến. Giai đoạn thứ hai là tìm ra những luật kết hợp từ những tập phổ biến thỏa độ tin cậy tối thiểu cho trước. Hầu hết các nghiên cứu đều tập trung giải quyết bài toán tìm tập phổ biến vì bài toán tìm luật kết hợp khá đơn giản.

Bài toán phát hiện tập phổ biến có thể được phát biểu như sau:

Cho một tập các mục I, một CSDL giao dịch D, độ phổ biến tối thiểu *minsup*, ngưỡng tin cậy *minconf*. Tìm tất cả các luật kết hợp X ⇒Y trên CSDL D sao cho: *sup(X ⇒ Y) ≥ minsup và conf(X ⇒ Y) ≥ minconf*. Bài toán khai thác luật kết hợp có thể được chia ra làm 2 bài toán con được phát biểu trong thuật toán sau:

- Đầu vào: *I, D, minsup, minconf*
- Đầu ra: Các luận kết hợp thỏa mãn *minsup* và *minconf*

Phương thức:

- Tìm tất cả các tập phổ biến từ CSDL D tức là tìm tất cả các tập mục có độ hỗ trợ lớn hơn hoặc bằng *minsup*.
- Sinh ra các luật từ các tập phổ biến sao cho độ tin cậy của luật lớn hơn hoặc bằng *minconf*.

Bước 1: Tìm các tập mục phổ biến.

Bước 2: Sinh các luật kết hợp từ tập mục phổ biến tìm được ở bước 1.

Tập phổ biến là cơ sở để tạo các luật kết hợp. Bài toán khai thác tập phổ biến (frequent itemset) là bài toán rất quan trọng trong lĩnh vực KTDL. Bài toán khai thác tập phổ biến là bài toán tìm tất cả tập mục (itemset) S có độ phổ biến (support) thỏa mãn độ phổ biến tối thiểu minSup:

$$Sup(s) \geq minSup$$

Định nghĩa độ phổ biến:

Cho CSDL bao gồm:

- Tập các mục I, tập các giao dịch D và tập mục X ⊆ I
- Độ phổ biến của X trong D - ký hiệu là sup(X) - được định nghĩa là số giao dịch mà X xuất hiện trong D.

Ví dụ: Với CSDL mẫu trong Bảng 1.2, ta có: Tập mục I = {A, C, D, T, W} và tập giao dịch D gồm có 6 giao dịch: {ACTW, CDW, ACTW, ACDW, ACDTW, CDT}

Độ phổ biến của tập mục X1 = {A} là số giao dịch trong D có chứa {A}, do đó sup(X_1) = 4. Tương tự, X_2 = {A, T} => độ phổ biến của là sup(X_2) = 3

Định nghĩa tập phổ biến:

Tập X ⊆ I được gọi là tập phổ biến nếu sup(X) ≥ minSup, với minSup là giá trị do người dùng chỉ định.

Ví dụ: Cũng với CSDL mẫu trong Bảng 1.2, với minSup = 3 (50%) thì tập X_2 = {A, T} là tập phổ biến vì có sup (X_2) = 3 ≥ minSup.

Tương tự, với X_3 = {A, C, T} thì sup (X_3) = 3 ≥ minSup và X_3 cũng là tập phổ biến. Ngược lại, với X_4 = {C, D, T} thì sup (X_4) = 2 < minSup, do đó X_4 không phải là tập phổ biến.

1.3.3. Các tính chất của tập phổ biến

Tính chất 1:

Độ phổ biến của tập con lớn hơn tập cha.

Cho hai tập phổ biến X, Y với X ⊂ Y thì sup(X) ≥ sup(Y)

Tính chất 2:

Mọi tập con của một tập phổ biến đều là tập phổ biến.

X là tập phổ biến và Y ⊂ X thì sup(Y) ≥ sup(X) ≥ minSup, vì vậy Y cũng là tập phổ biến.

Tính chất 3:

Mọi tập cha của một tập không phổ biến thì cũng không phổ biến.

X là tập không phổ biến và Y ⊃ X thì sup(Y) ≤ sup(X) < minSup, vì vậy Y cũng không phải là tập phổ biến.

1.3.4. Một số phương pháp biểu diễn cơ sở dữ liệu trong khai thác dữ liệu

Trong các CSDL quan hệ thông thường sẽ lưu trữ dữ liệu theo chiều ngang. Tức là các bảng dữ liệu hai chiều sẽ gồm N dòng tương ứng với các giao dịch, và M cột tương ứng với các mục. Việc bố trí theo chiều ngang giúp cho việc xác định các mục thuộc về một giao dịch đơn giản nhanh chóng. Tuy nhiên khi cần xác định một mục cụ thể thuộc vào những giao dịch nào thì cách bố trí theo chiều ngang lại gây ra khó khăn, khi đó ta phải duyệt tất cả các giao dịch có trong CSDL và ghi nhận những giao dịch có chứa mục cụ thể đó.

1.3.4.1. Biểu diễn cơ sở dữ liệu theo chiều ngang

Hầu hết các CSDL giao dịch được thiết kế theo chiều ngang. Trong đó, tất cả các dữ liệu được sắp xếp theo hàng khi có giao dịch xảy ra. Một bảng của CSDL được biểu diễn dưới dạng chuỗi các trang có chứa một hoặc nhiều dòng.

CSDL theo chiều ngang có nhiều thuận lợi cho việc lưu trữ dữ liệu với số lượng hàng và cột được định trước. Trong mô hình CSDL theo chiều ngang, thay vì tăng cường khả năng lưu trữ của mỗi thành phần trong hệ thống, nhiều thành phần được tăng thêm cho hệ thống. Vì vậy khả năng mở rộng và quản lý bộ nhớ của CSDL theo chiều ngang là rất tốt.

Ví dụ: Cho CSDL mẫu như bảng 1.2.

CSDL mẫu trong Bảng 1.2 được sắp xếp theo chiều ngang như sau:

Bảng 1.3. Bảng dữ liệu sắp xếp theo chiều ngang

Mã các giao dịch (TIDs)	**Mã mục (items)**				
	A	C	D	T	W
T1	1	1	0	1	1
T2	0	1	1	0	1
T3	1	1	0	1	1
T4	1	1	1	0	1
T5	1	1	1	1	1
T6	0	1	1	1	0

1.3.4.2. Biểu diễn cơ sở dữ liệu theo chiều dọc

Với các tổ chức dữ liệu theo chiều dọc, một CSDL gồm một danh sách các mục. Mỗi mục xác định một danh sách các định danh của giao dịch có chứa mục đó, ký hiệu tid-List. Những ưu điểm của cách tổ chức theo chiều dọc:

- Nếu tid-List đã được sắp theo thứ tự tăng dần thì độ hỗ trợ của k-itemset ứng viên có thể đã được tính toán bởi phép lấy giao của các tid-List của hai (k-1)-subset bất kỳ, với cách tổ chức này, thuật toán không cần phải duy trì cấu trúc dữ liệu phức tạp, không như cây băm và cũng không phải sinh ra tất cả các k-subset của các giao dịch hoặc thực hiện các thao tác tìm kiếm trên cây băm.
- Các tid-List chứa tất cả các thông tin liên quan về một tập mục, vì vậy, khi tính độ hỗ trợ cho một tập mục không cần phải quét toàn bộ CSDL. Vì tất cả các thông tin về một lớp tương đương là được nhóm cùng nhau nên có thể sinh ra các tập mục phổ biến trước khi chuyển sang lớp tiếp theo.

Trong CSDL mẫu ở Bảng 1.2, muốn biết mục A nằm trong những giao dịch nào thì ta phải duyệt hết tất cả các mục xem giao dịch nào chứa A thì liệt kê ra, kết quả các giao dịch chứa A là tập {1, 3, 4, 5} chứa một mục là rất tốn kém. Trong lúc đó việc xác định những giao dịch có chứa một mục cụ thể lại là công việc thường xuyên để tính độ phổ biến của một tập mục. Để tăng tốc độ khai thác tìm tập phổ biến, chúng ta có thể sử dụng cách sắp xếp dữ liệu theo chiều dọc. Nghĩa là bảng dữ liệu có sự đảo chiều, các dòng biến thành các cột và ngược lại:

*Ví dụ***:** CSDL mẫu trong Bảng 1.2 được sắp xếp theo chiều dọc như sau:

Bảng 1.4. Bảng dữ liệu sắp xếp theo chiều dọc

Mã mục (items)	**Mã các giao dịch (TIDs)**					
C	1	2	3	4	5	6
W	1	2	3	4	5	
A	1	3	4	5		
D	2	4	5	6		
T	1	3	5	6		

Lúc này việc xác định xem mục A có độ phổ biến bao nhiêu trong CSDL sắp xếp theo chiều dọc trở nên đơn giản bằng cách xác định dòng trong bảng tương ứng với mã mục A, kết quả là tập giao dịch {1, 3, 4, 5} và sup(A) = 4

Tuy nhiên cách sắp xếp dữ liệu theo chiều dọc sẽ gây khó khăn trong việc quản lý bộ nhớ vì bảng dữ liệu gia tăng kích thước theo chiều ngang thay vì theo chiều dọc.

Cả 2 cách sắp xếp dữ liệu theo chiều ngang và chiều dọc đều có điểm mạnh và điểm yếu riêng. Do đó, tùy theo yêu cầu thực tế mà người ta có cách sắp xếp dữ liệu phù hợp với mục đích sử dụng riêng.

Kết luận chương 1:

Trong chương 1, luận văn đã trình bày về: mục tiêu, kiến trúc, các phương pháp KTDL cũng như ứng dụng của KTDL vào thực tế. Các cấu trúc hệ thống song song, phân loại các kiến trúc song song và các chiến lược KTDL song song cũng được đề cập trong chương này.

Trong chương này cũng nêu ra một số định nghĩa, tính chất liên quan đến khai thác tập phổ biến để tìm luật kết hợp như khái niệm độ phổ biến, độ tin cậy, tập phổ biến, luật kết hợp. Trong chương tiếp theo, luận văn sẽ trình bày một số thuật toán cơ bản khai thác tập phổ biến và các nhận xét, đánh giá ưu nhược điểm của các thuật toán này.

CHƯƠNG 2. MỘT SỐ PHƯƠNG PHÁP KHAI THÁC TẬP PHỔ BIẾN

2.1. Thuật toán Apriori

2.1.1. Ý tưởng thuật toán

Phương pháp sinh ứng viên để tìm tập phổ biến được Agrawal và các đồng sự đề xuất từ năm 1993 với thuật toán Apriori. Ý tưởng của thuật toán Apriori dựa trên kết luận: *nếu một tập mục là phổ biến thì tất cả tập con của nó cũng phải phổ biến (tính chất 2–tập phổ biến).* Do vậy không thể có trường hợp một tập phổ biến có tập con là không phổ biến hay nói cách khác tập phổ biến nhiều mục hơn chỉ có thể được tạo ra từ các tập phổ biến ít mục hơn. Và đó là cách hoạt động của thuật toán Apriori:

- Bắt đầu từ các tập phổ biến chỉ có một mục.
- Tạo ra các tập phổ biến có k mục từ những tập phổ biến có (k-1) mục.

2.1.2. Nội dung thuật toán:

Thuật toán Apriori có nội dung như sau:

$\mathbf{L_k}$: Tập hợp của các tập phổ biến k (k-itemset). Mỗi phần tử của tập hợp này có hai trường: tập mục và độ phổ biến.

$\mathbf{C_k}$: Tập hợp của các tập ứng viên k. Mỗi phần tử của tập hợp này có hai trường: tập mục và độ phổ biến.

```
L1 = {tập hợp 1 mục};
For (k = 2; Lk-1 ≠ ∅; k++)
Begin
   Ck = apriori-gen (Lk-1); // Tạo ứng viên mới.
   For (all giao tác t ∈ D) // duyệt CSDL
   Begin
```

```
        Ct = subset(Ck, t); //các tập mục ứng viên có trong
    //giao tác t
        Forall ứng viên c ∈ Ct do
        c. count ++;
    end
    Lk = {c ∈ Ck | c.count ≥ minsup}
end
Answer = ∪k Lk; // Trả về tập hợp của các tập phổ biến
```

Diễn giải thuật toán:

- Bước đầu tiên của thuật toán đơn giản chỉ tính các mục xuất hiện để xác định tập hợp của các tập phổ biến có 1 mục.
- Lặp bước k:

  + Tập hợp L_{k-1} được sử dụng để tạo nên tập ứng viên C_k: sử dụng hàm **apriori-gen** được miêu tả là hàm lấy L_{k-1} (*tập hợp của các tập phổ biến k-1 mục*) là đầu vào và trả về C_k là tập hợp của tất cả các tập chứa k mục phát sinh từ tập hợp L_{k-1} bằng cách hợp các tập k-1 mục trong tập hợp L_{k-1}. Chú ý một ứng viên thuộc C_k thì tất cả các tập con của ứng viên đó phải có mặt trong L_{k-1} (*theo tính chất 2 của tập phổ biến*)

  + Bước kế tiếp, duyệt CSDL để tính độ phổ biến của các ứng viên trong tập hợp C_k. Từ đó, tính được tập hợp L_k.

  + Nếu $L_k = \emptyset$ thì dừng lại.

- Hợp của các L_k chính là các tập phổ biến cần tìm.

  Ví dụ minh họa: Xét CSDL mẫu trong Bảng 1.2 với minSup = 50% * 6 = 3

  Bước 1: Duyệt CSDL để tìm L_1 là các tập phổ biến có 1 mục có độ phổ biến ≥ 3:

Bảng 2.1. Các tập phổ biến có 1 mục

Database (D)	
TID	Nội dung
1	A, C, T, W
2	C, D, W
3	A, C, T, W
4	A, C, D, W
5	A, C, D, T, W

L_1	
Mục	Độ phổ biến
A	4
C	6
D	4
T	4
W	5

Bước 2: tính tập ứng viên C_2 dựa trên L_1 bằng phép hợp. Sau đó duyệt CSDL tính độ phổ biến của các ứng viên thuộc C_2, loại bỏ những ứng viên có độ phổ biến bé hơn minSup, để tạo thành L_2 là các tập phổ biến có 2 mục.

Bảng 2.2. Các tập phổ biến có 2 mục

C	
Mục	Độ phổ biến
AC	4
AD	2
AT	3
AW	4
CT	4
CD	4
CT	2
CW	5
DW	3
TW	3

L_2	
Mục	Độ phổ biến
AC	4
AT	3
AW	4
CT	4
CD	4
CW	5
DW	3
TW	3

Bước 3: Tương tự bước 2 tạo C_3 và L_3 từ L_2**.** Chú ý khi tạo C_3 thì loại bỏ những ứng viên có tập con 2 mục không nằm trong L_2. Từ L_2 để tạo C_3 có thể bằng cách từ

tập hợp tất cả mục có trong L_2, rồi lần lượt kiểm tra các tập mục 3 thuộc tính trong tập hợp đó theo chú ý ở dòng trên.

Bảng 2.3. Các tập phổ biến có 3 mục

C3	
Mục	Độ phổ biến
ACT	3
ACW	4
ATW	3
CDW	3
CTW	3

L3	
Mục	Độ phổ biến
ACT	3
ACW	4
ATW	3
CDW	3
CTW	3

Bước 4: Tạo C_4 và L_4 từ L_3

Bảng 2.4. Các tập phổ biến có 4 mục

C4	
Mục	Độ phổ biến
ACTW	3

L4	
Mục	Độ phổ biến
ACTW	3

Bước 5: Tạo C_5 và L_5 từ L_4. Ta có $C_5 = \emptyset$.

Như vậy thuật toán dừng lại ở bước 5 vì $L_5 = \emptyset$

Kết quả: với minSup = 50% thì:

Tập hợp các tập phổ biến $L_1 \cup L_2 \cup L_3 \cup L_4$ = {A, C, D, T, W, AC, AT, AW, CD, CT, CW, DW, TW, ACT, ACW, ATW, CDW, CTW, ACTW}

2.1.3. Nhận xét thuật toán Apriori

Thuật toán Apriori với n là độ dài lớn nhất của tập được sinh ra. Vậy thuật toán sẽ thực hiện duyệt toàn bộ các giao tác n+1 lần. Như vậy, nếu bỏ qua thời gian so sánh tìm sự xuất hiện của một mục trong một giao tác thì độ phức tạp của thuật toán Apriori là:

$$O(A) > O(n \times L)$$

Trong đó L là kích thước của CSDL còn n là độ dài cần đạt được của các mục.

Hạn chế chính của thuật toán Apriori là:

Chi phí cho số lượng khổng lồ các tập ứng viên. Ví dụ: nếu có 10^4 tập 1-mục phổ biến thì thuật toán Apriori cần phải sinh ra hơn 10^7 các ứng viên 2-mục và thực hiện kiểm tra sự xuất hiện của chúng. Hơn nữa để khám phá được một số tập phổ biến có kích thước (độ dài) là 1, thuật toán phải kiểm tra $(2^1 - 2)$ các tập phổ biến tiềm năng.

Đòi hỏi lặp lại nhiều lần duyệt CSDL để kiểm tra rất lớn số lượng tập ứng viên. Số lần duyệt CSDL của thuật toán Apriori bằng độ dài của tập phổ biến dài nhất tìm được. Trong trường hợp tập phổ biến dài hơn và CSDL rất lớn, có nhiều bản ghi, điều này là không khả thi. Thuật toán Apriori chỉ thích hợp cho các CSDL thưa (spare), với các CSDL dày (dense) thì thuật toán kém hiệu quả hơn rất nhiều.

Để xác định độ phổ biến của các tập ứng viên, thuật toán Apriori phải quét lại toàn bộ giao dịch trong CSDL, do đó sẽ tiêu tốn rất nhiều thời gian, đặc biệt khi số mục lớn. Ngoài ra, khi độ phổ biến tối thiểu minsup bị thay đổi thì thuật toán phải thực hiện lại từ đầu nên rất mất thời gian. Như vậy, nhìn chung các phương pháp sinh ứng viên để tìm tập phổ biến đều không hiệu quả vì phải đọc CSDL nhiều lần và phải phát sinh và kiểm tra một lượng lớn các ứng viên.

Trong số các thuật toán khai thác tập phổ biến, các thuật toán được cài đặt dựa trên thuật toán Apriori được sử dụng phổ biến bởi vì thực thi chúng đơn giản và dễ dàng. Thuật toán được xây dựng nhằm phát hiện các luật kết hợp giữa các đối tượng với độ phổ biến và độ tin cậy tối thiểu.

2.2. Thuật toán Eclat

2.2.1. Ý tưởng thuật toán.

Thuật toán Eclat dùng phương pháp để nhóm các tập mục phổ biến có liên quan với nhau bằng cách sử dụng lược đồ phân chia lớp tương đương. Mỗi lớp tương đương chứa tập các mục ứng viên quan hệ tương đương với nhau. Bên cạnh đó, thuật toán cũng sử dụng kỹ thuật tổ chức CSDL theo chiều dọc để nhóm các giao dịch có liên quan với nhau [3].

Phân lớp tương đương: Gọi L_k là tập các itemset phổ biến. Không mất tính tổng quát, giả sử L_k được sắp xếp theo thứ tự từ điển. Ta có thể phân hoạch các tập mục trong L_k thành các lớp tương đương như sau: Nếu các phần tử trong L_k có k–1 thành viên đầu tiên giống nhau thì chúng thuộc cùng một lớp.

Ký hiệu: Lớp tương đương chứa a là Sa = [a].

Trong phạm vi một lớp, ta sinh k-itemset ứng viên bằng cách kết nối tất cả $\left(\frac{Si}{2}\right) = |S_i|(|S_i| - 1)/2$ cặp tiền tố là định danh của lớp. Trong đó: |Si| là số phần tử của lớp có định danh là i. Các k-itemset ứng viên được sinh ra từ các lớp khác nhau sẽ độc lập với nhau.

Tổ chức dữ liệu: Thuật toán tổ chức CSDL theo chiều dọc. Với cách tổ chức này, một CSDL gồm danh sách các mục và mỗi mục xác định một danh sách các định danh của giao dịch có chứa mục đó, kí hiệu là tid-List. Ưu điểm của cách tổ chức CSDL theo chiều dọc như sau:

- Nếu tid-List được sắp xếp theo thứ tự tăng dần thì độ phổ biến của k-tập mục (k-itemset) ứng viên có thể đã được tính toán bởi phép lấy giao các tid-List của 2(k-1)-tập mục hỗ trợ (subset) bất kỳ.
- Các danh sách của định danh của giao dịch tid-List chứa tất cả các thông tin liên quan về một tập mục. Vì vậy khi tính độ hỗ trợ của một tập mục không cần quét toàn bộ CSDL.

2.2.2. Nội dung thuật toán Eclat

Begin

/ Giai đoạn khởi tạo*/*

1) Duyệt qua các phân hoạch CSDL cục bộ

2) Tính toán số đếm hỗ trợ cục bộ cho tất cả các 2-itemset

3) Xây dựng số đếm hỗ trợ tổng thể cho các tập mục chứa trong L_2.

*/*Giai đoạn biến đổi*/*

4) Phân hoạch L_2 thành các lớp tương đương

5) Lập lịch L_2 trên tập các bộ xử lý

6) Tổ chức phân hoạch dữ liệu cục bộ theo chiều dọc

7) Truyền các tid-List có liên quan tới các bộ xử lý khác

8) L_2 cục bộ = nhận các tid-List từ các bộ xử lý khác

*/*Giai đoạn đồng thời*/*

9) for parallel mỗi lớp tương E_2 trong L_2 cục bộ
Compute_Frequent (E_2)

*/*Giai đoạn rút gọn*/*

10) Tập hợp các kết quả đưa ra luật kết hợp

end.

Giải thích thuật toán

1) Giai đoạn khởi tạo

Giai đoạn khởi tạo bao gồm việc tính toán tất cả các 2-itemset phổ biến trong CSDL cần khai thác. Ta không cần tính số đếm hỗ trợ của các 1-itemset vì việc xác định số đếm hỗ trợ của 2-itemset có thể đạt được chỉ trong một lần duyệt CSDL. Để tính toán cho các 2-itemset, trên mỗi bộ xử lý sử dụng một mảng cục bộ và tiến hành chỉ số hóa các mục trong CSDL theo cả hai chiều. Mặt khác, mỗi bộ xử lý tính số đếm hỗ trợ cục bộ cho các 2-itemset và thực hiện phép lấy tổng rút gọn (sum-reduction) của tất cả các bộ xử lý để xây dựng các số đếm hỗ trợ tổng thể. Kết thúc giai đoạn khởi tạo, tất cả các bộ xử lý đều có những số đếm hỗ trợ tổng thể của tất cả các 2-itemset phổ biến L_2 trong CSDL.

2) Giai đoạn biến đổi gồm 2 bước

Bước 1: Đầu tiên L2 được phân hoạch thành các lớp tương đương. Sau đó các lớp tương đương này được gán cho các bộ xử lý sao cho cân bằng nhau.

Bước 2: CSDL đã được biến đổi từ định dạng theo chiều ngang thành chiều dọc và được phân phối lại. Do đó, trong bộ nhớ cục bộ của mỗi bộ xử lý, các tid-List của tất cả các 2-itemset trong một lớp tương đương bất kỳ được nó gán cho nó.

Lập lịch phân lớp tương đương

Đầu tiên, ta phân hoạch L2 thành các lớp tương đương bằng cách sử dụng tiền tố chung như mô tả ở trên. Tiếp theo, phân chia cho mỗi bộ xử lý một lớp tương đương. Mỗi lớp tương đương được gán một trọng số dựa vào số các phần tử trong một lớp. Vì phải khảo sát tất cả các cặp trong bước lặp tiếp theo, nên ta gán trọng số $(\frac{m}{2})$ cho 1 lớp với m là số phần tử của lớp tương đương tương ứng.

Sắp xếp các lớp dựa theo các trọng số và lần lượt gán cho bộ xử lý đã nạp ít nhất, nghĩa là bộ xử lý đó có trọng số toàn phần của các lớp nhỏ nhất. Nếu ước lượng tốt được số các tập mục phổ biến mà có thể nhận được từ một lớp tương đương thì có thể sử dụng ước lượng này làm một trọng số. Trong phạm vi một lớp, cũng có thể lấy độ hỗ trợ trung bình của các tập mục làm trọng số.

Biến đổi CSDL theo chiều dọc

Sau khi phân hoạch các lớp tương đương cân bằng giữa các bộ xử lý, ta biến đổi CSDL cục bộ từ định dạng theo chiều ngang theo chiều dọc. Điều này có thể thực hiện được trong 2 bước:

Bước 1: Mỗi bộ xử lý duyệt CSDL cục bộ của nó và xây dựng các tid-List cục bộ cho tất cả các 2-itemset.

Bước 2: Mỗi bộ xử lý cần xây dựng các tid-List toàn cục cho các tập mục trong các lớp tương đương của nó. Do đó, nó phải gửi các tid -List này cho các bộ xử lý khác và nhận các tid-List từ các bộ xử lý khác gửi đến.

3) Giai đoạn đồng thời

Cơ sở dữ liệu đã được phân bố lại, vì vậy các tid-List của tất cả các 2-itemset trong các lớp tương đương cục bộ của nó là đã thường trú trên đĩa cục bộ. Mỗi bộ xử lý có thể tính toán tất cả các tập mục phổ biến một cáh độc lập. Nó đọc trực tiếp từ bộ nhớ cục bộ các tid-List của các 2-itemset, sau đó sinh tất cả các tập mục phổ biến có thể trước khi chuyển sang bước tiếp theo, bước này bao gồm việc quét phân hoạch CSDL cục bộ đã được biến đổi một lần.

Trong phạm vi mỗi lớp tương đương, cần khảo sát tất cả các cặp 2-itemset và thực hiện lấy giao các tid-List tương ứng. Nếu số các phần tử của tid-List kết quả lớn hơn hoặc bằng độ hỗ trợ tối thiểu thì tập mục mới được bổ sung vào L3. Sau đó, tiếp tục phân hoạch L3 thành các lớp tương đương dựa trên các tiền tố chung độ dài bằng 2. Quá trình này được lặp lại thủ tục được thực hiện như sau:

```
Begin Compute_Frequent(Ek-1)
    for tất cả các itemset I1 và I2 trong Ek-1
     if ((I1.tidList ∩ I2tidList) ≥ minsup)
     Bổ sung (I1 ∪ I2) vào Lk;
     Phân hoạch Lk thành các lớp tương đương;
     For parallel mỗi lớp tương đương Ek trong Lk
     Compute_Frequent(Ek);
End Compute_Frequent
```

4) Giai đoạn rút gọn

Tại thời điểm cuối của giai đoạn đồng thời, chúng ta trích rút các kết quả từ mỗi bộ xử lý và đưa ra kết quả. Quá trình thực hiện các bước truyền thông khác nhau của thuật toán.

- Giai đoạn khởi tạo: Khi đã thu được các số đếm hỗ trợ của tất cả các 2-itemset, ta cần thực hiện một phép lấy tổng rút gọn để tính số đếm tổng thể.

- Giai đoạn biến đổi: Mỗi bộ xử lý quét phân hoạch CSDL cục bộ của nó lần thứ hai và xây dựng các tid-List theo chiều dọc đối với tất cả các 2-itemset phổ biến L_2.

  Quá trình biến đổi được hoàn thành qua 2 bước sau:

  Bước 1: Biến đổi tid-List cục bộ.

Trước tiên, ta chia L_2 thành hai nhóm. Các tập mục thuộc các lớp tương đương mà được gán cho bộ xử lý cục bộ, kí hiệu là G, các tập mục còn lại thuộc các lớp tương đương khác, kí hiệu là R. Với mỗi bộ xử lý P_i, bộ nhớ dành ra một vùng nhớ có kích thước:

$$\sum_{g \in G} global\_count(g) + \sum_{r \in R} partial\_count(r, P_i)$$

Với $g \in G$, r $\in R$ là tập các mục

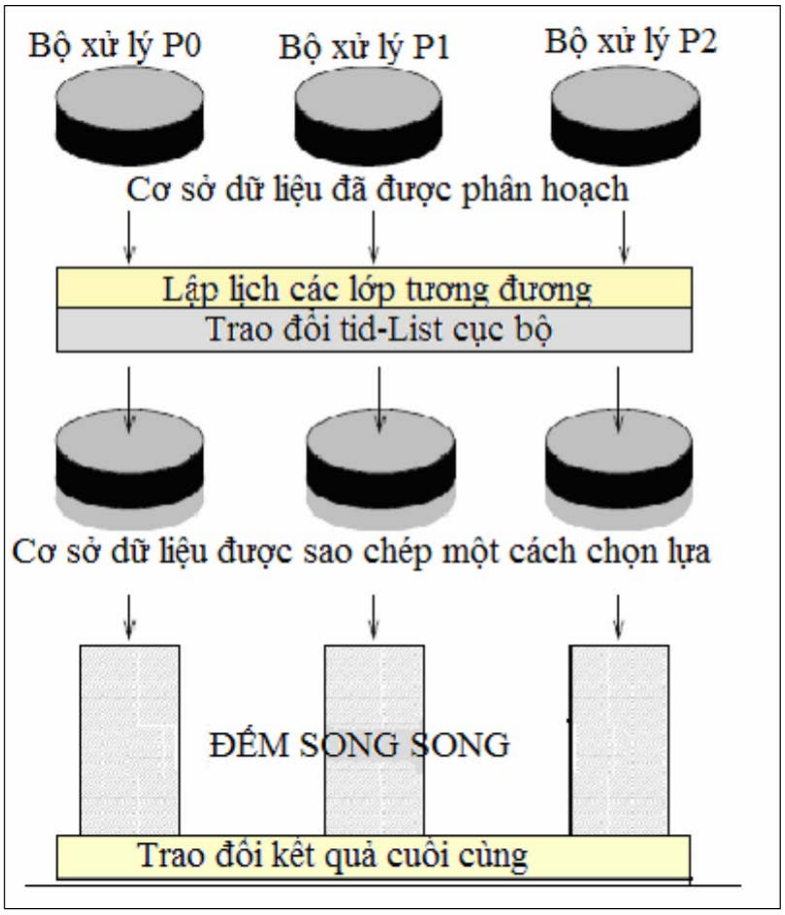
$partial_{count(r,P_i)}$: Số đếm hỗ trợ cục bộ của tập mục r trên bộ xử lí P_i

Tiếp theo, mỗi bộ xử lí thực hiện việc biến đổi và ghi tid-List của các phần tử G vào các khoảng trống thích hợp. Các phần tử của R được để trống.

Bước 2: Truyền tid-List

Một khi việc biến đổi CSDL cục bộ hoàn thành, ta cần phải nhận các tid-List của tất cả các 2-itemset trong G từ các bộ xử lý khác truyền đến và truyền các tid-List của R đến các bộ xử lý khác. Các tid-List đến được sao chép vào các khoảng trống thích hợp. Vì các phần giao dịch là phân biệt tăng đều, các tid-List của các tập mục trong G đã được viết ra ngoài đĩa, còn trong R thì bị loại bỏ. Để truyền các tid-List cục bộ qua kênh bộ nhớ, chúng ta sử dụng lợi thế của việc truyền thông điệp nhanh ở mức người sử dụng. Mỗi bộ xử lý xác định kích thước bộ đệm (2MB) cho một vùng truyền, một vùng nhận và dùng chung một định danh. Việc truyền thông được tiến hành theo cách khóa luân phiên các giai đoạn ghi đọc. Trong giai đoạn ghi, mỗi bộ xử lý ghi các tid-List của các tập mục trong P vào vùng truyền của nó cho đến khi đạt đến giới hạn không gian bộ đệm. Tại thời điểm này, nó đi vào giai đoạn đọc, nó lần lượt quét vùng nhận của mỗi bộ xử lý và đặt các tid-List này của G vào các khoảng trống thích hợp. Khi vùng đọc đã được quét xong, nó đi vào giai

đoạn ghi. Quá trình này được lặp lại cho đến khi nhận được tất cả các tid-List bộ phận. Tại thời điểm cuối của giai đoạn này, CSDL được định dạng theo chiều dọc. Thuật toán Eclat được tổng kết ở hình 2.1.



Hình 2.1. Thuật toán Eclat [3]

Sau đó, mỗi bộ xử lý đi vào giai đoạn đồng thời và tính toán các tập mục phổ biến như mô tả ở trên. Việc phép rút gọn cuối cùng được thực hiện tương tự như phép rút gọn trong giai đoạn khởi tạo.

2.2.3. Nhận xét thuật toán Eclat

Thuật toán Eclat dùng phương pháp nhóm tập phổ biến có liên quan với nhau bằng cách sử dụng lược đồ phân chia lớp tương đương, với mỗi lớp tương đương chứa các tập mục ứng viên quan hệ tương đương với nhau. Phương pháp này sử

dụng kỹ thuật tổ chức CSDL theo chiều dọc để nhóm các giao dịch có liên quan với nhau. Giải thuật này quét CSDL ít hơn, giảm thiểu chi phí vào và ra. Thuật toán có ưu điểm là tính toán nhanh độ phổ biến thông qua tập giao dịch tid-List.

Hạn chế chủ yếu của thuật toán này là chúng cần phải sinh ra và phân bố lại các tid-List. Hơn nữa, với một tập mục phổ biến có kích thước lớn, các phần chung chủ yếu của các tid-List này được lấy giao lặp lại nhiều lần đối với tất cả các tập con của nó. Để giảm bớt tình trạng này, một cách thiết lập tối ưu khác là chỉ kiểm tra những thay đổi trong tid-List thay cho việc lưu giữ các tid -List toàn cục thông qua các vòng lặp sao cho nó giảm đáng kể khối lượng dữ liệu được tính toán.

2.3. Thuật toán FP-Growth

2.3.1. Ý tưởng thuật toán

Thuật toán Apriori tìm tập phổ biến thực hiện bằng cách rút gọn kích thước các tập ứng viên nhờ kỹ thuật tỉa. Tuy nhiên trong tình huống số lượng các mục nhiều, độ phổ biến tối thiểu thấp, thuật toán Apriori gặp phải 2 chi phí lớn như đã nêu ở phần nhận xét thuật toán Apriori, đó là:

- Chi phí cho số lượng khổng lồ các tập ứng viên.
- Đòi hỏi lặp lại nhiều lần duyệt CSDL để kiểm tra rất lớn số lượng tập ứng viên.

Trong bài báo [11] các tác giả J.H.J.Pei và Y.Jin đã giới thiệu thuật toán FP-growth nhằm khắc phục nhược điểm nêu trên. Thuật toán tìm các tập phổ biến bằng phương pháp khác không sinh các ứng viên. Thuật toán FP-growth được phát triển theo 3 kỹ thuật chính:

- Mở rộng cấu trúc cây FP-tree dùng để nén dữ liệu thích hợp, chỉ có các mục độ dài 1(1-item) ở trong cây và các nút của cây được sắp đặt để các nút xuất hiện thường xuyên hơn có thể dễ dàng chia sẻ với các nút xuất hiện ít hơn. CSDL lớn được nén chặt tới cấu trúc dữ liệu nhỏ hơn (FP-tree) tránh được chi phí lặp lại việc duyệt CSDL.
- Phương pháp khai thác phát triển (growth) từng đoạn dựa trên FP-tree được thực hiện. Bắt đầu từ tập phổ biến độ dài 1, FP-growth chỉ xem xét cơ sở mục phụ

thuộc của nó như là CSDL con, bao gồm các tập mục phổ biến cùng xuất hiện với các mẫu hậu tố (suffix pattern), xây dựng FP-tree tương ứng với nó và thực hiện khai thác đệ quy trên cây này. Khai thác theo cách này có thể tránh chi phí cho việc sinh ra số lượng lớn các tập ứng viên.

- Kỹ thuật tìm kiếm được dùng ở đây là chia để trị nhằm phân rã nhiệm vụ khai thác thành tập nhiệm vụ nhỏ hơn với giới hạn các mục trong các CSDL nhằm thu gọn không gian tìm kiếm.

2.3.2. Cấu trúc cây FP – Tree

Cấu trúc FP-Tree (Frequent Pattern tree) được giới thiệu lần đầu tiên bởi các tác giả J.Han, J.Pei và Y.Yin đã khắc phục được nhược điểm của thuật toán Apriori là phải phát sinh và kiểm tra một lượng lớn các ứng viên.

Cấu trúc FP-tree được dùng để tổ chức lại CSDL cho thuận lợi hơn trong quá trình tìm tập phổ biến, đồng thời các thông tin được nén trong cây FP-tree với tỉ lệ tương đối cao. Những thuận lợi đó là:

Những mục không đủ độ phổ biến được loại ngay từ đầu, vì vậy việc tìm tập phổ biến chỉ thao tác trên một số lượng mục nhỏ hơn nhiều so với toàn bộ các mục.

Nhiều giao dịch sẽ được nén chung trong cây FP-tree và việc này giúp giảm bớt khá nhiều thao tác trong quá trình xác định độ phổ biến của tập mục.

Cấu trúc FP-tree cho phép thực hiện tìm kiếm theo chiều sâu và áp dụng mô hình chia để trị khá hiệu quả.

Cây FP-tree là cấu trúc cây với một số đặc điểm sau:

- Có một nút cha được đánh nhãn NULL, những nút con nối với nút cha là những thành phần chung của nhiều giao dịch được nén lại với nhau (item prefix subtree), bên cạnh cũng có một mảng các mục đơn phổ biến (frequent-item header table).
- Mỗi nút trong item prefix subtree có ba trường dữ liệu: mã mục, số tích lũy và con trỏ liên kết. Mã mục tương ứng mục mà nút này đại diện, số tích lũy là số giao dịch có chứa chung phần mục này, con trỏ liên kết dùng để liên kết 2 nút

đại diện chung một mã mục ở hai item prefix subtree khác nhau. Giá trị con trỏ liên kết mang giá trị rỗng khi là nút cuối cùng trong chuỗi liên kết.

- Mỗi phần tử trong một mảng các mục đơn phổ biến gồm 2 trường: mã mục và con trỏ liên kết đến đầu nút của chuỗi liên kết các nút cùng đại diện chung cho một mục.

2.3.3. Phép chiếu trên cây FP-tree:

Sau khi xây dựng cấu trúc FP-tree cho toàn bộ CSDL chỉ gồm những mục đơn thỏa độ phổ biến tối thiểu minSup, chúng ta phải duyệt cây để tìm ra những tập phổ biến thỏa minSup. Hiệu quả của quá trình khai thác phụ thuộc nhiều vào phương pháp duyệt. Phương pháp duyệt phải thỏa những yêu cầu:

- Đảm bảo kết quả tập phổ biến là đầy đủ.
- Kết quả các tập phổ biến không bị trùng lặp.
- Những tập phổ biến tạo ra thỏa ngưỡng minSup.

Để duyệt cây FP-tree, ta có thể sử dụng một trong hai phép chiếu dưới đây:

Phép chiếu từ dưới lên:

- Dựa trên thứ tự của f-list, chọn mục hạt giống bắt đầu từ mục có độ phổ biến nhỏ nhất thỏa minSup cho đến mục có độ phổ biến lớn nhất.
- Trên cây FP-tree, duyệt từ những nút chứa mục hạt giống tiến dần đến nút gốc để xây dựng f-list cục bộ và FP-tree cục bộ của mục hạt giống.
- Nếu duyệt hết mục trong f-list thì quay lui một bước và thực hiện tiếp.

Phép chiếu từ trên xuống:

- Dựa trên thứ tự của f-list, chọn mục hạt giống bắt đầu từ mục có độ phổ biến lớn nhất cho đến mục có độ phổ biến nhỏ nhất thỏa minSup.
- Trên cây FP-tree, duyệt từ nút chứa mục hạt giống tiến dần xuống nút lá của cây và xây dựng f-list cục bộ của mục hạt giống. Ghi nhận vị trí của nút con trực tiếp của những nút chứa mục hạt giống trong f-list cục bộ.
- Nếu f-list cục bộ không còn mục nào thì quay lui một bước và thực hiện tiếp.

2.3.4. Nội dung thuật toán FP-Growth:

Thuật toán FP-Growth có thể được mô tả như sau:

Đầu vào: CSDL các giao dịch và độ phổ biến tối thiểu minSup

Đầu ra: Tập các tập phổ biến FI thỏa độ phổ biến tối thiểu minSup

Begin

Bước 1: Tree$_0$ = createFPtree (CSDL, minSup)

Bước 2: FP-growth (Tree$_0$, null, minSup) **Hàm FP-growth (FP-tree, prefix, minSup). Các bước thực hiện:**

Bước 1: Nếu FP-tree chỉ có một đường đơn P thì chỉ cần tạo ra những tập phổ biến kết hợp giữa prefix và các tổ hợp của những mục trong P và độ phổ biến bằng với giá trị tích lũy nhỏ nhất của những nút tham gia vào tổ hợp. Sau khi phát sinh xong thì kết thúc hàm.

Bước 2: Ngược lại, lần lượt xét từng phần tử a trong f-list của FP-tree và phát sinh tập phổ biến (prefix $\cup$ a) có độ phổ biến bằng sup(a).

Bước 2.1: Duyệt cây FP-tree từ chuỗi các nút đại diện cho mục a, bắt đầu bằng con trỏ liên kết của phần tử a trong f-list và hướng lên nút gốc. Sau khi duyệt xong ta có được CSDL các giao dịch chiếu trên tập mục (prefix $\cup$ a), ký hiệu là **CSDL {prefix $\cup$ a}**

Bước 2.2: Tree **{prefix $\cup$ a}** = createFPtree (CSDL **{prefix $\cup$ a}**, minSup)

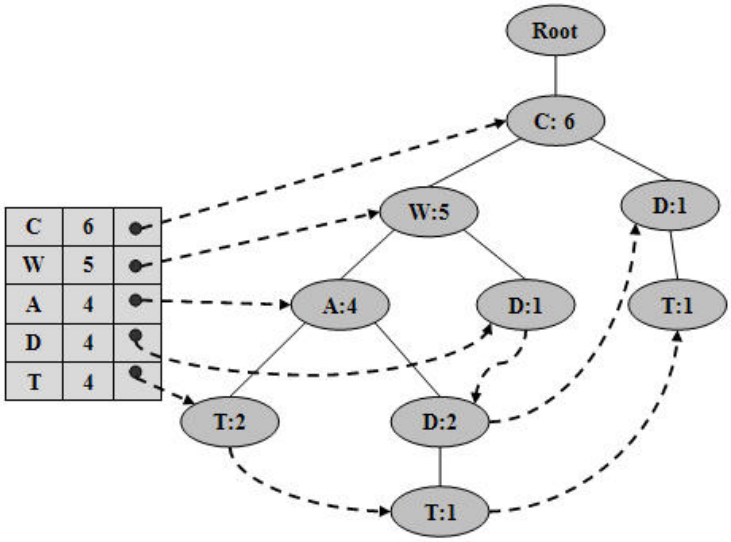
Bước 2.3: Nếu Tree **{prefix $\cup$ a} $\neq \emptyset$** thì gọi hàm:

FP-growth (Tree **{prefix $\cup$ a}**, (prefix $\cup$ a), minSup)

End

Ví dụ: Thực hiện thuật toán FP-growth để tìm các tập phổ biến trong CSDL mẫu ở Bảng 1.2 với độ phổ biến là minSup = 2. Sau khi tạo được Tree$_0$, gọi hàm:

FP-growth (Tree$_0$, null, minSup)



Hình 2.2. Cây $Tree_0$

Vì $Tree_0$ không phải đường đơn, nên xét mục (T) trong f-list và phát sinh tập phổ biến (T:4). Sau đó chiếu trên $Tree_0$ để tạo ra $CSDL_{\{T\}}$:

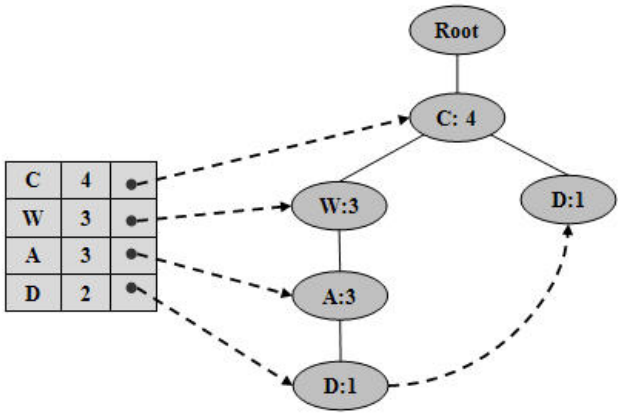
Chiếu trên tập mục ***(T:4)***

Cơ sở dữ liệu cục bộ của các giao dịch có chứa tập mục {**T**}

Bảng 2.5. Nội dung $CSDL_{\{T\}}$

Nội dung còn lại của các giao dịch có chứa tập mục {T}	**Giá trị tích lũy**
C, W, A	2
C, W, A, D	1
C, D	1

Tạo cây FP-tree cục bộ tương ứng với $CSDL_{\{T\}}$:



Hình 2.3. $Tree_{\{T\}}$ cục bộ tương ứng với $CSDL_{\{T\}}$

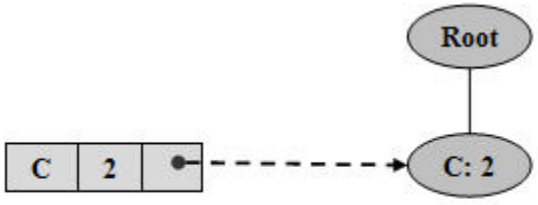
Vì $Tree_{\{T\}} \neq \emptyset$ nên gọi đệ quy chiều sâu hàm FP-growth ($Tree_{\{T\}}$, (T), minSup)

Vì $Tree_{\{T\}}$ không phải đường đơn, nên xét mục (D) trong f-list và phát sinh tập phổ biến (TD: 2).

Chiếu trên tập mục (TD:2):

Cơ sở dữ liệu cục bộ của các giao dịch chứa tập mục {T, D}

Tạo cây FP-tree cục bộ tương ứng với $CSDL_{\{TD\}}$:



Hình 2.4. $Tree_{\{TD\}}$ cục bộ tương ứng với $CSDL_{\{TD\}}$

Vì $Tree_{\{TD\}} \neq \emptyset$ nên gọi đệ quy hàm FP-growth ($Tree_{\{TD\}}$, (TD) ,minSup)

Vì $Tree_{\{TD\}}$ là đường đơn nên phát sinh tập phổ biến (TDC : 2).

Quay trở lại với $Tree_{\{T\}}$ xét tiếp mục (A) trong f-list của $Tree_{\{T\}}$ và phát sinh tập phổ biến (TA:3).

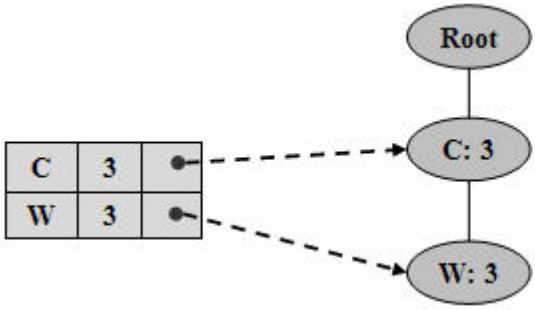
Chiếu trên tập mục (TA:3):

Cơ sở dữ liệu cục bộ của các giao dịch có chứa tập mục {T, A}

Bảng 2.6. Nội dung $CSDL_{\{TA\}}$

Nội dung còn lại của các giao dịch có chứa tập mục {T, A}	**Giá trị tích lũy**
C, W	3

Tạo cây FP-tree cục bộ tương ứng với $CSDL_{\{TA\}}$:



Hình 2.5. $Tree_{\{TA\}}$ cục bộ tương ứng với $CSDL_{\{TA\}}$

Gọi hàm đệ quy FP-growth ($Tree_{\{TA\}}$,(TA) ,minSup) và phát sinh được các tập phổ biến (TAC: 3) ; (TAW:3) ; (TACW:3).

Quay trở lại với $Tree_{\{T\}}$ xét tiếp mục (W) trong f-list của $Tree_{\{T\}}$ và phát sinh tập phổ biến (TW:3).

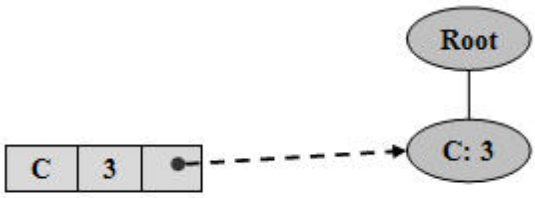
Chiếu trên tập mục (TW:3):

Cơ sở dữ liệu cục bộ của các giao dịch có chứa tập mục {T, W}

Bảng 2.7. Nội dung $CSDL_{\{TW\}}$

Nội dung còn lại của các giao dịch có chứa tập mục {T, W}	**Giá trị tích lũy**
C	3

Tạo cây FP-tree cục bộ tương ứng với $CSDL_{\{TW\}}$:



Hình 2.6. $Tree_{\{TW\}}$ cục bộ tương ứng với $CSDL_{\{TW\}}$

Gọi hàm đệ quy FP-growth ($Tree_{\{TW\}}$,(TW) ,minSup) và phát sinh được các tập phổ biến (TWC:3).

Quay trở lại với $Tree_{\{T\}}$ xét tiếp mục (C) trong f-list của $Tree_{\{T\}}$ và phát sinh tập phổ biến (TC :4).

Chiếu trên tập mục (TC:4):

$CSDL_{\{TC\}} = \emptyset$ dẫn đến $Tree_{\{TC\}} = \emptyset$

Quay trở lại với $Tree_0$ xét tiếp mục (D) trong f-list của $Tree_0$ và phát sinh tập phổ biến (D:4).

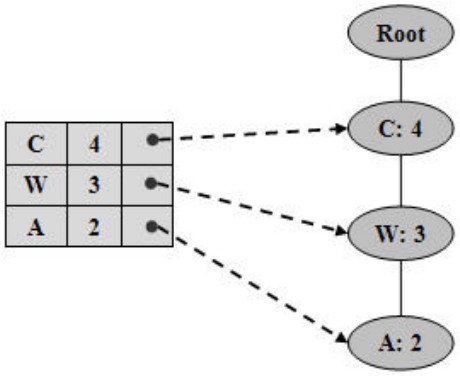
Chiếu trên tập mục (D:4):

Ta có cơ sở dữ liệu cục bộ của các giao dịch có chứa tập mục {D}

Bảng 2.8. Nội dung $CSDL_{\{D\}}$

Nội dung còn lại của các giao dịch có chứa tập mục { D }	Giá trị tích lũy
C, W, A	2
C, W	1
C	1

Tạo cây FP-tree cục bộ tương ứng với $CSDL_{\{D\}}$:



Hình 2.7. $Tree_{\{D\}}$ cục bộ tương ứng với $CSDL_{\{D\}}$

Gọi hàm đệ quy FP-growth ($Tree_{\{D\}}$,(D) ,minSup) và phát sinh được các tập phổ biến: (DA:2); (DW:3); (DC:4); (DAW:2); (DAC:2); (DWC:3); (DAWC:2).

Quay trở lại với $Tree_0$ xét tiếp mục (A) trong f-list của $Tree_0$ và phát sinh tập phổ biến (A:4).

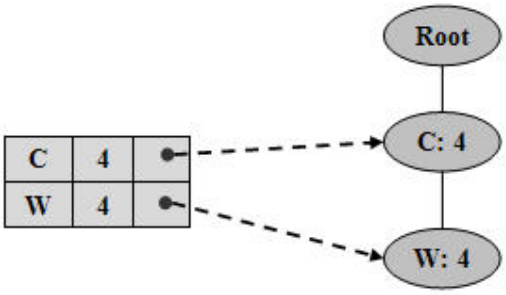
Chiếu trên tập mục (A:4) :

Ta có cơ sở dữ liệu cục bộ của các giao dịch có chứa tập mục {A}

Bảng 2.9. Nội dung $CSDL_{\{A\}}$

Nội dung còn lại của các giao dịch có chứa tập mục {A}	Giá trị tích lũy
C, W	4

Tạo cây FP-tree cục bộ tương ứng với $CSDL_{\{A\}}$:



Hình 2.8. $Tree_{\{A\}}$ cục bộ tương ứng với $CSDL_{\{A\}}$

Gọi hàm đệ quy FP-growth ($Tree_{\{A\}}$, (A), minSup) và phát sinh được các tập phổ biến: (AW : 4) ; (AC :4) ; (AWC : 4)

Quay trở lại với $Tree_0$ xét tiếp mục (W) trong f-list của $Tree_0$ và phát sinh tập phổ biến (W:5).

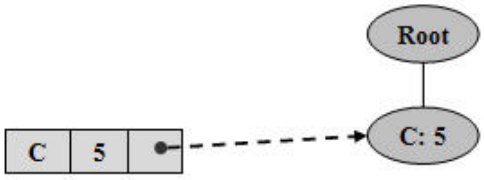
Chiếu trên tập mục (W:5):

Ta có cơ sở dữ liệu cục bộ của các giao dịch có chứa tập mục {W}

Bảng 2.10. Nội dung $CSDL_{\{W\}}$

Nội dung còn lại của các giao dịch có chứa tập mục {W}	Giá trị tích lũy
C	5

Tạo cây FP-tree cục bộ tương ứng với $CSDL_{\{W\}}$:



Hình 2.9. $Tree_{\{W\}}$ cục bộ tương ứng với $CSDL_{\{W\}}$

Gọi hàm đệ quy FP-growth ($Tree_{\{W\}}$,(W) ,minSup) và phát sinh được tập phổ biến: (WC: 5).

Quay trở lại với $Tree_0$ xét tiếp mục (C) trong f-list của $Tree_0$ và phát sinh tập phổ biến (C:6).

Chiếu trên tập mục (C:6) :

Cơ sở dữ liệu cục bộ của các giao dịch có chứa tập mục {C}: $CSDL_{C\}} = \emptyset$

Cấu trúc cây FP-tree cục bộ tương ứng với $CSDL_{\{C\}}$: $Tree_{\{C\}} = \emptyset$

Kết quả tập phổ biến thu được khi thực hiện thuật toán FP-growth:

Bảng 2.11. Kết quả tập phổ biến thỏa ngưỡng minSup = 2

STT	Tập phổ biến	Độ phổ biến (số giao dịch)	Độ phổ biến (%)
1.	T	4	66.67%
2.	T, D	2	33.33%
3.	T, D, C	2	33.33%
4.	T, A	3	50.00%
5.	T, A, W	3	50.00%
6.	T, A, C	3	50.00%
7.	T, A, W, C	3	50.00%
8.	T, W	3	50.00%
9.	T, W, C	3	50.00%
10.	T, C	4	66.67%
11.	D	4	66.67%
12.	D, A	2	33.33%
13.	D, W	3	50.00%
14.	D, W, A	2	33.33%
15.	D, C	4	66.67%
16.	D, A, C	2	33.33%
17.	D,W, C	3	50.00%
18.	D, A, W, C	2	33.33%
19.	A	4	66.67%
20.	A, W	4	66.67%
21.	A, C	4	66.67%
22.	A, C, W	4	66.67%
23.	W	5	83.33%
24.	W, C	5	83.33%
25.	C	6	100.00%

Để khai thác song song tập phổ biến sử dụng thuật toán FP-Growth, có thể thực hiện như các bước dưới đây:

Thuật toán khai thác tập phổ biến song song thực hiện 2 nhiệm vụ chính:

- ***Xây dựng song song FP-Tree***

Giai đoạn đầu của thuật toán xây dựng các FP-Tree đồng thời trên mỗi bộ xử lý. Chúng ta chia D cơ sở dữ liệu giao dịch cho P bộ xử lý sao cho chắc chắn mỗi bộ xử lý có N/P giao dịch. Với N và P lần lượt là tổng số giao dịch trong CSDL "D" và số các bộ xử lý. Công việc phân hoạch CSDL "D" cho P được thực hiện ngẫu nhiên. Khi kết thúc phân hoạch dữ liệu, công việc tiếp theo là xác định 1-itemset phổ biến trước khi xây dựng FP-Tree cục bộ. Mỗi bộ xử lý tính toán số đếm phổ biến của mỗi mục bằng cách quét phân hoạch CSDL cục bộ. Sau đó, các mục có độ phổ biến nhỏ hơn độ phổ biến tối thiểu được loại bỏ.

Giai đoạn tiếp theo là xây dựng các FP-Tree cục bộ, mỗi bộ xử lý quét CSDL cục bộ của nó, sau đó chèn các tập mục phổ biến vào cây FP-Tree. Công việc xây dực cây FP-Tree bởi mỗi bộ xử lý với CSDL cục bộ của nó giống như trong thuật toán tuần tự.

- ***Khai thác song song và sinh tập phổ biến***

Trong giai đoạn đầu, chúng ta xét toàn bộ các FP-Tree và tạo ra các mẫu điều kiện cơ sở.

Giai đoạn tiếp theo, chúng ta tập hợp các mẫu điều kiện cơ sở từ các bộ xử lý để xây dựng FP-Tree điều kiện cơ sở cho mỗi mục phổ biến.

Khi thực hiện sinh các FP-Tree điều kiện, mô hình "Chủ-cục bộ" được sử dụng. Khi bộ xử lý Chủ chuyển các mục cần khai báo cho các bộ xử lý cục bộ. Các bộ xử lý cục bộ sinh các FP-Tree điều kiện cho các mục đó. Sau khi hoàn thành, nó gửi mã thông báo đến bộ xử lý Chủ để yêu cầu mục kế tiếp. Chi phí cho việc truyền thông này tương đối thấp vì mỗi bộ xử lý chỉ chuyển 1 mã thông báo.

Bằng cách xây dựng lần lượt các mẫu điều kiện cơ sở và các cây điều kiện FP-Tree bởi mỗi bộ xử lý. Khi nhánh của một cây FP-Tree điều kiện được xây

dựng, có thể thu được tập các mục có khả năng phổ biến. Mô hình song song được áp dụng là mô hình "Chủ-cục bộ".

Với mô hình này, khi bộ xử lý cục bộ làm xong nhiệm vụ, nó được nhận ngay nhiệm vụ khác, Điều này giúp cho bộ xử lý luôn trong trạng thái hoạt động trong quá trình khai thác. Sau khi giai đoạn khai thác kết thúc, tất cả các kết quả được chuyển về bộ xử lý chủ để tổng hợp và đánh giá.

2.3.5. Nhận xét thuật toán FP-Growth

Thuật toán FP-Growth giải quyết được các hạn chế của thuật toán Apriori như không sinh số lượng lớn tập ứng viên, chỉ phải quét CSDL hai lần. Thuật toán cũng rút gọn hợp lý các thông tin cần thiết bởi các mục không phổ biến đã bị loại ngay từ đầu.

Thuật toán FP-Growth xử lý lượng lớn CSDL rất hiệu quả và có tốc độ thực thi tỷ lệ rất hiệu quả so với lượng giao dịch lớn, sự lặp lại nhiều lần hay lặp lại nhiều lần cục bộ các giao dịch sẽ được kết hợp lại tạo thành các nhánh của FP-tree.

Thuật toán FP-Growth khá hiệu quả vì sử dụng cấu trúc cây FP-tree. Tuy nhiên vì FP-Growth làm việc trên tất cả các tập phổ biến nên vẫn chậm hơn một số thuật toán khác sử dụng cây FP-tree.

Kết luận chương 2:

Trong chương này đã trình bày một số thuật toán khai thác tập phổ biến kinh điển như thuật toán Apriori, Eclat, FP-Growth. Qua đó, các ưu, nhược điểm của từng thuật toán cũng được nêu ra. Tuy nhiên, các thuật toán song song dựa trên xây dựng các hệ thống máy tính lớn, được thiết kế phân tán thì cần chi phí đồng bộ và giao tiếp dữ liệu lớn cũng như tốn kém về chi phí xây dựng phần cứng. Vì vậy để tiết kiệm chi phí đầu vào/ đầu ra cũng như chi phí giao tiếp, thiết kế hệ thống, chương tiếp theo của luận văn đi sâu tìm hiểu, nghiên cứu về một phương pháp khai thác tập phổ biến mới, có thể tối ưu hóa việc sử dụng phần cứng của một máy tính riêng lẻ để khai thác song song dữ liệu đó là phương pháp khai thác song song tập phổ biến dựa trên mảng Systolic.

CHƯƠNG 3. THUẬT TOÁN KHAI THÁC SONG SONG TẬP PHỔ BIẾN DỰA TRÊN MẢNG SYSTOLIC

Nhiều thuật toán song song và phân tán đã được đề xuất để khai thác các tập phổ biến trong CSDL giao dịch [15-17]. Hầu hết các thế hệ đầu tiên của thuật toán khai thác mẫu song song và phân tán được nghiên cứu dựa trên thuật toán Apropri. Tất cả những thuật toán đều có những điểm hạn chế giống Apriori là phải duyệt CSDL nhiều lần, số lượng tập ứng viên lớn.

Thuật toán FP-growth được phát triển dựa trên kỹ thuật Apriori, thuật toán xây dựng một số FP-tree cục bộ trong môi trường bộ nhớ phân tán. Một số lượng lớn các nghiên cứu về kỹ thuật khai thác FP-tree theo mô hình khai thác song song và phân tán đã được đưa ra ở trong các chiến lược khai thác song song và phân tán luật kết hợp [11]. Những công trình này phân vùng dữ liệu thành nhiều phần và sau đó xây dựng FP-tree riêng cho từng phần song song [11]. Mặt khác, có một số thuật toán khác xây dựng cây FP-tree cho toàn bộ CSDL, bằng kỹ thuật tạo cây song song.

Một thuật toán song song hiệu quả khai thác tập phổ biến dựa trên nền tảng song song không chia sẻ đã được trình bày trong bài báo về thuật toán Closet+, một chiến lược để khai thác tập phổ biến đóng. Bằng cách phân vùng hiệu quả danh sách các phần tử phổ biến trên bộ xử lý, thuật toán PFP-tree cố gắng giảm tối thiểu chi phí kết nối và chi phí đồng bộ hóa. LFP-Tree là một thuật toán khai thác phân tán và song song khác dựa trên cấu trúc FP-tree. Thuật toán phân chia các tập phần tử cần khai thác theo chiều rộng và chiều sâu của cây. Thời gian kết nối trong LFP-tree đã giảm bớt bằng cách giữ các phần tử có khả năng phổ biến trong các nút máy tính cục bộ của nó, vì vậy LFP-tree giảm được chút thời gian tính toán so với PFP-tree. Ngoài ra, nó có tốc độ tốt hơn so với PFP-tree khi số lượng bộ vi xử lý tăng lên.

Thuật toán song song Eclat dùng phương pháp nhóm tập phổ biến có liên quan với nhau bằng cách sử dụng lược đồ phân chia lớp tương đương, với mỗi lớp tương

đương chứa các tập mục ứng viên quan hệ tương đương với nhau. Phương pháp này sử dụng kỹ thuật tổ chức CSDL theo chiều dọc để nhóm các giao dịch có liên quan với nhau. Giải thuật này quét CSDL ít hơn, giảm thiểu chi phí vào và ra. Hơn nữa, sự độc lập giữa các bộ xử lý làm giảm chi phí truyền thông và đồng bộ.

Mặc dù các phương pháp FP-tree cục bộ hoặc toàn cục có thể được xây dựng mà chỉ tốn ít chi phí giao tiếp giữa các bộ vi xử lý, nhưng hầu như tất cả các phương pháp tiếp cận đều tốn quá nhiều chi phí giao tiếp, trong quá trình KTDL. Tất cả các phương pháp khai thác song song được phát triển trước đây cũng có một vấn đề chung là "làm thế nào chúng ta có thể chia các bộ dữ liệu giữa các máy tính và khai thác độc lập mà không cần phải chia sẻ thông tin và kết quả".

Mảng Systolic được sử dụng để giải quyết vấn đề này. Sử dụng kỹ thuật mảng Systolic cho phép khai thác tập phổ biến song song mà không cần phải chia sẻ thông tin và kết quả, đồng thời không làm quá tải dữ liệu đầu vào/ đầu ra (Input/ Output). Ngoài ra với việc sử dụng mảng systolic, có thể thực hiện khai thác song song trên cùng một máy tính tùy thuộc vào khả năng mạnh yếu của phần cứng, tận dụng được sức mạnh của bộ xử lý đồ họa (GPU) và vi bộ xử lý của máy tính (CPU). Qua đó, giảm thiểu được thời gian giao tiếp cũng như chi phí thiết lập hệ thống song song.

3.1. Bài toán khai thác song song tập phổ biến dựa trên mảng Systolic

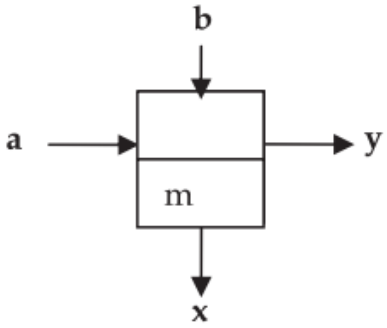
3.1.1. Cấu trúc mảng Systolic

Năm 1978 Kung và Leiserson đề xuất một loại kiến trúc được gọi là mảng Systolic cho những tính toán đặc biệt. Trong kiến trúc máy tính song song, một mảng Systolic là một mạng đồng nhất của các liên kết trong bộ vi xử lý-CPU được gọi là "bộ xử lý".

Một mảng Systolic là sự sắp xếp từng vi xử lý trong một mảng, dòng chảy dữ liệu đồng bộ chảy qua từng bộ xử lý trong mảng, thông thường các dòng dữ liệu khác nhau chảy theo hướng khác nhau. Mỗi bộ xử lý tại mỗi bước lấy dữ liệu từ

những bộ xử lý kế nó, xử lý nó, và kết quả được đi ra theo hướng ngược lại. Đây là một dạng đặc biệt của tính toán song song. Bộ xử lý tính toán dữ liệu và lưu trữ nó độc lập. Trong loại song song này, không giống như nhiều hình thức song song song khác, không bị mất tốc độ vì thời gian kết nối giữa các bộ xử lý.

Hình 3.1 cho thấy ví dụ của một bộ xử lý trong mảng Systolic. Bộ xử lý này có 2 đầu vào "a" và "b"' và 1 bộ nhớ "m" để lưu dữ liệu. Sau đó mỗi lần tính toán, bộ xử lý trả về 2 giá trị đầu ra x và y. Công thức tính toán được xác định cho bộ xử lý.



Hình 3.1. Một bộ xử lý trong mảng Systolic [4]

3.1.2. Mục đích sử dụng và hiệu quả của mảng Systolic

Một lợi ích lớn của mảng Systolic là tất cả các dữ liệu toán hạng và từng phần của kết quả được lưu trữ (đi qua) bên trong các mảng vi xử lý. Cấu trúc mảng Systolic không cần phải truy cập vào các bus bên ngoài, bộ nhớ trong đối với từng hoạt động như là trường hợp với các máy tuần tự Von Neumann hoặc Harvard. Nó cũng không bị ảnh hưởng bởi các giới hạn tuần tự về hiệu suất song song, vì các dữ liệu đầu vào được ngầm xử lý bởi các node được lập trình liên kết với nhau và không có các bước tuần tự trong việc quản lý các luồng dữ liệu cao song song.

Do đó mảng Systolic rất thích hợp để áp dụng vào các lĩnh vực trí thông minh nhân tạo, xử lý ảnh, nhận dạng mẫu, máy tính ảo và các nhiệm vụ khác. Bộ xử lý mảng Systolic cũng có thể áp dụng rất tốt ở lĩnh vực máy học bằng cách thực hiện tự cấu hình mạng lưới neural trong phần cứng.

Hiệu quả của mảng Systolic phụ thuộc rất nhiều vào các đặc tính vào/ ra của dữ liệu. Nó sẽ rất hiệu quả đối với những bài toán mà số liệu đọc/ ghi thực hiện với

nhịp độ cao, đều đều và liên tục như các bài toán xử lý ảnh, qui hoạch tuyến tính, v.v.

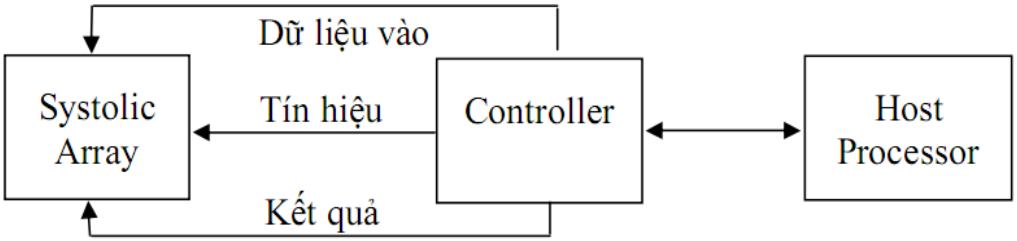
3.1.3. Mô tả chi tiết mảng Systolic

Trong mảng Systolic, mỗi thành phần xử lý được xem như một tế bào (một ô trong mảng) bao gồm:

- Một số thanh ghi.
- Một bộ cộng.
- Các mạch điều khiển.
- Đơn vị logic-số học ALU.

Các ô lân cận có thể trao đổi dữ liệu với nhau.

Dựa vào mảng Systolic người ta xây dựng kiến trúc bộ xử lý bằng mảng Systolic (SAP) [1].

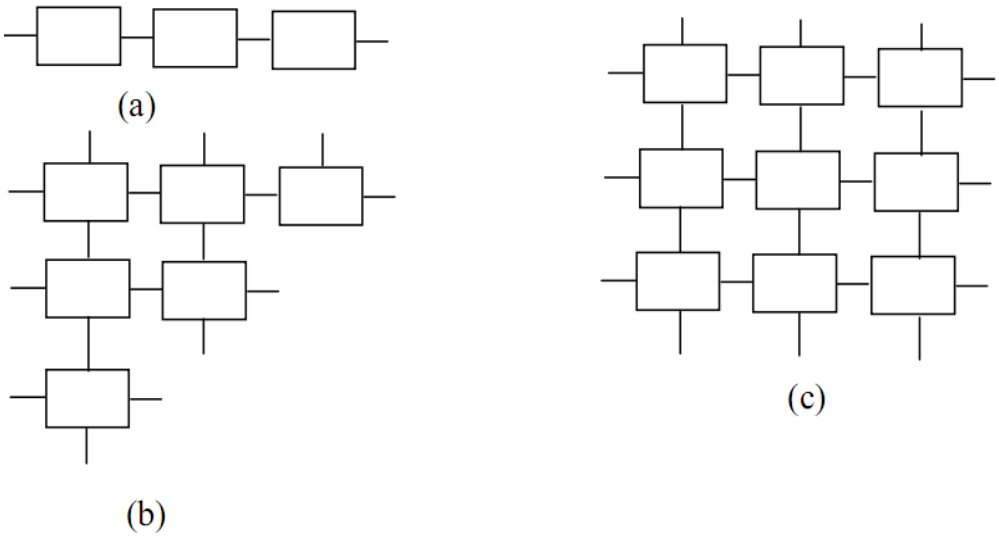


Hình 3.2. Kiến trúc bộ xử lý mảng Systolic

Trong kiến trúc SAP nêu trên: bộ điều khiển làm nhiệm vụ giao diện cho BXL chính và gửi các tín hiệu điều khiển quá trình vào/ra dữ liệu cho SA.

Hoạt động của hệ thống theo từng nhịp và lặp lại một cách đều đặn để tận dụng được khả năng song song của tất cả các phần tử xử lý.

Mảng Systolic có thể tổ chức theo nhiều cấu hình khác nhau. Hình 3.3 mô tả một số cấu hình phổ biến của mảng Systolic.



Hình 3.3. Một số cấu hình phổ biến của mảng Systolic: (a) mảng tuyến tính, (b) mảng hình tam giác, (c) mảng hai chiều hình vuông

Ví dụ: Xét bài toán nhân 2 ma trận cỡ 2×2: $A * B = C$

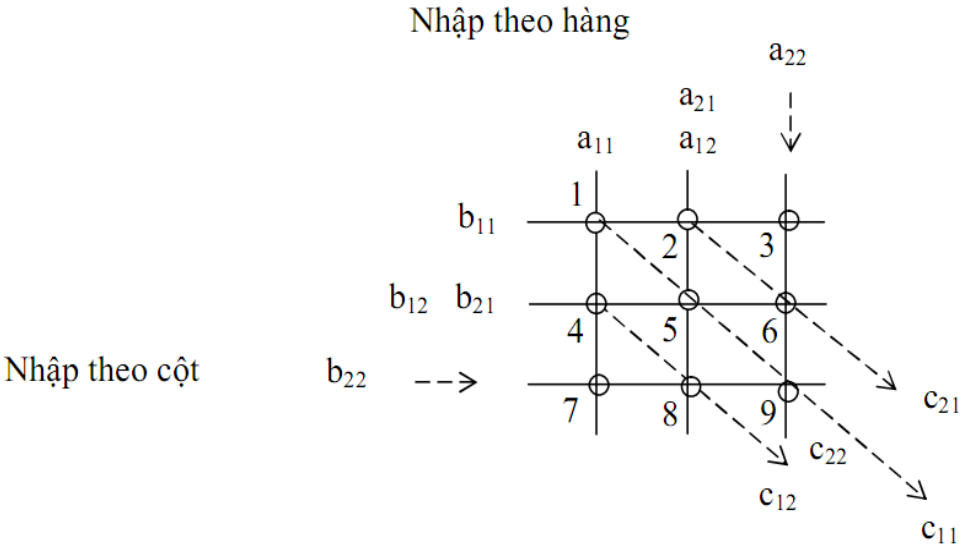
$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

Hiển nhiên $C_{ij} = \sum a_{ik} \times b_{kj} (k = 1,2)$

$$c_{11} = a_{11} \times b_{11} + a_{12} \times b_{21} \qquad c_{12} = a_{11} \times b_{12} + a_{12} \times b_{22}$$

$$c_{21} = a_{21} \times b_{11} + a_{22} \times b_{21} \qquad c_{22} = a_{21} \times b_{12} + a_{22} \times b_{22}$$

Chúng ta có thể thiết kế mảng Systolic có 9 thành phần xử lý để thực hiện nhân hai ma trận trên như sau:



Hình 3.4. Kiến trúc SA để thực hiện nhân hai ma trận

Mô tả các bước thực hiện nhân 2 ma trận trong kiến trúc mảng Systolic:

Nhịp 1:

- Nhập a_{11}, b_{11}vào ô số 1 và tính $a_{11} * b_{11}$.
- Nhập b_{21}vào ô số 4 và a_{21}vào ô số 2.

Nhịp 2:

- Truyền b_{11} từ ô số 1 sang ô số 2.
- Truyền a_{11} từ ô 1 sang ô 4 và tính $a_{11} * b_{21}$.
- Truyền b_{11} từ ô 1 sang ô 2 và tính $* b_{11}$.
- Truyền b_{12} từ ô 4 sang ô 5 và a_{12} từ ô 2 sang ô 5 và tính $a_{12} * b_{21}$.
- Tính $c_{11} = a_{12} * b_{11} + a_{12} * b_{21}$.
- Nhận tiếp b_{12} vào ô 4 và a_{21} vào ô số 2.

Nhịp 3:

- Truyền c_{11} từ ô số 5 sang ô số 9
- Truyền $a_{21} * b_{11}$ từ ô 2 sang ô 6
- Truyền b_{12} từ ô 5 sang ô 6
- Nhập a_{22} vào ô 3 và nhập b_{22} vào ô 7

Nhịp 4:

- Truyền a_{22} từ ô số 3 sang ô số 6
- Tính $a_{22} * b_{21}$
- Truyền $a_{21} * b_{11}$ từ ô 2 sang ô 6
- Cộng dồn kết quả được chuyển từ ô 2 sẽ cho $c_{21} = a_{21} * b_{11} + a_{22} * b_{21}$
- Chuyển c_{11} từ ô 9 ra và gán cho c_{11}

3.2. Thuật toán khai thác tập phổ biến sử dụng mảng Systolic

3.2.1. Mã hóa dữ liệu bằng ma trận bit

Để thực hiện tính toán trên máy tính, dữ liệu cần được mã hóa thành mã nhị phân mà cụ thể ở đây các tập dữ liệu được mã hõa thành một ma trận bit với giá trị tương ứng.

Ví dụ: Xét CSDL trong Bảng 3.1:

Bảng 3.1. CSDL có các phần tử được ký hiệu theo chữ cái

Tid	Itemset	Tập phổ biến Minsup = 2
1	*a, c, f, p*	*a, c, f*
2	*d, e, f, m*	*d, e, f*
3	*a, b, d, f*	*a, b, d, f*
4	*a, c, e, h*	*a, c, e*
5	*c, d, e, f, k*	*c, d, e, f*
6	*b, c, d, e, f*	*b, c, d, e, f*
7	*b, d, e, f*	*b, d, e, f*
8	*e, f*	*e, f*

Tập các item được sử dụng trong cơ sở dữ liệu này là I = {a, b, c, d, e, f, h, k, m, p}. Trong bộ dữ liệu này, X = {b, e, f} là tập phần tử phổ biến với sup=2 bởi vì nó là tập con của 2 giao dịch 6 và 7.

Để đơn giản hơn, cho tập X = bef. "bef" không phải là một tập đóng bởi vì tồn tại một tập cha "bdef" trong bộ dữ liệu có cùng độ phổ biến. Tuy nhiên "bdef" là một tập đóng bởi vì chỉ có một tập cha của nó "bcdef" có độ phổ biến là 1.

Với tập dữ liệu D và một ngưỡng minsup, vấn đề của việc khai thác tập phổ biến là tìm tất cả tập phổ biến tương ứng với ngưỡng minsup, tương tự với vấn đề của

việc khai thác tập phổ biến đóng là tìm tập phổ biến đóng tương ứng với ngưỡng minsup.

Chúng ta có thể biểu diễn một bộ dữ liệu thành một ma trận bit [1] với m hàng và n cột sao cho mỗi hàng tương ứng với một giao dịch của tập dữ liệu và mỗi cột tương ứng với một phần tử trong tập "I". Mỗi hàng của tập dữ liệu là một chuỗi "n bit", nếu một hàng dữ liệu có chứa phần tử "i" thì bit thứ "i" của chuỗi tương ứng là 1 và ngược lại là 0. Do đó bộ dữ liệu được nén thành một ma trận bit.

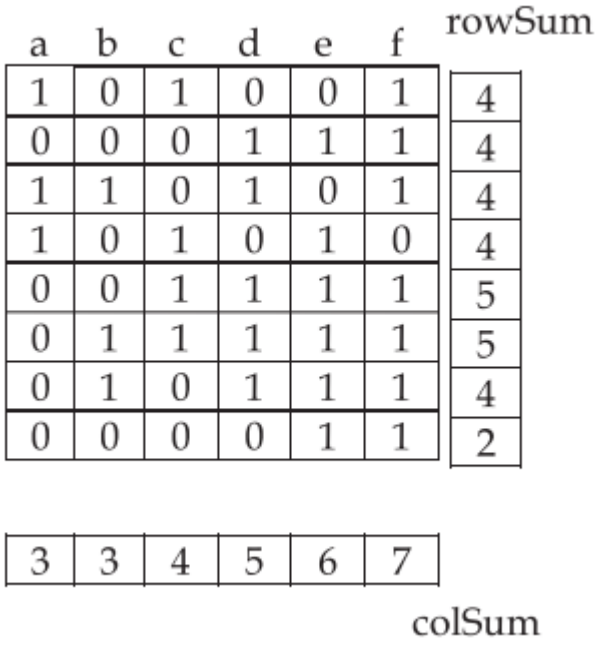
Ví dụ: Ma trận bit tương ứng với dữ liệu của Bảng 3.1 là Hình 3.5

a	b	c	d	e	f	h	k	m	p	rowSum
1	0	1	0	0	1	0	0	0	1	4
0	0	0	1	1	1	0	0	1	0	4
1	1	0	1	0	1	0	0	0	0	4
1	0	1	0	1	0	1	0	0	0	4
0	0	1	1	1	1	0	1	0	0	5
0	1	1	1	1	1	0	0	0	0	5
0	1	0	1	1	1	0	0	0	0	4
0	0	0	0	1	1	0	0	0	0	2
3	3	4	5	6	7	1	1	1	1	colSum

Hình 3.5. Ma trận bit của tập dữ liệu

Hai mảng của ma trận bit: colSum là một mảng n-phần tử chỉ ra cho biết độ phổ biến của từng phần tử và rowSum là một mảng có m phần tử chỉ ra số lượng các phần tử trong mỗi hàng. Chúng ta có thể lượt bớt cột của ma trận bit theo minsup, loại bỏ tất cả các cột tương ứng với colsum có giá trị nhỏ hơn minsup.

Cột thứ 3 của bảng 3.1 cho thấy dữ liệu được cắt bớt dựa trên ngưỡng minsup = 2. Hình 3.6 là một ma trận bit đã được rút gọn sau khi loại bỏ các cột tương ứng với colSum có giá trị nhỏ hơn minsup = 2



Hình 3.6. Ma trận bit rút gọn với minsup = 2

Để xác định xem một tập phần tử X là phổ biến hay không, chúng ta xây dựng ColAndVector của nó bằng cách thực hiện toán tử "And" trên các cột tương ứng với tất cả các phần tử của X. Nếu tổng của các giá trị trong một ColAndVector không nhỏ hơn minsup thì tập phần tử của ColAndVector đó là phổ biến.

Ví dụ: ColAndVector (ab) chỉ có một bit với giá trị 1 (bit thứ 3) và do đó ab là không phổ biến.

Để chỉ ra tập phổ biến X là đóng từ ma trận bit, chúng ta nên xây dựng ColAndVector của X và toán hạng "And" trên ColAndVector với cột của từng phần tử không tồn tại trong X. Nếu kết quả cuối cùng của toán hạng AND là bằng với colAndVector của X thì X không đóng.

Ví dụ: Trong Hình 3.6, colAndVector (bef) = 00000110 và colAndVector (bdef) = 00000110 và vì thế "bef" không phải là tập đóng.

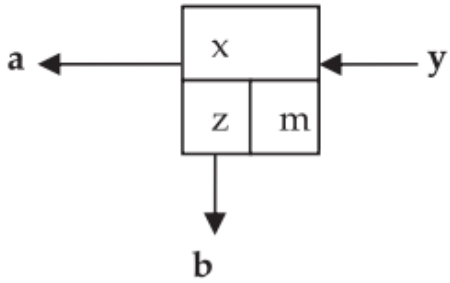
3.2.2. Xây dựng cấu trúc mảng Systolic để khai thác tập phổ biến

Để sử dụng mảng Systolic như là môi trường để phát triển thuật toán khai thác song song tập phổ biến, đầu tiên cần xác định giá trị đầu vào, đầu ra và bộ nhớ trong mỗi bộ xử lý. Vì mục đích này, đầu tiên cần xây dựng một mảng Systolic có khả năng xây dựng tất cả các tập phổ biến của một bộ dữ liệu và vector bit tương ứng của nó để cho biết là nó có phải là tập phổ biến hay không.

Ý tưởng chính là:

- Xác định những phần tử phổ biến trong tập dữ liệu, nếu có tồn tại "n" phần tử phổ biến trong tập dữ liệu, sẽ có $2^n - 1$ tập phần tử có khả năng phổ biến.
- Kết quả là có 1 đến $2^n - 1$ chuỗi nhị phân của tập phần tử, và mỗi phần tử tương ứng với với 1 bit của trong từng chuỗi.

Để thực hiện ý tưởng này, mảng Systolic một chiều được thiết kế với các bộ xử lý như Hình 3.7.



Hình 3.7. Chức năng của một bộ xử lý trong mảng Systolic 1 chiều [4]

Thông tin và luật của từng bộ xử lý trong mảng Systolic như sau:

- Mỗi bộ xử lý có một bộ nhớ **(x)** lưu trữ vector bit của một phần tử.
- vector bit tương ứng với mỗi bộ xử lý có một bộ nhớ **(m)** lưu trữ tên của một phần tử.
- Mỗi bộ xử lý có một bộ nhớ **(z)** lưu trữ một bit. Nếu có tồn tại phần tử của bộ xử lý tương ứng trong tập phần tử thì **z = 1**, ngược lại **z = 0.**

*Mỗi bộ xử lý có đầu vào **(y)** gồm 3 phần:*

- **y(IS)** là một tập phần tử.

- **y(array)** là ứng của tập phần tử.
- **y(sw)** để xác định xem tiến trình xử lý đã hoàn thành hay không.

Mỗi bộ xử lý có 2 đầu ra ***"a"*** *và* ***"b".***

Đầu tiên, giả sử chúng ta có một mảng một chiều với n bộ xử lý (n là số lượng các phần tử phổ biến trong các bộ dữ liệu) sao cho đầu ra **"a"** của bộ xử lý **"i"** là đầu vào **"y"** của bộ xử lý **"i+1"** (cho mỗi 0 < i < n).

Đầu vào ban đầu cho bộ xử lý đầu tiên trong từng tiến trình là **y**. Với:

- $\boldsymbol{y(IS)} = \emptyset$
- **y(array)** là một **vector** n bit có giá trị = 1
- $\boldsymbol{y(sw)} = 1$

Ví dụ: Xét CSDL ở Bảng 3.1.

Nếu minsup là 2 thì kết quả là cột colSum của hình 3.5. Do đó tập các phần tử phổ biến là {a, b, c, d, e, f} tương ứng ma trận bit Hình 3.6.

Vì mỗi bộ xử lý của mảng Systolic lưu trữ các bit đại diện cho một phần tử, vì vậy cần chuyển bộ dữ liệu sang chiều dọc. Các bộ dữ liệu theo chiều dọc có thể được tạo ra bởi chuyển đổi ma trận bit tương ứng. Trong Hình 3.8 cho thấy ma trân bit chuyển đổi tương ứng Hình 3.6

									rowSum
a	1	0	1	1	0	0	0	0	3
b	0	0	1	0	0	1	1	0	5
c	1	0	0	1	1	1	0	0	6
d	0	1	1	0	1	1	1	0	4
e	0	1	0	1	1	1	1	1	6
f	1	1	1	0	1	1	1	1	5
colSum	3	1	3	4	3	1	2	1	

Hình 3.8. Ma trận bit chuyển đổi từ tập phần tử tương ứng với hình 3.5

Mỗi dòng của ma trận bit chuyển đổi tương ứng với một vector bit của một phần tử và được lưu trong bộ xử lý của mảng Systolic.

Ý tưởng khai thác là xây dựng tất cả các tập phần tử có thể có và vector bit tương ứng của nó, và kiểm tra độ phổ biến của nó bằng cách đếm số lượng các bit có giá trị "1" trong vector bit tương ứng. Vì mỗi cột của ma trận bit chuyển đổi tương ứng với một hàng của các tập dữ liệu, nên nếu số bit có giá trị 1 trong vector bit tương ứng của tập phần tử lớn hơn minsup thì tập đó sẽ là một tập phổ biến. Mẫu này và vector bit tương ứng của nó di chuyển giữa các bộ xử lý và có một biến kiểm tra quá trình tạo mẫu được kết thúc qua từng bước.

Quá trình tính toán của mỗi bộ xử lý trong qua từng bước được dựa trên các quy tắc sau:

Với y(sw) = 1:

- Nếu z = 1:
  - y(array) = ((x) AND (y(array))
  - y(IS) = y(IS)∪m
  - y(sw) = 1
  - z = 0
  - a = y
  - b = *
- Nếu z = 0:
  - y(sw) = 0
  - z = 1
  - a = y
  - b = *

Với y(sw) = 0:

- Nếu z = 1:
  - y(array) = ((x) AND (y(array))
  - y(IS) = y(IS) = y(IS)∪m
  - y(sw) = 0
  - a = y
  - b = *
- Nếu z = 0:
  - a = y
  - b = *

Quá trình tính toán này tạo ra tất cả các số từ 1 đến $2^n - 1$ trong các bộ xử lý của mảng Systolic mà mỗi bit của số được lưu trữ trong một bộ xử lý của mảng.

3.3. Phương pháp khai thác song song dựa trên mảng Systolic

3.3.1. Phương pháp tiếp cận chia để trị

Phương pháp tiếp cận chia để trị được sử dụng để cải thiện phương pháp song song. Trong cách tiếp cận này tập dữ liệu được phân chia dựa trên các tập phổ biến tối thiểu. Nói chung, các tập khai thác của mỗi bộ dữ liệu được chia thành các loại khác nhau dựa trên các phần tử. Vì mục tiêu này, trước tiên các phần tử trong mỗi giao dịch được sắp xếpvề cùng một cấp độ và sau đó phân loại ra những tập phổ biến theo cấp độ như sau:

- Những tập có chứa những phần tử đầu tiên (cùng cấp) sẽ là thành viên của nhóm thứ 1.
- Những tập có chứa các phần tử thứ hai và không có chứa phần tử đầu tiên sẽ là thành viên của nhóm thứ 2.
- Tập có chứa các phần tử thứ ba và không phần tử đầu tiên và thứ hai sẽ là thành viên của nhóm thứ ba và vân vân.

Ví dụ: Xét các ma trận bit của hình 3.7. Ma trận bit này có 6 phần tử mà được sắp xếp theo thứ tự alphabets (ở đây thứ tự sắp xếp chung là dựa vào alphabets). Mẫu rút trích của ma trận bit này được chia thành 6 loại khác nhau như sau:

(1) Những mẫu chứa a.

(2) Những mẫu chứa b và không chứa a.

(3) Những mẫu chứa c và không chứa a và b

(4) Những mẫu chứa d và không chứa a và b và c

(5) Những mẫu chứa e và không chứa a và b và c,d

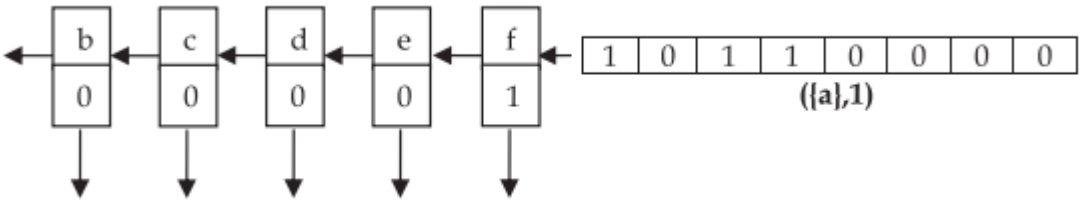
(6) Những mẫu chứa f.

Hướng tiếp cận này được sử dụng trong khai thác tập phổ biến song song để giảm thời gian thực thi. Vì mục tiêu này, mỗi nhóm được chia thành mảng systolic và sử dụng mảng để xây dựng mẫu của từng nhóm. Vì vậy, phương pháp tăng mức độ song song.

3.3.2. Mảng Systolic 2 chiều

Dựa trên hướng tiếp cận chia để trị, một mảng Systolic với n-1 bộ xử lý được sử dụng để khai thác loại thứ nhất của tập dữ liệu (tập phổ biến có chứa a).

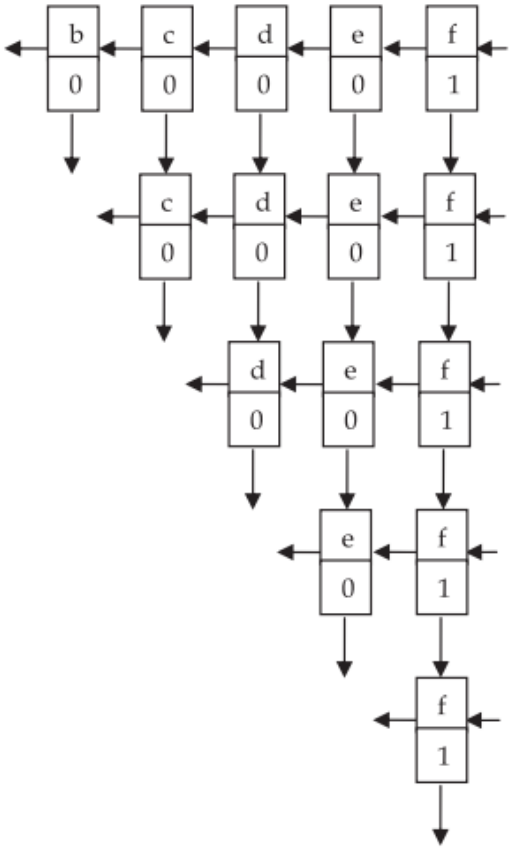
Hình 3.9 cho thấy cấu trúc của mảng này.



Hình 3.9. Khởi tạo ban đầu của mảng systolic để khai thác tập phần tử có chứa "a"

Trong mảng này, cấu trúc giống như mảng trong phần 4 được sử dụng nhưng đầu vào bao gồm phần tử "a" và mảng không có bộ xử lý tương ứng với phần tử "a".

Giá trị đầu vào di chuyển trong mảng và bộ xử lý giống như mảng được mô tả ở trên. Có n - 1 mảng cho n - 1 phần tử (tất cả nhưng trừ phần tử cuối cùng) của tập dữ liệu.



Hình 3.10. Mảng systolic hai chiều khai thác tập dữ liệu phổ biến [4]

Bộ xử lý có khả năng liên tục tính toán mỗi lần hoàn thành từng dòng của mảng trong khi dữ liệu đầu vào tiến đến phía bên trái "1" của mảng systolic. Hơn nữa nếu số bit "1" trong y (array) nhỏ hơn độ phổ biến tiến trình của y sẽ dừng lại. Vì vậy có thể sắp xếp các phần tử theo độ phổ biến của nó và xác suất dừng tiến trình tăng lên ở các hàng cao hơn, nhiều bộ xử lý hơn hơn và do đó thuật toán của sẽ được cải thiện.

3.4. Thuật toán khai thác dựa trên mảng Systolic

Thông qua cách tiếp cận chia để trị, thuật toán song song SABMA (Systolic Array Based Mining Algorithm) được trình bày, khai thác các tập phổ biến của bộ dữ liệu hiệu quả. SABMA dùng 1 mảng Systolic 2 chiều có n-1 dòng, với mỗi dòng "k" ($\forall\ 1 \leq k\ \leq n-1$) chứa "n-k" bộ xử lý.

Thuật toán KTDL dựa trên mảng Systolic (SABMA) được mô tả như sau:

```
Begin
      Sw: boolean;
      Is: String;
      bitVector: array[1..m] of boolean;
End
Procedure SABMA
Input
      m: integer // số lượng hàng của tập dữ liệu
      n: integer // số lượng tập phổ biến
      items: array[1...n] of integer // tập hợp tất cả các
tập phổ biến
      y: array[1...n-1] of InputStruct // đầu vào các dòng
trong mảng systolic
      minsup: integer // độ phổ biến tối thiểu
      bitMat: array[1...n][1...m] of Boolean
Output
      File: textFile // file chứa các tập phổ biến tìm được
trong dataset
      Var
            i: integer;
      Begin
            t:=1;
            while(có ít nhất 1 bit z = 0) then
      Begin
             for i:=1 to n-1 do
            Begin
            y[i].IS:= intostr(items[i] + '');
            y[i].bitVector := bitmat[item[i]];
            y[i].sw = 1;
            đưa y[i] đến bộ xử lý đầu tiên của dòng thứ "i"
trong mảng systolic
            if(có tồn tại giá trị trong output "a" của bộ xử
lý ở vị trí ([log2^(t+1)] + 1)
      then
            if (item là tập phổ biến) then output (File, a.IS)
      End
      t:= t+1;
End
End
```

Diễn giải thuật toán:

Tham số đầu vào của thuật toán này bao gồm:

- **"m"** là số dòng của tập dữ liệu (là số của cột ma trận bit chuyển đổi).
- **"n"** là số phần tử phổ biến (là số hàng của ma trận bit chuyển đổi).
- **"y"** là dữ liệu đầu vào của mỗi hàng của mảng Systolic.
- **minsup** do người dùng chỉ định là độ phổ biến tối thiểu.
- **bitMat** là ma trận bit chuyển đổi từ tập dữ liệu.

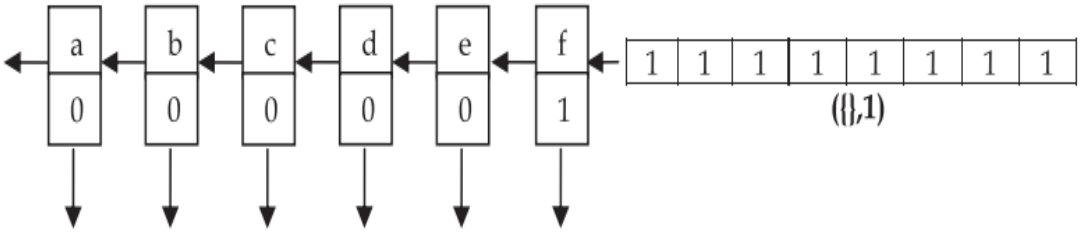
Trong phần chính (trong khối chính "while' của thuật toán), các thuật toán xác định đầu vào cho các hàng khác nhau của mảng Systolic, lặp đi lặp lại, sau đó gởi chúng đến các bộ xử lý thứ nhất. Trong mỗi vòng của vòng lặp, các thuật toán cũng kiểm tra output của các bộ xử lý cuối cùng để xuất các giá trị thành file.

Trong mỗi vòng lặp for bên trong, đầu tiên giá trị đầu vào y được khởi tạo với các phần tử có giá trị tương ứng.

Ví dụ: Xét CSDL đã được mã hóa bằng ma trận bit ở hình 3.8. Có 6 phần tử {a, b, c, d, e, f} trong tập dữ liệu.

=> n = 6 và có 5 vòng lặp cho 5 dòng.

Khởi tạo ban đầu của mảng Systolic trong trường hợp này như dưới đây. Hình 3.9 là cho thấy giá trị đầu vào của mảng Systolic khi được khởi tạo lần đầu.



Hình 3.9. Khởi tạo ban đầu của mảng Systolic [4]

Trong hình 3.9 chỉ chỉ ra tên từng phần tử, vị trí của nó trong bộ xử lý và vector bit chuyển đổi tương ứng của nó.

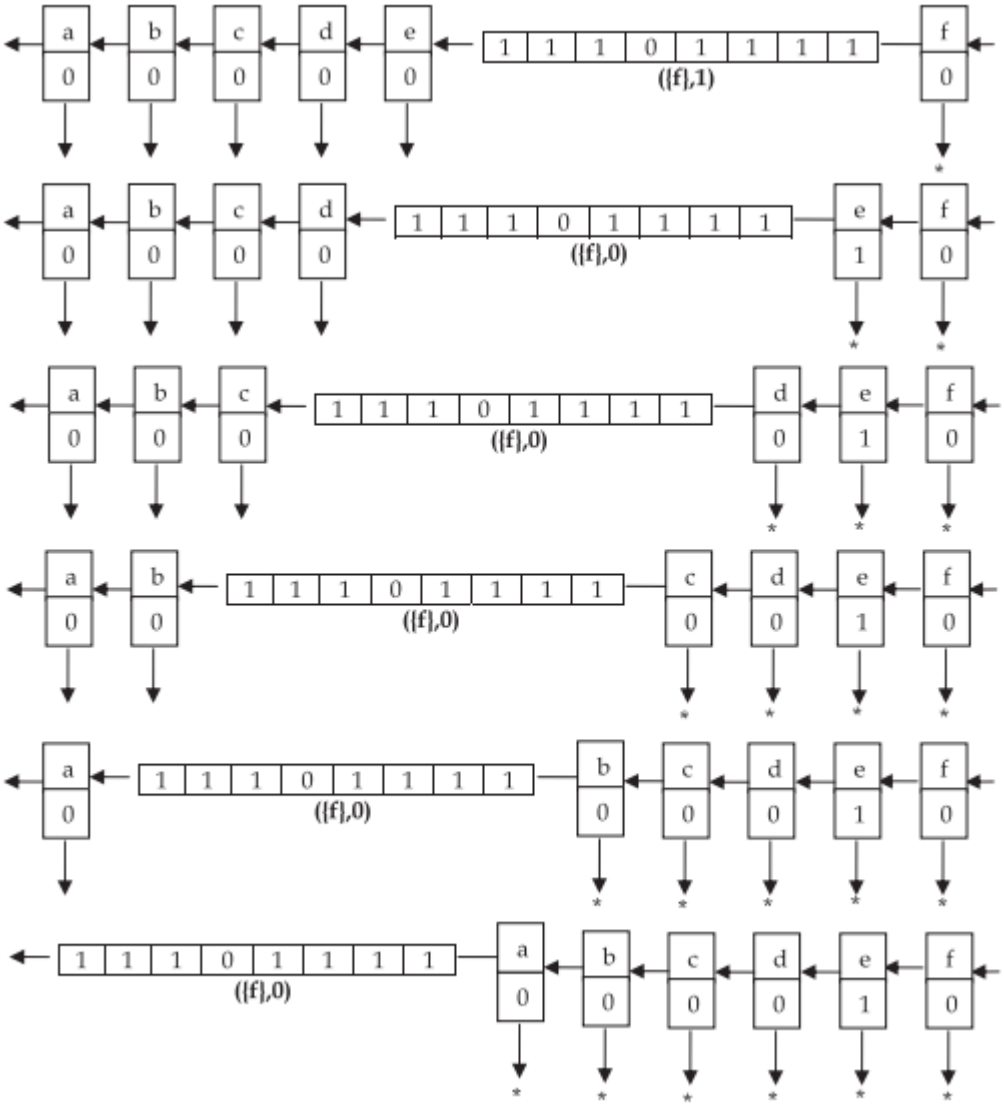
Trong lần lặp đầu tiên, y[1] được khởi tạo với giá trị:

Quá trình lưu trữ trong mảng được mô tả trong hình trong hình 3.10 – 3.12

Vì vectorbit được khởi tạo là vector bit của tập ∅ nên có giá trị của tất cả các ô là 1. Bit z đầu tiên cũng được đánh dấu = 1 để tiết kiệm vòng lặp xử lý.

Đầu tiên 2 vectorbit được AND với nhau để cho ra vectorbit mới.

y[1] .IS='a', y[i] bitVector =10110000 (dòng tương ứng của của phần tử a trong ma trân bit), và y[1].sw =1 cho dòng đầu tiên.



Hình 3.10. Lần di chuyển đầu tiên của mảng Systolic [4]

Hình 3.10- chỉ ra tập phần tử {f} là phổ biến vì số bit 1(đầu ra) của y(array) là 7 và lớn hơn minsup (minsup = 2).

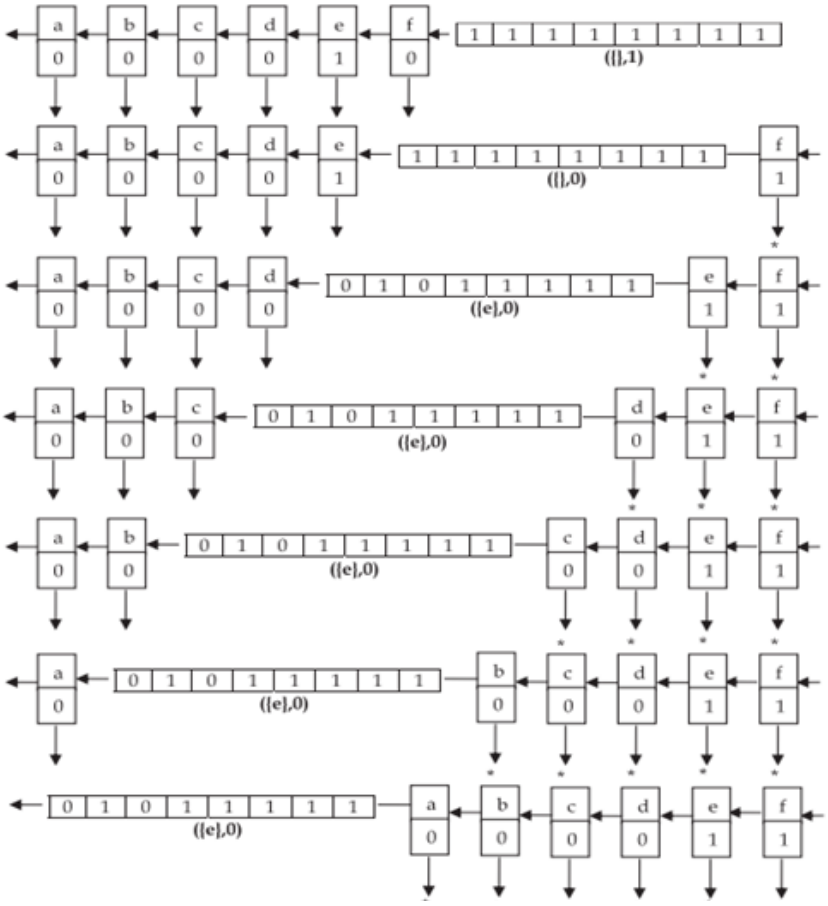
Tương tự, quá trình chuẩn bị giá trị đầu vào tương ứng cho dòng kế tiếp được lặp lại. Sau đó khởi tạo đầu vào, y[1] di chuyển từ bộ xử lý 1 đến bộ xử lý khác trên hàng đầu tiên của mảng Systolic.

Khi giá trị đầu vào này đến bộ xử lý số ($[\log_2{}^{t+1}] + 1$), thuật toán kiểm tra độ phổ biến của các tập phổ biến, và nếu nó là phổ biến, nó sẽ được xuất ra file.

Ở lần lặp thứ 2 của vòng lặp for, y[3] được khởi tạo (‘c’,10011100,1) và di chuyển trên hàng thứ 3 của mảng Systolic.

Khi nó đến số bộ xử lý ($[\log_2{}^{2+1}] + 1$) = 2, tương ứng với tập dữ liệu (ce) sẽ được kiểm tra có phổ biến hay không, và ‘ce’ sẽ được xuất ra file ở đầu ra là tập phổ biến.

Hình 3.11 cho thấy lần di chuyển thứ 2 của mảng Systolic.

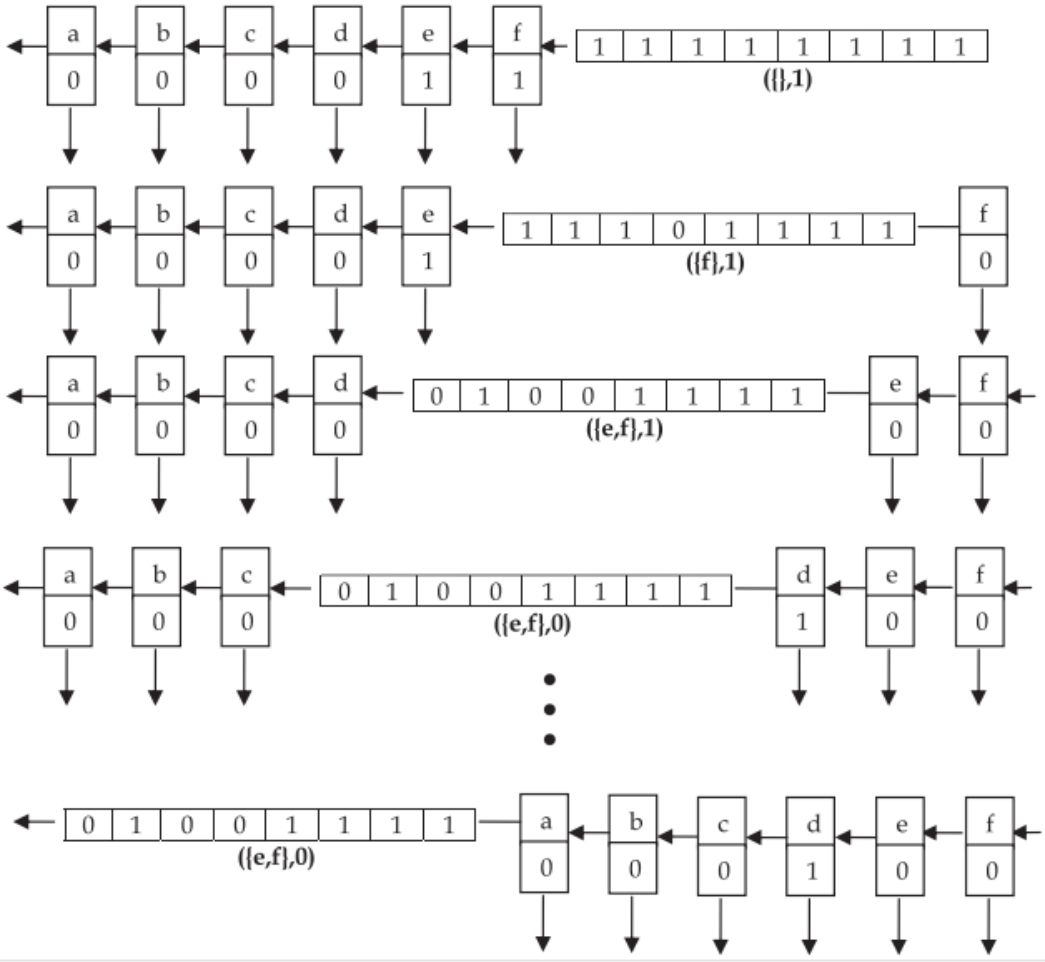


Hình 3.11. Lần di chuyển thứ hai của mảng Systolic [4]

Có thể nhìn thấy rằng sau lần di chuyển đầu tiên các bộ xử lý của mảng xây dựng dạng nhị phân của 2 và sau lần di chuyển thứ hai nó xây dựng các dạng nhị

phân của 3. Cần chú ý rằng, quá trình di chuyển diễn ra liên tục, ngay sau khi lần di chuyển đầu tiên vào bộ xử lý đầu tiên xong, sẽ di chuyển ngay lập tức vào bộ xử lý thứ 2, và dữ liệu tiếp theo lại di chuyển vào bộ xử lý đầu tiên, cứ như thế tiếp tục.

Hình 3.12 cho thấy quá trình di chuyển của y vào mảng Systolic trong lần thứ 3. Với z = ”1” thuật toán tiếp tục thực hiện phép toán AND 2 vectorbit với nhau để có được vectorbit mới. Sau đó vectorbit này được di chuyển tiếp đến hết hàng để kiểm tra xem có còn bit z = 1 nữa hay không. Số lượng transaction được đánh dấu là 1 xuất hiện ở vector bit này sẽ là độ hỗ trợ của tập {e,f} này. Qua hình vẽ có thể tấy độ hỗ trợ của {e,f} = 5 > minsup. Vậy {e,f} là tập phổ biến.



Hình 3.12. Lần di chuyển thứ 3 của mảng Systolic [4]

Dựa trên quy luật mảng Systolic, sau khi hoàn tất dòng thứ 3 của mảng Systolic, z bit của bộ xử lý d, e và f sẽ tương ứng là 0, 1 và 1.

Giá trị z hàng này chuẩn bị đầu vào cho tập phần tử kế tiếp sẽ là phần tử c (cef). Vòng lặp bên ngoài (while) đảm bảo quét tất cả tập phần tử trong mỗi dòng của mảng Systolic.

Quá trình được lặp đi lặp lại cho đến khi bit "z" được di chuyển đến vị trí $2^n - 1$ thì thuật toán được kết thúc. Việc thực hiện song song quá trình xử lý trên mảng systolic vừa giúp không gây trùng lắp dữ liệu và tăng tốc độ khai thác lên rất nhiều.

Nhìn vào hình trên, ta thấy rằng sau tiến trình đầu, bit "1" bên trái của mảng không thay đổi. Vì chỉ có log^{k+1} (cho mỗi k) bit bên phải của k được sử dụng vì vậy một counter được sử dụng để ngừng sự dịch chuyển sau khi kết thúc di chuyển vào bộ xử lý bên trái là "1" của mảng Systolic.

CHƯƠNG 4. XÂY DỰNG CHƯƠNG TRÌNH THỬ NGHIỆM

4.1. Môi trường cài đặt

Thử nghiệm thuật toán khai thác tập phổ biến dựa trên mảng Systolic (SABMA) được tiến hành trên cơ sở so sánh thời gian thực hiện so với thuật toán song song FP-Growth.

Thuật toán được cài đặt trên máy tính xách tay có cấu hình bộ vi xử lý (CPU) Intel core i5 3210M – 2.50GHz, Ram DDRIII 8.0GB, ổ cứng SSD 480GB, card đồ họa (VGA) Nvidia-Gefore 650GT 2GB. Các thuật toán được cài đặt bằng ngôn ngữ C# trên môi trường .Net 4.5 và sử dụng hệ điều hành Windows 8.1, 64-bit.

Các thuật toán được tiến hành trên các CSDL giao dịch dược tổng hợp từ UCI Machine Learning Repositary (https://archive.ics.uci.edu/ml/datasets.html) gồm Accident, Chess, Connect. Dưới đây là bảng tóm tắt các đặc điểm của CSDL được sử dụng:

Bảng 4.1. Cơ sở dữ liệu thử nghiệm

Tập dữ liệu	Kích thước	Số lượng giao dịch (Transactions)	Số lượng mục (items)
Accident	33.8Mb	340183	468
Chess	0.35Mb	3196	75
Connect	8.82Mb	67557	129

4.2. Kết quả của thuật toán

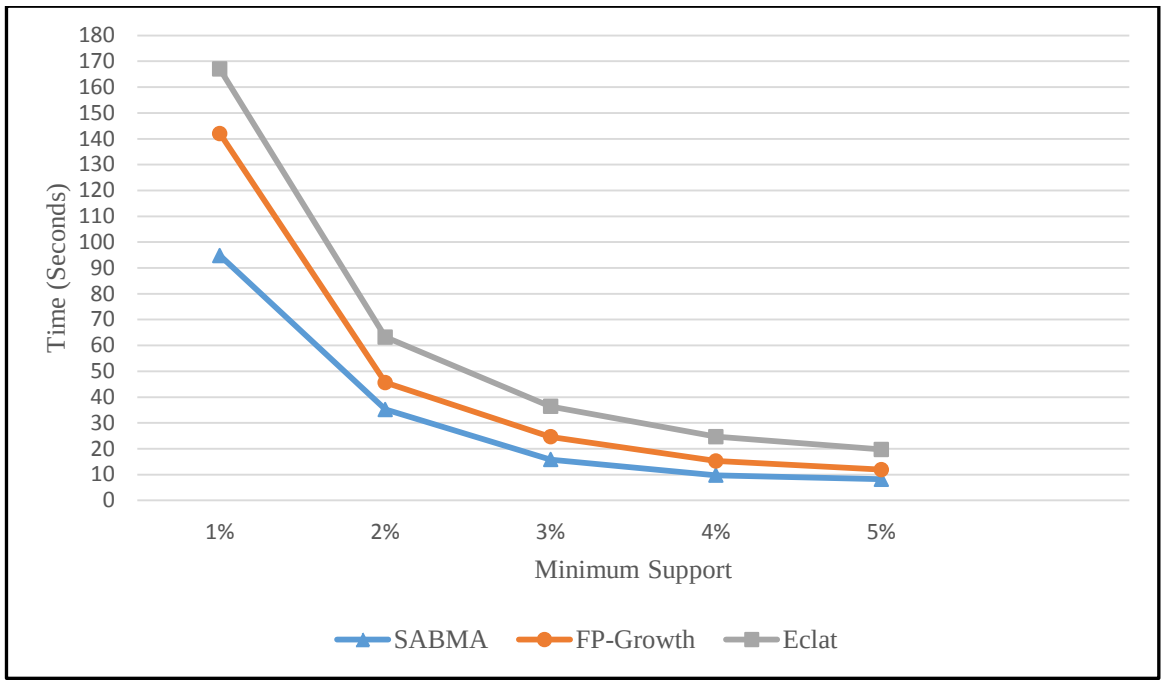
Chương trình thử nghiệm có thể thực hiện song song đồng thời trên nhiều nhân của CPU và GPU. Từng bộ xử lí trong mảng Systolic được đưa vào các nhân của máy tính đồng thời và liên tục được duyệt qua mảng theo các điều kiện được mô tả ở chương 3.

Trong chương trình thử nghiệm, thuật toán SABMA được cài đặt để chạy trên GPU, số lượng nhân xử lý song song của máy tính được sử dụng để thử nghiệm là 384 bộ vi xử lý trên GPU. Nhờ vào cấu trúc của mảng Systolic, thuật toán có khả

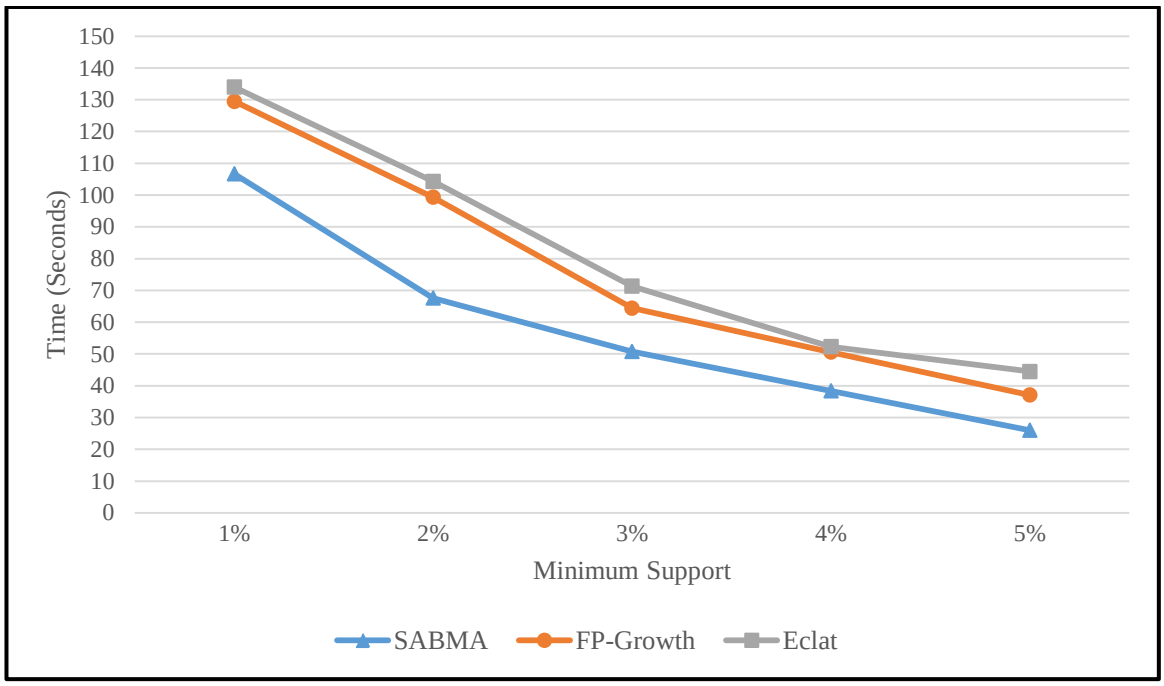
năng khai thác trên GPU xử lý đồng thời trên nhiều nhân. Ở cấu hình máy tính được sử dụng để thực nghiệm, số luồng xử lý song song trong trường hợp này có thể lên đến tối đa 184 input đầu vào đồng thời. Trong khi đó, thuật toán Eclat và FP-Growth vì không có cấu trúc dạng mảng systolic nên không thể tận dụng được sức mạnh của GPU đa luồng để khai thác.

Thuật toán SABMA có thời gian thực hiện ngắn hơn so với thuật toán FP-Growth và Eclat. Với việc tận dụng được tối đa phần cứng trong việc xử lý song song trên bộ vi xử lý (CPU) hoặc card đồ họa (GPU) thuật toán giúp tiết kiệm được chi phí thiết kế hệ thống song song để khai thác dữ liệu. Giới hạn của thuật toán phụ thuộc vào số lượng nhân xử lý trên máy tính. Với cấu hình máy càng mạnh thì thời gian xử lý trên mỗi máy của thuật toán SABMA càng nhanh.

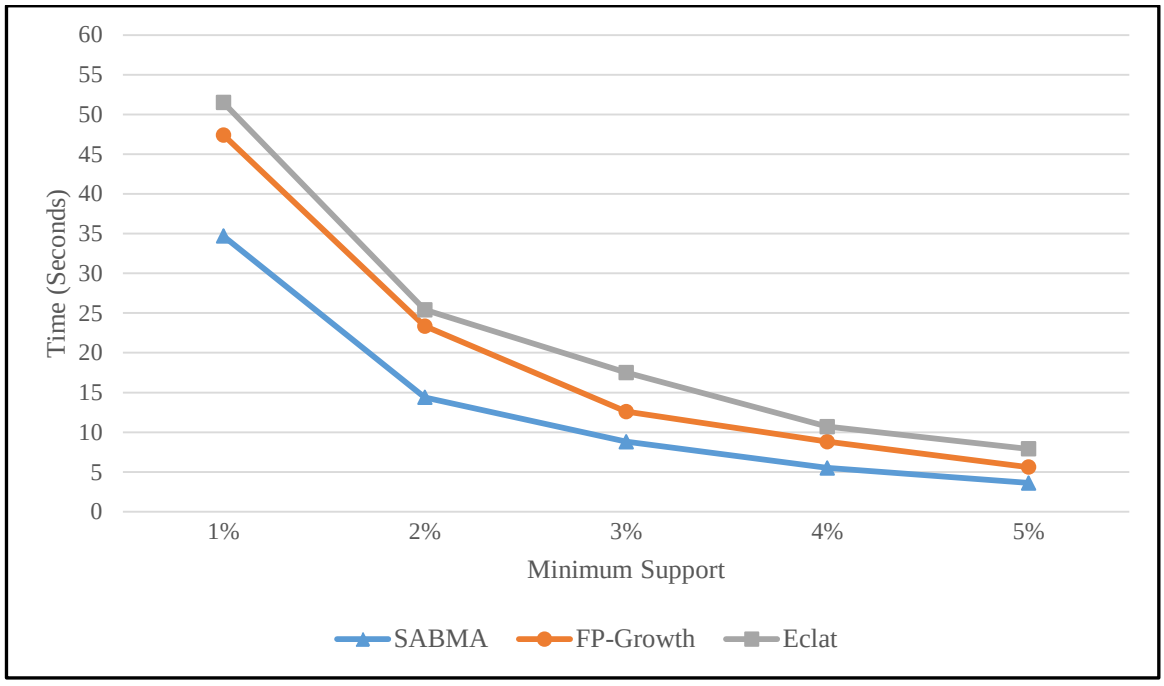
Các hình ở dưới thể hiện thời gian thực hiện của thuật toán SABMA, Eclat và FP-Growth trên các CSDL Accident, Chess, Connect với các độ phổ biến tối thiểu khác nhau. Thời gian thực hiện được tính từ khi các thuật toán nhận tham số đầu vào đến khi nhận được kết quả. Các biểu đồ cho thấy thời gian thực thi của thuật toán SABMA nhanh hơn thuật toán FP-Growth và Eclat.



Hình 4.1. Thời gian thực hiện trên tập dữ liệu Accident



Hình 4.2. Thời gian thực hiện trên tập dữ liệu Chess



Hình 4.3. Thời gian thực hiện trên tập dữ liệu Connect

4.3. Nhận xét

Cả 3 thuật toán đều cho kết quả đầu ra là tất cả các tập phổ biến với độ chính xác 100%. Vì kích thước không giống nhau của các CSDL được sử dụng để thử nghiệm nên chương trình không xác định độ phổ biến tối thiểu bằng một giá trị chính xác. Thay vào đó, độ phổ biến được xác định bằng tỉ lệ phần trăm số lượng giao dịch.

Vì thuật toán được thực thi theo phương pháp vét cạn (tìm kết quả bằng cách xem xét tất cả các phương pháp có thể) nên số phương án cần kiểm tra quá lớn. Ưu điểm của phương pháp này là luôn bảo đảm tìm ra kết quả chính xác và đòi hỏi ít bộ nhớ. Tuy nhiên, đối với các CSDL càng lớn thì thời gian thực thi của thuật toán càng dài. Để giảm thời gian thực thi thuật toán, trong chương trình thử nghiệm đã tiến hành kiểm tra một số điều kiện dừng để tối ưu hóa cho thuật toán như kiểm tra dữ liệu đầu vào trong khi duyệt qua mảng Systolic nếu y < minSup thì dữ liệu đó là không phổ biến.

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Kết luận

Luận văn đã đạt được một số kết quả cụ thể như sau:

- Luận văn đã trình bày tổng quan về KTDL để phát hiện tri thức, mục tiêu, quá trình, các phương pháp khai thác dữ liệu. Trình bày các ứng dụng, khó khăn và thách thức trong việc KTDL.
- Tìm hiểu về các vấn đề trong khai thác tập phổ biến để tìm luật kết hợp. Trình bày phân tích, đánh giá chi tiết về khái niệm, ưu, nhược điểm một số thuật toán khai thác tập phổ biến đã được phát triển.
- Dựa trên các phân tích, đánh giá về ưu, nhược điểm của các thuật toán khai thác song song tập phổ biến, luận văn đã nghiên cứu về một phương pháp khai thác song song tập phổ biến chỉ sử dụng một máy tính thay vì khai thác song song trên hệ thống lớn nhằm tối ưu hóa khả năng sử dụng phần cứng trong khai thác tập phổ biến. Thuật toán khai thác song song tập phổ biến dựa trên mảng Systolic đã được tìm hiểu, trình bày và thử nghiệm trên các CSDL giao dịch.
- Xây dựng và cài đặt chương trình thử nghiệm khai thác song song tập phổ biến dựa trên thuật toán song song SABMA để ứng dụng cho bài toán khai thác tập phổ biến. Ngoài ra, luận văn còn cài đặt thêm thuật toán FP-Growth, Eclat để so sánh đối chiếu. Kết quả cho thấy, thuật toán SABMA hiệu quả hơn về mặt thời gian thực thi thuật toán. Để đạt được hiệu quả như vậy, thuật toán SABMA đã tận dụng được khả năng của phần cứng máy tính để thực hiện khai thác song song tập phổ biến trên các vi xử lý trong cùng một máy tính. Tuy nhiên nếu dữ liệu đầu vào quá lớn thì thời gian thực thi thuật toán cũng tăng lên rất cao.

Hướng phát triển

Với kết quả nghiên cứu của luận văn, trong tương lai em sẽ tiếp tục nghiên cứu sâu hơn để tìm cách cải tiến của thuật toán này. Vì thuật toán sử dụng vectorbit để tìm tập phổ biến, hướng nghiên cứu để cải tiến sẽ áp dụng vectorbit động để nén các CSDL thưa. Về mặt chương trình thực nghiệm khai thác trên GPU, hiện tại chỉ có thể chạy trên các máy tính có card đồ họa do Nvidia sản xuất. Hướng phát triển của chương trình là sẽ tìm cách để có thể khai thác trên nhiều loại card màn hình khác nhau.

TÀI LIỆU THAM KHẢO

Tiếng Việt

[1] Đỗ Phúc (2006), *"Giáo trình khai thác dữ liệu"*, Nxb Đại học Quốc gia TP Hồ Chí Minh.

[2] Đoàn văn Ban, Nguyễn Mậu Hân, *"Xử lý song song và phân tán"*, NXB KH&KT, 2006.

[3] Giang Thị Thu Huyền, Luận văn "Nghiên cứu các luật kết hợp song song trong khai thác dữ liệu", 2010.

Tiếng Anh

[4] M.K. Sohrabi and A.A. Barforoush (2013), *"Parallel frequent itemset mining using systolic arrays"*, Knowledge-Based Systems, 37, 462–471.

[5] A. Javed, A. Khokhar, *"Frequent pattern mining on message passing Multi-processor systems", Distributed and Parallel Databases 16* (2004) 321–334.

[6] Bay Vo, Tuong Le, Frans Coenen, T.P Hong (2013), *"A hybrid approach for mining frequent itemsets"*, IEEE SMC'13, Manchester, UK, 4647-4651.

[7] Bay Vo, Bac Le, Thang N. Nguyen (2011), *"Mining frequent itemsets from multidimensional databases"*, ACIIDS 2011, Daegu, Korea, LNAI 6591, 177-186 (Springer).

[8] D. Chen, C. Lai, W. Hu, W.G. Chen, Y. Zhang, W. Zheng, *"Tree partition based parallel frequent pattern mining on shared memory systems"*, in: IEEE Parallel and Distributed Processing Symposium, 2006.

[9] K.M. Yu, J. Zhou, W.C. Hsiao, *Load balancing approach parallel algorithm for frequent pattern mining*, in: PaCT, 2007, pp. 623–631.

[10] M.K. Sohrabi, A.A. Barforoush, *"Efficient colossal pattern mining in high dimensional datasets"*, Knowledge Based Systems (2012)

[11] S.K. Tanbeer, C.F. Ahmed, B.-S. Jeong, *"Parallel and distributed algorithms for frequent pattern mining in large databases"*, IETE Technical Review (2010).