Thuật toán song song khai thác itemset lợi nhuận phổ biến Skyline

Nguyễn Mạnh Hùng[1], Nguyễn Thị Thùy Trâm[2]
[1] Viện CNTT & TT, HVKTQS, Hà Nội, Việt Nam
[2] Trung tâm Phát triển Công nghệ Thông Tin, ĐH Công nghệ Thông Tin - ĐH Quốc gia Tp.HCM, Việt Nam
Tác giả liên hệ: Nguyễn Mạnh Hùng, Manhhungk12@mta.edu.vn
Ngày nhận bài: 28/07/2023, ngày sửa chữa: 09/09/2023, ngày duyệt đăng: 26/09/2023
Định danh DOI: 10.32913/mic-ict-research-vn.v2023.n2.1233

Tóm tắt: Các itemset lợi nhuận phổ biến Skyline (SFUI) có thể cung cấp nhiều thông tin hữu ích hơn cho việc ra quyết định với việc xem xét cả hai yếu tố là tần suất xuất hiện và lợi nhuận của chúng. Kể từ khi bài toán khai thác itemset lợi nhuận phổ biến Skyline được Goyal V. và các cộng sự đề xuất vào năm 2015, đến nay đã có nhiều thuật toán tuần tự được đề xuất nhằm cải thiện hiệu suất khai thác, tuy nhiên hầu hết các thuật toán đều có hiệu suất kém khi khai thác các tập dữ liệu lớn phổ biến hiện nay. Trong bài báo này, chúng tôi đề xuất một thuật toán song song có tên là ParaSFUI-UF dựa trên thuật toán tuần tự SFUI-UF là một thuật toán khai thác itemset lợi nhuận phổ biến Skyline hiệu quả nhất hiện nay. Các kết quả thực nghiệm cho thấy thuật toán ParaSFUI-UF vượt trội so với thuật toán SFUI-UF.

Từ khóa: *Data mining, frequent itemset, high utility itemset, Skyline Frequent-Utility Itemset, multi-core, parallel processing*

Title: **Parallel Algorithm Exploits Skyline Common Interest Element Set**

Abstract: Skyline common-utility element sets (SFUIs) can provide more useful information for decision-making by considering both their frequency and their benefits. Since the Skyline utility-common element set mining problem was proposed by Goyal V. and colleagues in 2015, up to now, many sequential algorithms have been proposed to improve mining performance. However, most algorithms have poor performance when exploiting today's popular large data sets. In this paper, we propose a parallel algorithm called ParaSFUI-UF based on the sequential algorithm SFUI-UF, which is the most effective algorithm for exploiting the Skyline common-benefit element set today. Experimental results show that the ParaSFUI-UF algorithm outperforms the SFUI-UF algorithm.

Keywords: *Data mining, frequent itemset, high utility itemset, Skyline Frequent-Utility Itemset, multi-core, parallel processing*

I. GIỚI THIỆU

Khai thác tập lợi nhuận phổ biến Skyline (SFUIM) nhằm khắc phục hạn chế của khai thác tập phổ biến và khai thác tập lợi nhuận cao. Cụ thể, khác với khai thác tập phổ biến và khai thác tập lợi nhuận cao, để khai thác tập lợi nhuận phổ biến Skyline người sử dụng không cần phải thiết lập các tham số đầu vào là ngưỡng độ hỗ trợ tối thiểu minSup hay là ngưỡng lợi nhuận tối thiểu minUtil. Từ khi bài toán khai thác tập lợi nhuận phổ biến Skyline được Goyal V. và các cộng sự đề xuất [1] vào năm 2015, đến nay đã có một số thuật toán được công bố nhằm nâng cao hiệu suất khai thác. Tuy nhiên, do không gian các itemset ứng viên của bài toán có độ phức tạp hàm mũ cho nên bài toán khai thác tập lợi nhuận phổ biến Skyline vẫn là một thách thức, đặc biệt khi xử lý dữ liệu lớn. Các thuật toán tuần tự [2-5] hoạt động tốt trên các tập dữ liệu nhỏ, tuy nhiên hầu hết chúng không thể hoàn thành nhiệm vụ khai thác trên các bộ dữ liệu quy mô lớn trong thời gian chấp nhận được do giới hạn về khả năng tính toán.

Mặt khác, tính toán song song [6,7], được phát triển để xử lý các tập dữ liệu lớn, cung cấp một giải pháp tiềm năng cho bài toán này. Đối với các bài toán khai thác tập phổ biến và bài toán khai thác tập lợi nhuận cao thì đã có khá nhiều thuật toán song song được công bố [11-14]. Tuy nhiên, tính đến hiện nay để giải bài toán SFUIM chỉ có các thuật toán tuần tự được công bố, trong đó thuật toán tuần tự SFUI-UF được cho là hiệu quả nhất.

Trong bài báo này chúng tôi đề xuất thuật toán song song ParaSFUI-UF sử dụng ưu điểm của tính toán song song để nâng cao hiệu suất khai thác. Nội dung tiếp theo của bài báo được tổ chức như sau: mục 2 trình bày các nghiên cứu liên quan, mục 3 thuật toán song song ParaSFUI-UF và thử

nghiệm đánh giá thuật toán, mục 4 kết luận.

II. CÁC NGHIÊN CỨU LIÊN QUAN

1. Khai thác itemset lợi nhuận - phổ biến Skyline

Cho I là một tập hữu hạn các item $I = i_1, i_2, \ldots, i_m$ và $X \subseteq I$ là một itemset, nếu X chứa k item thì X được gọi là *k-itemset*. Một cơ sở dữ liệu giao dịch (CSDL) D chứa n giao dịch: $D = T_1, T_2, \ldots, T_n$. Mỗi giao dịch $T_i \in D(1 \leq i \leq n)$ chứa một tập con các item thuộc tập I. Tần suất xuất hiện của tập X, ký hiệu là $f(X)$, là số lượng giao dịch trong D có chứa tập X.

Lợi nhuận trong của một item i_p trong giao dịch T_i, ký hiệu là $q(i_p, T_i)$, là số lượng của item i_p xuất hiện trong giao dịch T_i. Lợi nhuận ngoài của item i_p, kí hiệu là $prof(i_p)$, là đơn vị lợi nhuận của item i_p (là giá bán hoặc lợi nhuận khi bán được một item i_p). Lợi nhuận của item ip trong giao dịch T_i, kí hiệu là $u(i_p, T_i) = q(i_p, T_i) * prof(i_p)$.

Lợi nhuận của itemset X trong giao dịch T_i chứa tập X, ký hiệu là $u(X, T_i) = \sum_{i_p \in X} u(i_p, T_i)$. Lợi nhuận của itemset X trong CSDL D, ký hiệu là $u(X) = \sum_{X \subseteq T_i} u(X, T_i)$. Lợi nhuận giao dịch của giao dịch T_i, ký hiệu là $TU(T_i) = u(T_i, T_i)$.

Vì lợi nhuận của itemset không có tính chất đóng, nên Liu và cộng sự [8] đã đề xuất một khái niệm gọi là lợi nhuận giao dịch có trọng số, ký hiệu là $TWU(X) = \sum_{X \subseteq T_i} TU(T_i)$. TWU được sử dụng để cắt tỉa không gian các itemset ứng viên trong các thuật toán khai thác itemset lợi nhuận cao. Ví dụ CSDL D trong Bảng I và lợi nhuận ngoài của từng item trong Bảng II.

Bảng I
CSDL GIAO DỊCH

TID	Transaction	TU
T1	(A2, 2), (A4, 1)	9
T2	(A1, 2), (A2, 1), (A3, 2), (A4, 1), (A5, 1)	25
T3	(A2, 2), (A4, 2),(A5, 1)	18
T4	(A4, 2), (A5, 1)	14
T5	(A1, 4), (A2, 2), (A4, 1), (A5, 2)	41
T6	(A2, 4), (A5, 2)	16
T7	(A3, 5)	5

Bảng II
BẢNG LỢI NHUẬN CỦA CÁC ITEM

item	A1	A2	A3	A4	A5
Lợi nhuận	6	2	1	5	4

Xem xét một itemset dựa trên 2 yếu tố là tần suất xuất hiện và lợi nhuận: Một tập X là trội hơn tập Y nếu $f(X) \geq f(Y)$ và $u(X) \geq u(Y)$, **HOẶC** $f(X) \geq f(Y)$ và $u(X) \geq u(Y)$, ký hiệu là: $X \succ Y$.

Một itemset X trong CSDL D là *itemset lợi nhuận phổ biến Skyline* (SFUI) nếu X không bị trội hơn bởi bất kỳ một itemset nào khác.

Bài toán khai thác itemset lợi nhuận phổ biến Skyline (SFUIM): Cho một CSDL D với bảng lợi nhuận ngoài tương ứng của từng item, khai thác itemset lợi nhuận phổ biến Skyline(SFUIM) chính là việc đi tìm tất cả các tập SFUIs trong CSDL D.

Điểm khác biệt của bài toán khai thác itemset lợi nhuận phổ biến Skyline so với các bài toán khai thác itemset phổ biến và bài toán khai thác itemset lợi nhuận cao đó là khi thực hiện khai thác các itemset lợi nhuận phổ biến Skyline, người sử dụng không cần phải xác định các giá trị ngưỡng về độ hỗ trợ tối thiểu (minSup) và lợi nhuận tối thiểu (minUtil).

Như đã giới thiệu ở trên, Goyal V. và các cộng sự đã đề xuất bài toán SFUIM và thuật toán SKYMINE [1] sử dụng cấu trúc UP-Tree và kỹ thuật tăng trưởng mẫu để khai thác các tập SFUIs.

Sau đề xuất của Goyal V., đã có một số thuật toán được công bố nhằm nâng cao hiệu năng khai thác SFUIs: Trong [2, 10] các tác giả đã đề xuất cấu trúc mảng umax và sử dụng cấu trúc danh sách lợi nhuận UL [9] để xây dựng thuật toán SFU-Miner khai thác các tập SFUIs. So với thuật toán SKYMINE, thuật toán SFU-Miner ưu điểm hơn nhờ sử dụng cấu trúc umax để cắt tỉa không gian các itemset ứng viên và cấu trúc UL có thể xác định trực tiếp SFUIs mà không cần tạo các tập ứng viên.

Lin và công sự [3] đã đề xuất cấu trúc utility-max và sử dụng cấu trúc danh sách lợi nhuận UL để xây dựng 02 thuật toán duyệt theo chiều sâu SKYFUP-D và thuật toán duyệt theo chiều rộng SKYFUP-B khai thác các tập SFUIs. SKYFUP hiệu quả hơn SKYMINE và SFU-Miner.

Nguyen và công sự [4] đã đề xuất cấu trúc danh sách lợi nhuận mở rộng EUL, so sánh với UL, EUL bổ sung thêm 02 trường: lợi nhuận của itemset và lợi nhuận của item cuối cùng trong itemset. Trên cơ sở cấu trúc EUL, Nguyen và công sự đề xuất thuật toán FSKYMINE khai thác SFUIs. Thuật toán FSKYMINE hiệu quả hơn so với thuật toán được đề xuất trong [2, 10].

Wei Song và cộng sự [5] đã đề xuất một số cấu trúc và chiến lược để nâng cao hiệu quả khai thác các tập SFUIs:

- Thứ nhất, đề xuất cấu trúc mảng lợi nhuận tối đa, ký hiệu là MUA và sử dụng danh sách lợi nhuận UL kết hợp với MUA để cắt tỉa các itemset và các mở rộng không tiềm năng.

- Thứ 2, sử dụng TWU để lọc các item không phải là SFUIs. Thứ 3, đề xuất khái niệm lợi nhuận tối thiểu của tập mở rộng, ký hiệu là MUE, để cắt tỉa không gian các itemset ứng viên. Trên cơ sở các kỹ thuật được đề xuất ở

trên, Wei Song và cộng sự đã đề xuất thuật toán SFUI-UF. SFUI-UF là thuật toán khai thác các tập SFUIs hiệu quả nhất hiện nay.

2. Một số khái niệm cơ sở

Định nghĩa 2.1 (Itemset lợi nhuận phổ biến tiềm năng Skyline): [2]. Một itemset X là itemset lợi nhuận phổ biến tiềm năng (PSFUI) nếu không tồn tại tập Y bất kỳ sao cho $f(Y) = f(X)$ và $u(Y) \geq u(X)$.

Trong [2], Pan và các cộng sự đã chứng minh tất cả các SFUIs đều là PSFUIs, vì vậy nhiều thuật toán khai thác các tập SFUIs có thể thực hiện theo phương pháp 2 bước: bước 1, tìm ra tất cả các tập PSFUIs, và sau đó tiến hành lọc PSFUIs để tìm ra các tập SFUIs.

Định nghĩa 2.2 (mảng MUA): [5]. Gọi fmax là tần suất tối đa của tất cả các itemset trong CSDL D và MUA là một mảng có kích thước bằng fmax. Giá trị của item thứ $f(1 \leq f \leq fmax)$ trong mảng MUA được định nghĩa như sau: $MUA(f) = max\{u(X)|f(X) \geq f\}$

Trong đó X là itemset bất kỳ trong CSDL D. Mảng MUA tương tự như mảng utilmax[3] nhưng khác nhau về kích thước. Cụ thể, theo [3], utilmax có kích thước bằng số lượng giao dịch của CSDL D, trong khi đó MUA có kích thước bằng $fmax$. Trong [5] Wei Song và cộng sự đã chứng minh $|MUA| \leq |utilmax|$, trong đó $|MUA|$ và $|ultimax|$ tương ứng là kích thước của mảng MUA và mảng utilmax.

Định lý 2.1: Gọi X là một itemset. Nếu tổng của tất cả các giá trị iutil và rutil trong X. UL nhỏ hơn $MUA(f(X))$, thì X và tất cả các phần mở rộng của X không phải là SFUIs [5].

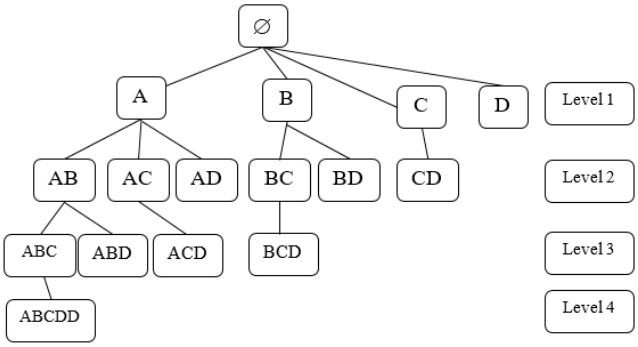
Định nghĩa 2.3 (Lợi nhuận tối thiểu của SFUIs):

[5]. Xét CSDL D, lợi nhuận tối thiểu của SFUIs, ký hiệu là MUS, trong D là lợi nhuận tối đa của các item đơn lẻ có tần suất cao nhất trong D. Cụ thể MUS được định nghĩa như sau: $MUS = max\{u(i_p)|f(i_p) = fmax\}$,

Trong đó i_p là một item trong D và fmax là tần suất lớn nhất của tất cả các tập chứa một item trong D.

Định lý 2.2: Gọi i_p là tập một item. Nếu $TWU(i_p) \leq MUS$ thì i_p và tất cả các itemset có chứa i_p không phải là $SFUIs$. Như vậy, khi itemset chứa một item có TWU thấp hơn MUS thì itemset này và tất cả các tập chứa nó có thể được cắt tỉa một cách an toàn. Trong thuật toán SFUI-UF, MUS được đặt là các giá trị ban đầu của mỗi item trong MUA [5].

Định nghĩa 2.4 (Lợi nhuận tối thiểu của phần mở rộng): Cho X là một itemset, i_p là một item, và tập $X \cup i_p$ là một tập mở rộng của itemset X theo item i_p. Khi đó, lợi nhuận tối thiểu của tập mở rộng $X \cup i_p$, ký hiệu là $MUE(X \cup i_p)$, được định nghĩa bằng $u(X)$, cụ thể như sau: $MUE(X \cup i_p) = u(X)$.



Hình 1. Không gian tìm kiếm của thuật toán SFUI-UF.

Ví dụ, xét CSDL trong Bảng I và Bảng II, ta có: $MUE(A1A4) = u(A1) = 36; MUE(A2A4) = u(A2) = 22$

Định lý 2.3: Cho X là một itemset và itemset Y là một mở rộng của X. Nếu $u(Y) \leq MUE(Y)$, Y không phải là SFUI [5].

Thuật toán SFUI-UF[5] dựa trên các định lý 1, 2, 3 để cắt tỉa không gian các itemset ứng viên. Nội dung tiếp theo, bài báo đề xuất thuật toán song song khai thác SFUIs.

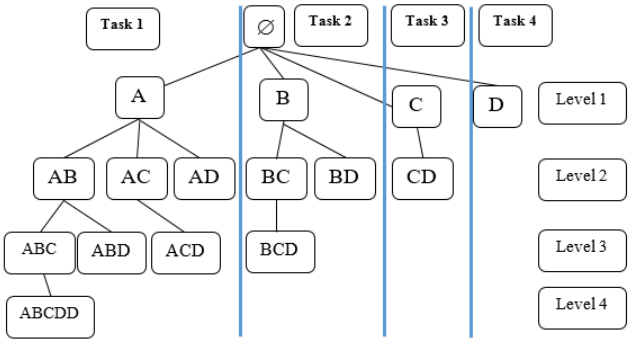
III. THUẬT TOÁN SONG SONG KHAI THÁC ITEMSET LỢI NHUẬN - PHỔ BIẾN SKYLINE

Ngày nay ngay cả các máy tính cá nhân cũng đã có bộ xử lý đa nhân (multi-core) và việc sử dụng lập trình song song nhằm phát huy tối đa sức mạnh của bộ xử lý đa nhân cũng khá phổ biến trong các ứng dụng. Lập trình song song mang lại nhiều lợi ích như: nâng cao hiệu năng xử lý và giúp giải quyết được các bài toán có kích thước lớn. Tuy nhiên, việc phát triển các thuật toán song song hoặc cải tiến các thuật toán tuần tự hiện có để có thể thực hiện song song sẽ gặp phải một số thách thức như: vấn đề về đồng bộ, vấn đề về truyền thông và vấn đề về cân bằng tải.

Nhằm phát huy tốt nhất các ưu điểm của các bộ xử lý đa nhân trên các máy tính ngày nay, trong bài báo này chúng tôi đề xuất một thuật toán song song khai tác tập lợi nhuận phổ biến Skyline có tên là ParaSFUI-UF. Thuật toán ParaSFUI-UF là mở rộng của thuật toán SFUI-UF theo hướng có thể thực hiện song song việc khám phá không gian các itemset ứng viên.

Thuật toán SFUI-UF khám phá các tập SFUIs theo phương pháp tìm kiếm theo chiều sâu trên cây liệt kê các itemset bắt đầu từ nút gốc rỗng. Hình 1 minh hoạ một cây liệt kê với 04 item A, B, C, D.

Phương pháp ParaSFUI-UF thực hiện song song công việc khám phá không gian các itemset ứng viên theo phương pháp chia để trị: chia không gian các itemset ứng



Hình 2. Phân hoạch không gian các itemset ứng viên trong ParaSFUI-UF.

viên thành các không gian con, mỗi không gian con sẽ gồm MP item ở mức Level1 trên cây liệt kê, MP là tham số được tính toán theo số lượng nhân của bộ xử lý đa nhân; Việc khám phá trên mỗi không gian con được gọi là một tác vụ (task); Các tác vụ sẽ được xử lý song song trên các nhân của bộ xử lý đa nhân (multi-core). Hình 2 minh họa việc chia không gian các itemset ứng viên thành các không gian con, mỗi không gian con gồm 1 item ở Level1 và các mở rộng của nó ở mức tiếp theo.

Phương pháp ParaSFUI-UF gồm 2 giai đoạn:

Giai đoạn 1, từ dòng 1 đến dòng 13 trong Thuật toán 1 nhằm xác định danh sách lợi nhuận UL của các tập 1 item, mảng MUA.

Giai đoạn 2, từ dòng 14 đến dòng 19 trong Thuật toán 1. Chi tiết giai đoạn 2 được thực hiện trong Thuật toán 2 và Thuật toán 3. Hàm tail(X) trong tham số đầu vào của Thuật toán 2 được định nghĩa là tất cả các item đứng sau tất cả các item trong tập X theo một thứ tự Δ nào đó. Điểm thay đổi chính của thuật toán ParaSFUI-UF so với thuật toán SFUI-UF nằm ở vòng lặp for từ dòng 1 đến dòng 13 của Thuật toán 2 và vòng lặp for từ dòng 1 đến dòng 9 của Thuật toán 3. Cụ thể, trong Thuật toán 2, mỗi MP item trong tail(X) sẽ được khám phá song song theo phương pháp tìm kiếm theo chiều sâu trong một tác vụ trên một nhân của bộ xử lý. Hình 2 cho thấy các không gian con là độc lập với nhau và hợp của chúng sẽ bằng không gian tổng thể của bài toán. Mỗi tác vụ sẽ giữ riêng cho mình X, tail(X) và mảng MUA.

Trong thuật toán 3, thực hiện song song hóa việc tìm ra các tập không phải là itemset lợi nhuận phổ biến Skyline từ các tập tiềm năng S-PSFUIs. Cụ thể, mỗi MP item trong S-PSFUIs sẽ được xử lý song song trong các tác vụ.

1. Vấn đề đồng bộ kết quả khai thác

Trong thuật toán 2, các tác vụ được thực thi song song trên các nhân của bộ xử lý và chúng phải đồng bộ hóa thao tác ghi kết quả mỗi khi tìm ra được một tập tiềm năng PSFUI. Cụ thể trong dòng lệnh 6 và dòng lệnh 7, khi phát hiện ra tập X' là itemset lợi nhuận phổ biến tiềm năng Skyline, thao tác ghi tập X' ra tập kết quả và thao tác xóa các tập Y không phải là tập tiềm năng ra khỏi tập kết quả phải được đồng bộ giữa các tác vụ.

Algorithm 1 ParaSFUI-UF

Input CSDL D cùng với bảng lợi nhuận ngoài của từng item

Output Tất cả các tập SFUIs

1: #Giai đoạn 1
2: Scan D once to calculate $fmax$ and MUS;
3: Delete item with TWU values lower than MUS;
4: Calculate the TWU values of the remaining items;
5: **for all** $T_d \in D$ **do**
6: Sort items in T_d using TWU - ascending oder;
7: **for all** $i \in T_d \in D$ **do**
8: Update i.UL;
9: **end for**
10: **end for**
11: **for** $j = 1$ to $fmax$ **do**
12: $MUA(j) = MUS$;
13: **end for**
14: #Giai đoạn 2
15: Shared S-PSFUI=∅;
16: Shared S-SFUI=∅;
17: ParaP-Miner(∅,tail(∅),MUA);
18: ParaS-Miner(S-PSFUI);
19: **return** $all\ SFUIs$;

Algorithm 2 ParaP-Miner

Input X, $tail(X)$, MUA

Output Tất cả các tập PSFUIs là mở rộng của tập X

1: **for all** Parallel MP item $i \in tail(X)$ **do**
2: $X' = X \cup i$;
3: **if** $u(X') \geq MUE(X')$ **then**
4: **if** $u(X') \geq MUA(f(X'))$ **then**
5: **Update** MUA according to $f(X')$ and $u(X')$;
6: **Synchronize**: $S - PSFUI = S - PSFUI \cup X'$;
7: **Synchronize**: Remove Y from $S - PSFUI$ if $(f(Y) == f(X'))$;
8: **end if**
9: **if** $\sum d \in X.tids(iutil(X', T_d) + rutil(X', T_d)) \geq MUA(f(X'))$ **then**
10: $P - Mine(X', tail(X'), MUA)$;
11: **end if**
12: **end if**
13: **end for**

Algorithm 3 ParaS-Miner

Input S-PSFUIs
Output Tất cả các tập SFUIs

1: **for all** Parallel MP itemset X in S-PSFUI **do**
2: **for all** itemset Y in S-PSFUI **do**
3: **if** $f(X) \geq f(Y)$ and $u(X) \geq u(Y)$ or $f(X) \geq f(Y)$ and $u(X) \geq u(Y)$ **then**
4: **Synchronize**: Remove Y from $S - PSFUI$;
5: **end if**
6: **end for**
7: **end for**
8: $S - SFUI = S - PSFUI$;
9: **return** $S - SFUI$;

2. Vấn đề cân bằng tải

Trong các thuật toán song song, việc xử lý tốt vấn đề cân bằng tải sẽ cải thiện được hiệu năng tổng thể của chương trình. Việc cân bằng tải bao gồm hai yếu tố: thứ nhất, giảm thiểu dữ liệu phải truyền thông giữa các tác vụ; thứ 2, tối thiểu tổng thời gian nhàn rỗi của các bộ xử lý (các nhân trong bộ xử lý đa nhân). Bộ xử lý đa nhân được thiết kế theo mô hình xử lý song song sử dụng bộ nhớ chia sẻ do đó người lập trình sẽ không phải quan tâm đến vấn đề truyền thông dữ liệu giữa các tác vụ. Hơn nữa, các nhân của một bộ xử lý đa nhân thường có khả năng tính toán như nhau, vì vậy để cân bằng tải, thuật toán phải giao khối lượng công việc như nhau cho mỗi nhân xử lý nhằm đảm bảo các nhân luôn ở trong trạng thái xử lý công việc trước khi thuật toán kết thúc. Các tác vụ trong các thuật toán song song sẽ được quản lý trong một task-pool. Sau đó, các tác vụ này sẽ được xử lý song song theo chiến lược tác vụ đến trước sẽ được xử lý trước trong các nhân của bộ xử lý. Như vậy, trên bộ xử lý đa nhân, nếu thời gian xử lý đồng bộ giữa các tác vụ nhỏ, để thực hiện cân bằng tải chúng ta nên chia tác vụ có khối lượng tính toán nhỏ. Tuy nhiên, mỗi tác vụ trong task-pool sẽ được cấp phát bộ nhớ để lưu trữ các biến riêng của chúng, do đó nếu kích thước của task-pool quá lớn sẽ gây ra hiện tượng tràn bộ nhớ. Để tối thiểu hóa kích thước của task-pool, chúng ta có thể chọn số tác vụ bằng đúng số nhân của bộ xử lý. Tuy nhiên, điều này lại có thể ảnh hưởng đến vấn đề cân bằng tải. Cụ thể, trên Hình 2 cho thấy các tác vụ ở phía bên trái (ứng với các item ở đầu của tail(X)) sẽ có không gian các itemset ứng viên lớn hơn các tác vụ phía bên phải, do đó những nhân xử lý các tác vụ ở phía bên trái sẽ có thể hoàn thành công việc chậm hơn so với những nhân thực hiện các tác vụ ở phía bên phải.

3. Thử nghiệm

Các tác giả trong [5], khi thực nghiệm thuật toán SFUI-UF, đã sắp xếp các item theo thứ tự giảm dần của giá trị TWU và duyệt các item trên cây liệt kê theo thứ tự từ trái sang phải để đánh giá thuật toán SFUI-UF. Tuy nhiên, trong thuật toán song song, thứ tự tìm kiếm của các item sẽ bị thay đổi và không nhất quán giữa các lần chạy khác nhau. Vì vậy, để đánh giá ảnh hưởng của thứ tự tìm kiếm đến kích thước của không gian các itemset ứng viên và tốc độ thực hiện của thuật toán, trong phần thực nghiệm này chúng tôi sẽ thực hiện 02 nội dung: Nội dung 1, các item ở Level 1 của cây liệt kê được tổ chức thành một hàng đợi xoay vòng theo thứ tự giảm dần của giá trị TWU, đánh giá thuật toán SFUI-UF bằng cách thay đổi vị trí xuất phát: TH1: Xuất phát từ đầu hàng đợi (từ item bên trái cùng trên cây liệt kê); TH2: Xuất phát từ vị trí cách đầu hàng đợi một khoảng bằng 1/3 chiều dài hàng đợi; TH3: Xuất phát từ giữa hàng đợi; TH4: Xuất phát từ vị trí cách đầu hàng đợi một khoảng bằng 2/3 chiều dài hàng đợi; TH5: Xuất phát từ cuối hàng đợi. Nội dung 2, so sánh phương pháp ParaSFUI-UF với thuật toán SFUI-UF ở 2 trường hợp điển hình là TH1 và TH5, các trường hợp khác hoàn toàn tượng tự.

Mã nguồn thuật toán SFUI-UF được lấy từ công bố của nhóm tác giả [5] trong bộ thư viện nguồn mở SPMF[15].

Phương pháp ParaSFUI-UF được cài đặt và đánh giá trên môi trường Java 17 sử dụng fork-join framework để chia không gian các itemset ứng viên thành các không gian con theo phương pháp đệ quy. Số lượng tác vụ con được thực hiện song song được tính toán tự động theo số nhân của bộ xử lý trên máy tính. Trong quá trình tiến hành thực nghiệm, chúng tôi thấy rằng để cân bằng tải, và không bị tràn bộ nhớ do kích thước hàng đợi lớn thì số lượng tác vụ được định nghĩa bằng 2 lần số nhân của bộ xử lý. Các thực nghiệm chúng tôi đã thực hiện trên máy tính có cấu hình 8-core 3.00GHz, 8GB RAM chạy hệ điều hành Windows 10 phiên bản 64 bit.

Các cơ sở dữ liệu sử dụng gồm: mushroom_utility (viết tắt là M_Util), chess_utility (viết tắt là C_Util) và connect_utility (viết tắt là Con_Util) được lấy từ [15].

Kết quả thực nghiệm nội dung 1 được trình bày trong Bảng III và Bảng IV:

Thời gian chạy 05 trường hợp trong Bảng IV là kết quả trung bình của 20 lần chạy. Từ kết quả thực nghiệm thuật toán SFUI-UF trên 03 CSDL ở Bảng III và Bảng IV, chúng ta thấy kích thước của không gian các itemset ứng viên và thời gian thực hiện của thuật toán có xu hướng giảm khá nhanh khi vị trí bắt đầu duyệt dần về cuối hàng đợi. Kết quả thực nghiệm nội dung 2 được trình bày trong 02 bảng: Bảng V và Bảng VI như sau:

Các kết quả đánh giá thời gian thực hiện và không gian các itemset ứng viên của thuật toán ParaSFUI-UF trong các Bảng V và Bảng VI là các giá trị trung bình của 20 lần chạy chương trình. Kết quả thực nghiệm trong Bảng V và

Bảng III
SO SÁNH KHÔNG GIAN CÁC ITEMSET ỨNG VIÊN CỦA SFUI-UF

Dataset	#SFUIs	TH1	TH2	TH3	TH4	TH5
M_Util	17	22,710	12,820	9,236	4,506	4,743
C_Util	35	915,101	269,561	117,050	86,364	81,301
Con_Util	46	4,349,427	4,952,510	1,968,260	685,520	654,885

Bảng IV
SO SÁNH KẾT QUẢ THỜI GIAN THỰC HIỆN CỦA SFUI-UF (MS)

Dataset	TH1	TH2	TH3	TH4	TH5
M_Util	1,588	1,500	1,322	998	1,012
C_Util	28,273	17,732	12,708	12,524	11,925
Con_Util	4,141,415	4,267,812	2,821,565	1,810,984	1,762,345

Bảng V
SO SÁNH KHÔNG GIAN CÁC ITEMSET ỨNG VIÊN

Dataset	SFUI-UF (TH1)	SFUI-UF (TH5)	ParaSFUI-UF	
	#PSFUIs	#PSFUIs	#SFUIs	#PSFUIs
M_Util	22,710	4,743	17	11,650
C_Util	915,101	81,301	35	17,981
Con_Util	4,349,427	654,885	46	54,081

Bảng VI
SO SÁNH THỜI GIAN THỰC HIỆN(MS)

Dataset	SFUI-UF (TH1)	SFUI-UF (TH5)	ParaSFUI-UF
M_Util	1,588	1,012	637
C_Util	28,273	11,925	788
Con_Util	4,141,415	1,762,345	43,439

Bảng VI cho thấy thuật toán ParaSFUI-UF vượt trội so với thuật toán SFUI-UF nguyên bản [5] và thuật toán SFUI-UF thực hiện theo TH5.

IV. KẾT LUẬN

Trong bài báo này, chúng tôi đã đề xuất thuật toán song song ParaSFUI-UF trên cơ sở mở rộng thuật toán hiệu quả nhất hiện nay SFUI-UF. Thuật toán ParaSFUI-UF đã thực hiện chia không gian các itemset ứng viên SFUIs thành các không gian con và thực hiện khám phá các không gian con một cách song song trên các nhân của bộ xử lý đa nhân. Phần thực nghiệm đã tiến hành 2 nội dung đánh giá: Thứ nhất là đánh giá sự ảnh hưởng của thứ tự tìm kiếm đến kích thước của không gian các itemset ứng viên và tốc độ thực hiện của thuật toán SFUI-UF; Thứ hai là đánh giá, so sánh phương pháp ParaSFUI-UF so với thuật toán SFUI-UF: kết quả cho thấy, phương pháp ParaSFUI-UF có hiệu quả vượt trội so với SFUI-UF.

TÀI LIỆU THAM KHẢO

[1] Vikram Goyal, Ashish Sureka, Dhaval Patel, "Efficient skyline itemsets mining", *C3S2E '15: Proceedings of the Eighth International C* Conference on Computer Science & Software Engineering* (2015), 119–124.

[2] Pan Jeng-Shyanga, Lin Jerry Chun-Weib, Yang Lub, Fournier-Viger Philippec, Hong Tzung-Peid, "Efficiently mining of skyline frequent-utility patterns", *Intelligent Data Analysis*, vol. 21, no. 6 (2017), 1407-1423.

[3] Jerry Chun-Wei Lin, Lu Yang, Philippe Fournier-Viger, Tzung-Pei Hong, "Mining of skyline patterns by considering both frequent and utility constraints", *Engineering Applications of Artificial Intelligence*, Volume 77 (2019), 229-238.

[4] Hung Manh Nguyen; Anh Viet Phan; Lai Van Pham: FSKYMINE, "A Faster Algorithm For Mining Skyline Frequent Utility Itemsets", *Proceedings of the 6th NAFOSTED Conference on Information and Computer Science*, (2019), 251–255.

[5] Wei Song, Chuanlong Zheng, Philippe Fournier-Viger, "Mining Skyline Frequent-Utility Itemsets with Utility Filtering", *18th Pacific Rim International Conference on Artificial Intelligence*, PRICAI (2021), Part I ,411–424.

[6] Bertil Schmidt, Jorge Gonzalez-Martinez, Christian Hundt, Moritz Schlarb, Parallel Programming: Concepts and Practice, Morgan Kaufmann (2017).

[7] Julian Shun: Shared-Memory Parallelism Can Be Simple, Fast, and Scalable, Association for Computing Machinery and Morgan & Claypool (2017).

[8] Ying Liu, Wei-keng Liao, Alok Choudhary, "A two-phase algorithm for fast discovery of high utility itemsets", *Advances in Knowledge Discovery and Data Mining, 9th Pacific-Asia Conference*, PAKDD (2005), 689–695.

[9] Mengchi Liu, Junfeng Qu: "Mining high utility itemsets without candidate generation", *21st ACM International Conference on Information and Knowledge Management*, (2012), 55–64.

[10] Jerry Chun-Wei Lin, Lu Yang, Philippe Fournier-Viger, Siddharth Dawar, Vikram Goyal, Ashish Sureka, and Bay Vo, "A more efficient algorithm to mine skyline frequent-utility patterns", *In International Conference on Genetic and Evolutionary Computing*, 2017, 127–135.

[11] Bay Vo, Loan T. T. Nguyen, Trinh D. D. Nguyen, Philippe Fournier-Viger, Unil Yun, "A Multi-Core Approach to Efficiently Mining High-Utility Itemsets in Dynamic Profit Databases", *IEEE Access* (2020), 85890 – 85899.

[12] Mohammed J. Zaki, "Parallel Sequence Mining on Shared-

Memory Machines", *Journal of Parallel and Distributed Computing*, Volume 61, Issue 3 (2001), 401-426.
[13] Krishan Kumar Sethi, Dharavath Ramesh , Damodar Reddy Edla: P-FHM+, "Parallel high utility itemset mining algorithm for big data processing", *Procedia Computer Science*, Volume 132 (2018), 918-927.
[14] O. Jamsheela, G. Raju, "Parallelization of Frequent Itemset Mining Methods with FP-tree: An Experiment with PrePost+ Algorithm", *The International Arab Journal of Information Technology*, Vol. 18, No. 2(2021), 208-231.
[15] An Open-Source Data Mining Library: http://www.philippe-fournier-viger.com/spmf/

Nguyễn Mạnh Hùng nhận bằng Tiến sĩ Bảo đảm toán học cho Máy tính và Hệ thống tính toán năm 2004 tại Trung tâm Tính toán, Viện Hàn lâm Khoa học, Liên bang Nga. Hiện đang công tác tại Viện CNTT&TT, Học viện KTQS.
Lĩnh vực nghiên cứu: Khai phá dữ liệu; Tính toán song song; Khoa học dữ liệu.
Email: ManhHungk12@mta.edu.vn

Nguyễn Thị Thùy Trâm tốt nghiệp Đại học chuyên ngành Công nghệ Thông tin, Trường ĐH Công nghệ Thông tin – ĐH Quốc Gia - Thành phố Hồ Chí Minh. Tốt nghiệp Thạc sĩ Khoa học máy tính năm 2020 của Học viện Kỹ thuật Quân sự. Hiện nay công tác tại Trung tâm Phát triển Công nghệ Thông tin – Trường ĐH Công nghệ thông tin – ĐH Quốc gia TP.HCM.
Lĩnh vực nghiên cứu: Khai phá dữ liệu; Tính toán song song; Rút gọn thuộc tính.
Email: tramntt@uit.edu.vn