MỞ ĐẦU

Thực trạng

Việc áp dụng lý thuyết đồ thị vào các bài toán thực tiễn đã được tiến hành từ lâu. Tuy nhiên, khi lượng dữ liệu ngày càng lớn, chẳng hạn dữ liệu từ các mạng xã hội như Facebook, thì việc mô hình hoá dữ liệu bằng lý thuyết đồ thị lại được quan tâm và đã minh chứng được hiệu năng nổi bật khi áp dụng vào thực tế. Khi mô hình hoá dữ liệu bằng đồ thị, thông thường các thực thể (chẳng hạn các thành viên mạng xã hội) sẽ được biểu diễn thông qua các đỉnh còn các quan hệ giữa các thực thể (chẳng hạn như quan hệ bạn bè giữa các thành viên) được quy về các cạnh liên kết các đỉnh trong đồ thị.

Đối với các bài toán mô hình hoá bằng đồ thị có quy mô lớn về cả số đỉnh và số cạnh, một trong những thách thức lớn được đặt ra là cần phải có những phương pháp (i) tổ chức dữ liệu đồ thị hiệu quả và (ii) tối ưu hoá các phép toán trên đồ thị để nâng cao hiệu năng thi hành.

Bài toán nghiên cứu

Với thực trạng đã đặt ra, trong luận án này chúng tôi quan tâm đến bài toán nghiên cứu đề xuất phương pháp tổ chức dữ liệu đồ thị hiệu quả kết hợp cùng với những giải pháp song song hoá các phép toán (cả có tương tranh lẫn không tương tranh) trên đồ thị quy mô lớn cả về số cạnh lẫn số đỉnh để có thể tiến hành xử lý các truy vấn và phân tích trên đồ thị một cách hiệu quả nhất. Tính hiệu quả được quan tâm trong luận án này sẽ được thể hiện trên cả không gian bộ nhớ lưu trữ và tốc độ thi hành các phép toán tương tranh.

Mục tiêu, phạm vi nghiên cứu và các nội dung chính

Mục tiêu nghiên cứu

Từ bài toán đặt ra, mục tiêu chính của luận án là khảo sát, đánh giá các giải pháp hiện đại về xử lý các phép toán tương tranh trên đồ thị quy mô lớn; từ đó đề xuất phương pháp biểu diễn dữ liệu đồ thị phù hợp và nâng cao hiệu năng thi hành các phép toán đồng thời trên đồ thị có quy mô lớn.

Phạm vi và phương pháp nghiên cứu

Về phạm vi, do khuôn khổ thời gian, trong luận án này chúng tôi chỉ chú trọng đến bài toán nghiên cứu trên đồ thị không trọng số. Với kiểu đồ thị đó, các phép toán tương tranh được quan tâm sẽ được quy về thành các phép toán thêm cạnh, xoá cạnh và truy vấn khoảng cách ngắn nhất giữa hai đỉnh. Đối với các phép toán hỗ trợ phân tích đồ thị mạng xã hội, một số độ đo trung tâm sẽ được quan tâm, đề xuất giải pháp nâng cao hiệu năng tính toán các độ đo này trong luận án.

Về phương pháp nghiên cứu, trong luận án này chúng tôi sẽ kết hợp cả phương pháp nghiên cứu lý thuyết lẫn nghiên cứu thực nghiệm. Các đề xuất trong luận án cũng được chú trọng phân tích, đánh giá tường minh về tính đúng đắn, độ phức tạp về mặt lý thuyết. Việc thực nghiệm cũng được tiến hành trên những bộ dữ liệu thường xuyên được cộng đồng nghiên cứu sử dụng và trên cùng nền tảng tính toán để có thể so sánh với những giải pháp khác đã được công bố.

Chương 1: LÝ THUYẾT NỀN TẢNG

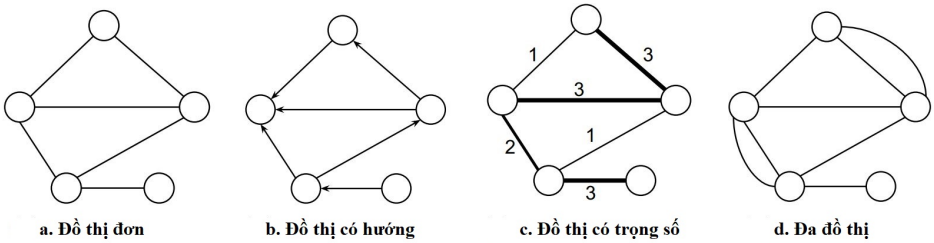
1.1 Lý thuyết đồ thị

1.1.1 Khái niệm

Đồ thị là một cấu trúc dữ liệu linh hoạt, được thể hiện dưới dạng một tập các đỉnh (vertices) và các cạnh (edges), hay được gọi với thuật ngữ khác là tập các nút (nodes) và các quan hệ kết nối giữa chúng với nhau (relationships).

1.1.2 Kiểu đồ thị

Một số các kiểu đồ thị được thể hiện trong Hình 1.1 dưới đây:



Hình 1.1: Một số kiểu đồ thị cơ bản

1.2 Biểu diễn đồ thị

Đồ thị có thể được biểu diễn bởi những phương pháp sau đây:

- Danh sách các cạnh

- Ma trận liền kề
- Danh sách liền kề
- Ma trận liên thuộc
- Ma trận hàng thưa nén

1.3 Các phép toán chính trên đồ thị

Trong phần này, chúng tôi sẽ trình bày những kết quả tóm lược các phép toán chính trên đồ thị. Các phép toán này được chia làm hai loại chính: (i) duyệt đồ thị, và (ii) phân tích đồ thị.

1.3.1 Duyệt đồ thị

Định nghĩa 1.1. *Duyệt đồ thị (hay đôi khi còn gọi tìm kiếm trên đồ thị) là quá trình thăm (kiểm tra hoặc/và cập nhật) mỗi đỉnh trong đồ thị. Quá trình duyệt đồ thị thường được phân loại dựa theo thứ tự các đỉnh được khám phá.*

Với giả thiết đồ thị G được biểu diễn theo phương pháp sử dụng danh sách liền kề, hai phương pháp duyệt theo chiều rộng trước và duyệt theo chiều sâu trước được tiến hành theo giải thuật dưới đây:

1.3.1.1 Duyệt theo chiều rộng trước - BFS

Định nghĩa 1.2. *Duyệt theo chiều rộng trước (Breadth-First Search - BFS) là giải thuật để duyệt cấu trúc dữ liệu đồ thị. Giải thuật này cho phép từ một đỉnh trong đồ thị, gọi là đỉnh gốc, khám phá tất cả các đỉnh liền kề có cùng độ sâu so với nút gốc trước khi chuyển đến các đỉnh ở độ sâu tiếp theo.*

1.3.1.2 Duyệt theo chiều sâu trước - DFS

Định nghĩa 1.3. *Duyệt theo chiều sâu trước (Depth-First Search - DFS) là giải thuật để duyệt cấu trúc dữ liệu đồ thị, cho phép từ một đỉnh trong đồ thị, gọi là đỉnh gốc, khám phá càng xa càng tốt dọc theo từng đỉnh liền kề trước khi quay trở lại. Hay khác với BFS khám phá các đỉnh anh em cùng mức trước, DFS cho phép khám phá các nút con trước khi quay lại khám phá các đỉnh anh em.*

1.3.2 Phân tích đồ thị

Lý thuyết đồ thị được ứng dụng trong nhiều lĩnh vực khác nhau hiện nay. Tuỳ thuộc vào mỗi lĩnh vực ứng dụng đó mà việc phân tích đồ thị được tiến hành với những phương pháp và kỹ thuật khác nhau. Một số miền lĩnh vực có thể ứng dụng lý thuyết đồ thị để phân tích bao gồm: mạng xã hội (social network); mạng sinh học (biological network); mạng liên kết (link network); ...

1.3.2.1 Tính khoảng cách

Trong đồ thị G, việc xác định đường đi ngắn nhất từ một đỉnh u đến đỉnh v sẽ cho phép tính được khoảng cách giữa chúng. Tuỳ thuộc vào mỗi kiểu đồ thị mà cách tính khoảng cách giữa u và v được xây dựng khác nhau.

1.3.2.2 Đường đi ngắn nhất

Trong lý thuyết đồ thị, đường đi ngắn nhất từ đỉnh u đến v được định nghĩa là đường đi có khoảng cách $dis(u, v)$ ngắn nhất. Với đồ thị không có trọng số, đường đi ngắn nhất chính là đường đi có số cạnh nhỏ nhất từ u đến v.

1.3.2.3 Độ trung tâm

Đối với các phép toán phân tích đồ thị, luận án quan tâm đến hai độ trung tâm sau đây:

Định nghĩa 1.4. *Độ trung tâm gần của một đỉnh v được định nghĩa dựa theo công thức do Bavelas đề xuất như sau:*

$$CC(v) = \frac{1}{\sum_{u \in V} dst(u, v)} \tag{1.1}$$

với $dst(u, v)$ là khoảng cách ngắn nhất từ nốt u đến v.

Định nghĩa 1.5. *Độ trung tâm trung gian của một đỉnh v được tính bởi công thức sau:*

$$BC(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}, \tag{1.2}$$

với σ_{st} là khoảng cách ngắn nhất từ đỉnh s đến t và $\sigma_{st}(v)$ là số đường đi ngắn nhất đó đi qua đỉnh v.

1.4 Xử lý song song

Định nghĩa 1.6. *Tính toán song song (parallel computing) là kiểu tính toán trong đó nhiều phép tính (hoặc việc thi hành nhiều tiến trình) được tiến hành một cách đồng thời dựa trên nguyên tắc những bài toán lớn đều có thể chia thành nhiều bài toán nhỏ hơn có thể tiến hành đồng thời.*

Việc đánh giá hiệu năng của các mô hình tính toán song song hiện nay thường sử dụng luật Amdahl (Amdahl's Law),cho rằng hệ số tăng tốc của một chương trình song song được xác lập dựa vào tỷ lệ mã (code) P của chương trình đó được song song hoá:

$$SpeedUp = \frac{1}{1 - P} \tag{1.3}$$

Khi xét thêm số lượng CPU tham gia vào tính toán song song, hệ số này được xác định bằng biểu thức sau:

$$SpeedUp = \frac{1}{S + \frac{P}{N}} \tag{1.4}$$

trong đó, P là tỷ lệ mã song song, N là số lượng CPU và S là tỷ lệ mã tuần tự.

Chương 2: TỐI ƯU HOÁ TRUY VẤN TƯƠNG TRANH TRÊN ĐỒ THỊ ĐỘNG

2.1 Giới thiệu

Chương này trình bày giải pháp cải thiện hiệu suất xử lý các truy vấn đồng thời trên đồ thị có hướng, không trọng số với quy mô lớn và có tính thay đổi nhanh. Chiến lược của chúng tôi dựa trên các ý tưởng: (i) cấu trúc dữ liệu thích hợp, (ii) giảm thiểu không gian tìm kiếm, và (iii) thi hành song song hiệu quả.

2.2 Đặc tả bài toán

Xét đồ thị $G = (V, E)$ là đồ thị có tập đỉnh V và E lớn, từ vài triệu đỉnh đến hàng tỷ đỉnh, liên tục có những thay đổi (thêm/bớt) về số lượng thực thể V (tức số lượng đỉnh) và thay đổi về số lượng quan hệ (tức số lượng cạnh). Các quan hệ E giữa các thực thể trong G được mô hình hoá không xét đến trọng số và có hướng.

2.2.1 Mô hình dữ liệu và truy vấn

Biểu diễn các cạnh đồ thị có quy mô lớn dựa theo danh sách liền kề là cách tiếp cận phù hợp nhất.

Các phép toán trên đồ thị hoàn toàn có thể được mô hình hoá dựa theo ba phép toán chính: Thêm cạnh [$'A'$ u v], Xoá cạnh [$'D'$ u v], Truy vấn [$'Q'$ u v]

2.2.2 Bài toán đặt ra

Định nghĩa 2.1. *Xét đồ thị $G = (V, E)$ tại thời điểm t có n phép toán tương tranh (concurrent operations) $S = \{op_1, op_2..., op_n\}$. Mỗi phép toán op_i của S tương ứng với một trong ba phép toán đã đặc tả ở trên và được biểu diễn bằng bộ ba $op_i = (a_i, u_i, v_i)$; trong đó $a \in ['A','D','Q']$ là kiểu phép toán thao tác với cặp đỉnh (u, v). Yêu cầu đặt ra là phải tiến hành xử lý các phép toán trong tập S một cách nhanh nhất có thể mà vẫn đảm bảo được tính nhất quán của G đối với thứ tự thi hành các phép toán trong S.*

Việc thực thi lịch S được chúng tôi tiến hành theo hướng xây dựng giải pháp thi hành trên các hệ thống tính toán đa CPU, nhiều lõi và có không gian bộ nhớ chính lớn.

2.3 Giải pháp 1: akgroup

Việc thi hành các phép toán tương tranh trong S được tiến hành theo giải thuật 2.1 sau:

Thuật toán 2.1: Giải pháp 1: Các bước thực hiện lịch thi hành S

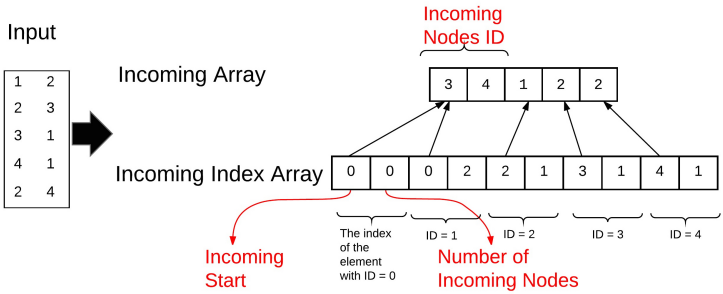
Input: $S[a, u, v]$ mảng đại diện lịch thi hành chứa các phép toán tương tranh; Q là danh sách chứa các truy vấn tính khoảng cách ngắn nhất trong G
Output: G được cập nhật và danh sách $Dist[.]$ chứa khoảng cách ngắn nhất theo S

```
1  query_num ← 0 ;
2  foreach (a, u, v) ∈ S do
3      if a =' A' then
4          add_edge(u, v) ;
5      end
6      else if a =' D' then
7          del_edge(u, v) ;
8      end
9      else if a =' Q' then
10         Q[query_num] ← (u, v) ;
11         query_num+ = 1;
12     end
13 end
14 if query_num > 0 then
15     exec_queries(Q, Dist) ;
16 end
17 return Dist[.] ;
```

2.3.1 Cấu trúc dữ liệu đồ thị phù hợp

Dữ liệu đồ thị G sẽ được biểu diễn như sau:

- Mỗi nút v sẽ được định danh bằng một tự nhiên nằm trong khoảng $[0..(|V|-1)]$
- Các đỉnh đến/đi của một nút sẽ được lưu trữ trong một vùng liền kề của một mảng tương ứng gọi là incoming_edges/outgoing_edges.
- Việc định vị và số lượng các đỉnh liền kề của v trong mảng incoming_edges/outgoing_ đó sẽ được tiến hành dựa theo mảng chỉ mục incoming_index/outgoing_index. Để tăng tỷ lệ "cache hit", số đỉnh liền kề num và vị trí bắt đầu pos trong $incoming\_edges/outgoing\_index$ sẽ được lưu theo cặp kề nhau (pos, num).



Hình 2.1: Minh hoạ cấu trúc dữ liệu đồ thị

2.3.2 Tối ưu các truy vấn

2.3.2.1 Giải thuật tính khoảng cách ngắn nhất

Việc tính khoảng cách ngắn nhất từ u đến v trong đồ thị có hướng không trọng số G sẽ dựa trên giải thuật bBFS, duyệt BFS theo cả hai chiều: từ u duyệt dần theo chiều rộng trước và từ v lần ngược lên cũng theo chiều rộng. Quá trình duyệt chỉ kết thúc khi tồn tại đỉnh nằm trong cả hàng đợi duyệt của chiều đi và chiều đến.

- Sử dụng hai mảng bitmaps đi và đến để mỗi bit xác lập trạng thái đỉnh tương ứng vị trí bit đó đã được duyệt hay chưa.
- Sử dụng chiến lược lựa chọn chiều duyệt BFS có ghi nhận thêm số lượng các cháu trong hàng đợi: theo chiều đi hay đến dựa trên tổng số đỉnh con và đỉnh cháu ở mỗi hàng đợi.

2.3.2.2 Xử lý song song truy vấn

Để có thể tiến hành song song các truy vấn tìm khoảng cách ngắn nhất, giải pháp của chúng tôi đề xuất dựa vào các ý tưởng sau:

- Song song hoá quá trình thực hiện các truy vấn tính khoảng cách trên các luồng khác nhau chứ không song song quá trình tính khoảng cách ngắn nhất.
- Sử dụng hai hàng đợi toàn cục để chứa các đỉnh đến/đi khi duyệt đồ thị (gọi là $incoming\_queue$ và $outgoing\_queue$); Việc xử lý truy vấn tìm khoảng cách ngắn nhất được tiến hành trong một luồng riêng biệt và sử dụng một mảng $incoming\_queue$ và $outgoing\_queue$ riêng cho luồng đó.
- Hai mảng toàn cục để lưu vết các đỉnh đã duyệt (in_maps và out_maps). Hai mảng lưu vết này sẽ được cấp phát trước và mỗi mảng có kích thước là

$\frac{|V|}{32} * threadNum$ phần tử kiểu Integer (4-bytes) với $threadNum$ là số luồng song song.

- Giải pháp song song sẽ được cài đặt dựa trên bộ thư viện Cilkplus do Intel xây dựng .

2.4 Giải pháp 2: akGroupPlus

Được xây dựng dựa trên giải pháp 1 và thực hiện cải tiến trên cả hai phương diện: thay đổi cách thức tổ chức dữ liệu cho phù hợp hơn và áp dụng những kỹ thuật song song hoá các truy vấn một cách hiệu quả hơn đối với cả các phép toán cập nhật lẫn truy vấn tính khoảng cách ngắn nhất.

2.4.1 Tổ chức dữ liệu đồ thị kèm trạng thái

Dữ liệu đồ thị được tổ chức theo cả chiều đi lẫn chiều đến như sau:

- (i) Mỗi đỉnh được định danh bằng một ~~giá trị~~ số 4-bytes;
- (ii) Các đỉnh cạnh đến/đi của đỉnh u sẽ được lưu trong vector có sắp xếp thứ tự tăng dần $incomingEdges[u]/outgoingEdges[u]$. Mỗi phần tử v kích thước 4 bytes của vector đó sẽ được mã hoá: 30 bits trái nhất đầu tiên để định danh đỉnh đến/đi; 2 bits còn lại để thể hiện trạng thái của cạnh (u, v) (ALIVE, DEAD, UNKNOWN).

Việc tổ chức các danh sách đỉnh liền kề theo cả chiều đi lẫn chiều đến sẽ cho phép chúng ta duyệt BFS theo cả hai chiều.

2.4.2 Xử lý các phép toán tương tranh

Việc thi hành các phép toán tương tranh trong S sẽ được xử lý theo các bước sau:

- Lưu lại có thứ tự các phép toán cập nhật vào mảng Updates; các phép toán truy vấn đường đi ngắn nhất vào mảng Queries;
- Tiến hành xử lý tuần tự các phép toán cập nhật và đánh dấu các cạnh được cập nhật về trạng thái UNKNOWN;
- Tiến hành xử lý song song các truy vấn đường đi ngắn nhất;
- Cập nhật chính thức trạng thái các cạnh liên quan đến các phép toán cập nhật, tức chuyển trạng thái các cạnh đó từ UNKNOWN về DEAD (nếu bị xoá) hay ALIVE (nếu được thêm vào);

Các bước xử lý trên có thể được minh hoạ cụ thể hơn qua giải thuật 2.2 dưới đây:

Thuật toán 2.2: akGroupPlus: Thi hành các phép toán tương tranh trong S

```
1  Function akGroupPlus(S)
       Input: Đồ thị G và danh sách S có n phép toán (a, u, v) với 'a' là phép toán
       Output: G ghi nhật tất cả cập nhật và danh sách khoảng cách ngắn nhất của tất cả truy
               vấn 'Q'
2      for t = 0; t < n; t++ do
3          (a,u,v) = Op[t]; if a =' Q' then
4              Queries.push_back(t,u,v) ;            /* đẩy bộ (t, u, v) vào vector Queries */
5          end
6          else
7              Updates.push_back(t,a,u,v) ;      /* đẩy bộ (t, a, u, v) vào vector Updates  */
8          end
9      end
10     Call UpdateSerial(Updates) ;          /* Cập nhật các cạnh theo giải thuật ??  */
11     Call ParallelQuery(Queries) ;      /* Thi hành song song các truy vấn 'Q' theo giải
        thuật 2.3  */
12     Call CommitSerial(Updates) ;      /* Ghi nhận các cạnh thêm/xoá trong G theo giải
        thuật ??  */
```

2.4.3 Tối ưu hoá các phép toán cập nhật

Tất cả các phép toán cập nhật cạnh trong vector *Updates* sẽ được thi hành bằng cách cập nhật trước tiên trạng thái UNKNOWN đối với các đỉnh liên quan đến các cạnh thêm/xoá trong *incomingEdges*/*outgoingEdges*. Việc ghi nhận chính thức các phép toán cập nhật sẽ được tiến hành sau khi đã hoàn thành quá trình xử lý các truy vấn khoảng cách ngắn nhất trong *Queries*. Trong quá trình xử lý các truy vấn đó, nhãn thời gian sẽ được sử dụng để xác định khi nào thì ghi nhận các cạnh có trạng thái UNKNOWN.

2.4.3.1 Xử lý song song truy vấn

Với tập các truy vấn đã được lưu trong vector *Queries*, bộ thư viện CilkPlus được sử dụng để cài đặt tính toán song song.

Từ đó, việc xử lý song song các truy vấn khoảng cách trong *Queries* được tiến hành như trong giải thuật 2.3 dưới đây:

Thuật toán 2.3: Thi hành song song các truy vấn khoảng cách trên đồ thị G

```
1  Function ParallelQuery(Queries)
       Input: Queries chứa các truy vấn khoảng cách (u, v) tại thời điểm t; đồ thị G
       Output: Danh sách Dist[.] chứa các khoảng cách tương ứng
       /* Thi hành song song vòng For theo phương pháp CilkPlus                    */
2      for i = 0; i < Queries.size(); i++ do
3          (u, v, t) ← Queries[i] ;
4          Dist[i] = ComputeShortest2(u, v, t) ;
5      end
6      return Dist[.];
```

2.5 Giải pháp 3: bigGraph

Giải pháp bigGraph được xây dựng hoàn toàn dựa trên giải pháp 2, nhưng bổ sung thêm kỹ thuật song song các phép toán cập nhật. Khi đó, lịch thi hành các phép toán tương tranh S sẽ được xử lý theo các bước sau:

- Lưu lại có thứ tự các phép toán cập nhật vào mảng Updates; các phép toán truy vấn đường đi ngắn nhất vào mảng Queries;
- Tiến hành xử lý *song song* các phép toán cập nhật và đánh dấu các cạnh được cập nhật về trạng thái UNKNOWN với ý tưởng: mở $numThreads$ luồng đồng thời; việc cập nhật danh sách đỉnh liền kề của v chỉ được phép tiến hành trên luồng thứ v mod $numThreads$.
- Tiến hành xử lý song song các truy vấn đường đi ngắn nhất;
- Cập nhật chính thức trạng thái các cạnh liên quan đến các phép toán cập nhật, tức chuyển trạng thái các cạnh đó từ UNKNOWN về DEAD (nếu bị xoá) hay ALIVE (nếu được thêm vào) theo phương pháp xử lý *song song* với cùng ý tưởng nêu trên;

Các bước xử lý trên có thể được minh hoạ cụ thể hơn qua giải thuật 2.4 dưới đây:

Thuật toán 2.4: bigGraph: Thi hành các phép toán tương tranh trong S

```
1  Function bigGraph(S)
       Input: Đồ thị G và danh sách S có n phép toán (a, u, v) với 'a' là phép toán
       Output: G ghi nhật tất cả cập nhật và danh sách khoảng cách ngắn nhất của tất cả truy
               vấn 'Q'
2      for t = 0; t < n; t++ do
3          (a,u,v) = Op[t]; if a =' Q' then
4              Queries.push_back(t,u,v) ;            /* đẩy bộ (t, u, v) vào vector Queries */
5          end
6          else
7              Updates.push_back(t,a,u,v) ;      /* đẩy bộ (t, a, u, v) vào vector Updates  */
8          end
9      end
10     Call UpdateParallel(Updates) Call ParallelQuery(Queries) Call
        CommitParallel(Updates)
```

2.6 Thực nghiệm và đánh giá

2.6.1 Môi trường thử nghiệm, đánh giá

Ba giải pháp đã trình bày trong chương 2 đã được chúng tôi tiến hành thử nghiệm đánh giá trên nhiều môi trường tính toán khác nhau, cụ thể như sau:

- Giải pháp 1 được thử nghiệm trên môi trường máy tính cá nhân với cấu hình Intel® Core™ i7-3720QM (6MB Cache, up to 3.60 GHz, 4 cores-8

threads), bộ nhớ 8GB, CentOS 7, gcc 5.1.1 và hệ thống tính toán tại cuộc thi ACM SigMod Programming Contest 2016: 2 x Intel(R) Xeon(R) CPU E5-2697 v4 @ 2.30GHz (45MB Cache, 18-cores per CPU), bộ nhớ chính 128GB, CentOS Linux release 7.2.1511, gcc 6.3.0.

- Với giải pháp 2 và 3, thực nghiệm trên hệ thống tính toán hiệu năng cao: 2 x Intel(R) Xeon(R) CPU E5-2697 v4 @ 2.30GHz (45MB Cache, 18-cores per CPU), bộ nhớ chính 128GB, CentOS Linux release 7.2.1511, gcc 6.3.0.

2.6.2 Dữ liệu thực nghiệm

Các giải pháp xử lý phép toán tương tranh trên đồ thị đều được tiến hành với những bộ dữ liệu đã được công bố từ các tổ chức có uy tín khoa học trên thế giới:

- Dữ liệu từ cuộc thi SigMod Programming Contest 2016

Bảng 2.1: Thống kê các đồ thị sử dụng trong SigMod 2016

	Đồ thị nhỏ	**Đồ thị trung bình**	**Đồ thị lớn**	**Đồ thị rất lớn**
Số đỉnh	1.574.074	2.000.000	1.971.281	6.009.555
Số cạnh	3.232.855	5.000.000	5.533.214	16.518.948

- Dữ liệu SNAP: Chúng tôi chọn hai bộ dữ liệu chính là **Pokec** và **LiveJournal** để tiến hành các thực nghiệm đánh giá.

Bảng 2.2: Thống kê các bộ dữ liệu đồ thị sử dụng trong thực nghiệm

Tham số	**SigMod**	**Pokec**	**LiveJournal**
Số cạnh	1.574.074	68.993.773	30.622.564
Số đỉnh	3.232.855	4.847.571	1.632.803
Đường kính (khoảng cách ngắn nhất lớn nhất)	9	16	11

2.6.3 Sinh các tập lịch thi hành thử nghiệm

Lịch thi hành tương tranh S với các phép toán được lưu trong tệp text:

- Cấu trúc mỗi dòng là một phép toán (a u v)
- Hai tập dữ liệu (workload) sinh ngẫu nhiên với 1.000.000 phép toán cập nhật lẫn truy vấn khoảng cách ngắn nhất: workload 8-1-1 (phân bố các phép toán Q/A/D lần lượt là 0,8/0,1/0,1) và workload 5-4-1 (phân bố các phép toán Q/A/D lần lượt là 0,5/0,4/0,1).

2.6.4 Thực nghiệm và đánh giá kết quả

Dựa trên môi trường thực nghiệm, dữ liệu đã thu thập với các workloads sinh ngẫu nhiên và phương pháp thực nghiệm nêu trên, chúng tôi đã tiến hành cài đặt và thực nghiệm cả ba giải pháp xử lý các truy vấn tương tranh đã trình bày ở trên.

2.6.4.1 Kết quả từ cuộc thi ACM SigMod Contest 2016

Đội tham dự	Small	Medium	X-Large	XX-Large
H_minor_free	0,107	0,220	0,886	2,333
uateam	0,202	0,217	1,085	2,123
akgroup	0,118	0,494	1,284	2,878

2.6.4.2 Đánh giá giải pháp akGroup

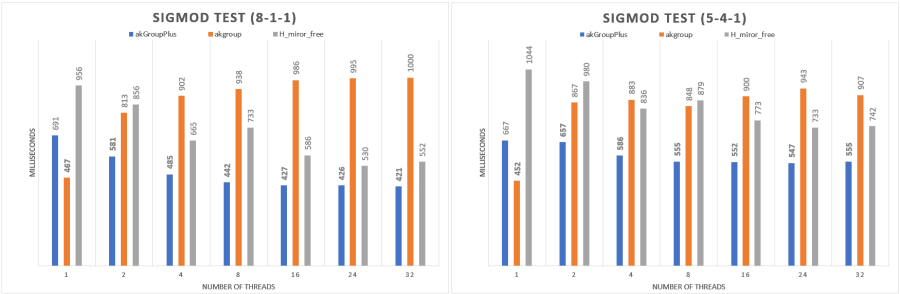
Kết quả đánh giá giải pháp akgroup so với một số giải pháp khác:

Tập dữ liệu	akgroup	BFS+Cilkplus	NetworkX	SNAP C++
SigMod Test	**1,428s**	18,078s	3319,4s	5045s
LiveJournal	**32,045s**	55,630s	Không thể thi hành	>24h
Pokec	**14,286s**	27,825s	>10h	>15h

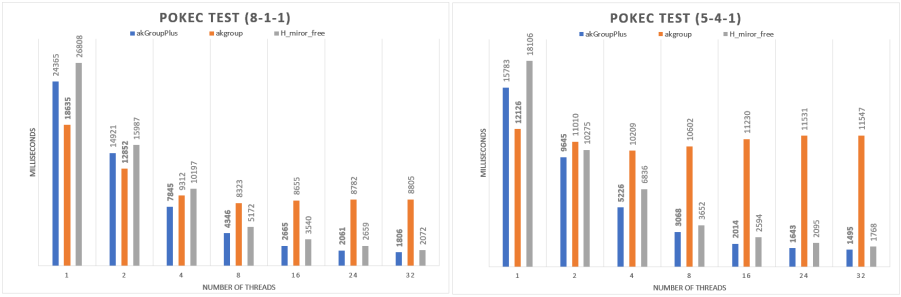
Có thể thấy giải pháp akgroup nhìn chung đã mang lại hiệu quả thi hành tốt hơn so với các giải pháp khác được thử nghiệm.

2.6.4.3 Đánh giá giải pháp akGroupPlus

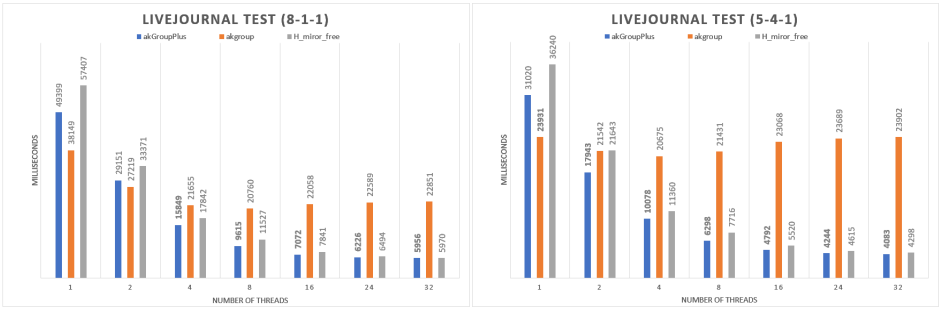
Thực nghiệm so sánh giải pháp này với hai giải pháp khác là: **akgroup** - giải pháp 1 nêu trên; và **H_minor_free** giải pháp đã được giải nhất trong cuộc thi ACM SigMod Contest 2016.



Hình 2.2: Kết quả đánh giá với bộ dữ liệu Sigmod Dataset



Hình 2.3: Kết quả đánh giá với bộ dữ liệu Pokec Dataset

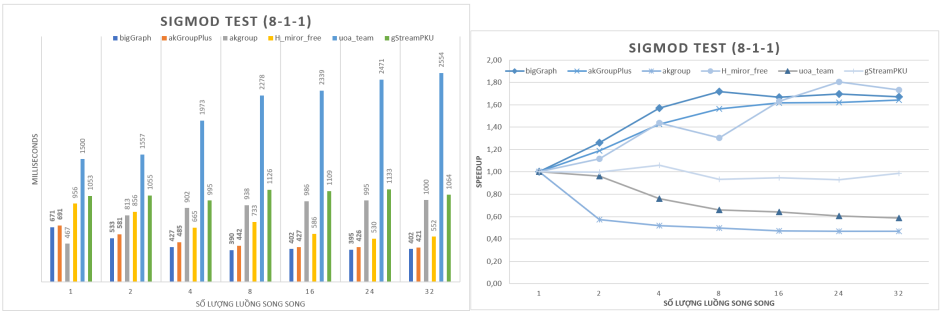


Hình 2.4: Kết quả đánh giá với bộ dữ liệu LiveJournal Dataset

So sánh với bộ dữ liệu LiveJournal và Pokec, *akGroupPlus* luôn mang lại kết quả thời gian thi hành tốt hơn so với hai giải pháp còn lại ngoại trừ khi thi hành tuần tự.

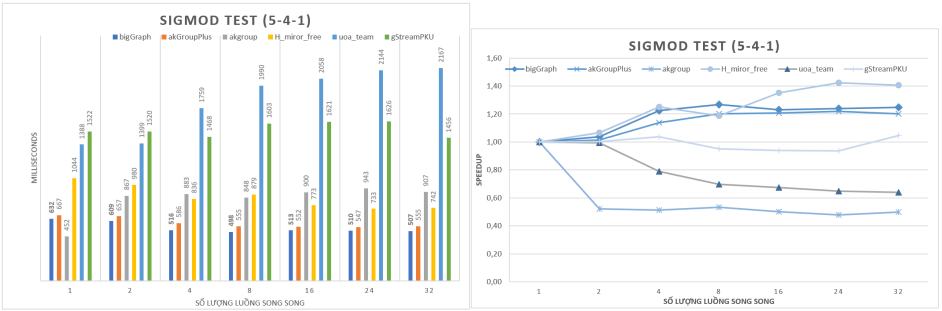
2.6.4.4 Đánh giá giải pháp bigGraph

Đánh giá so sánh **bigGraph** với **akgroup**; **H_minor_free**; và một số công cụ khác như **uoa_team** (đội giành giải nhì); **gStreamPKU**; và **while1** (hai đội đồng giải Ba cùng với akgroup)



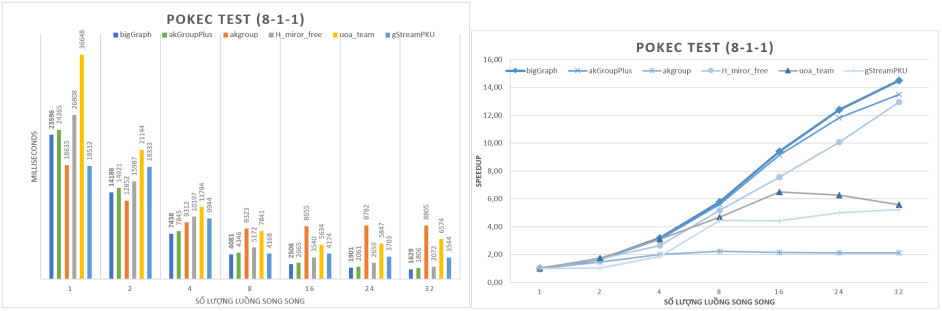
(a) Thời gian thi hành (b) Hệ số tăng tốc

Hình 2.5: Kết quả thực nghiệm với bộ dữ liệu SigMod 8-1-1



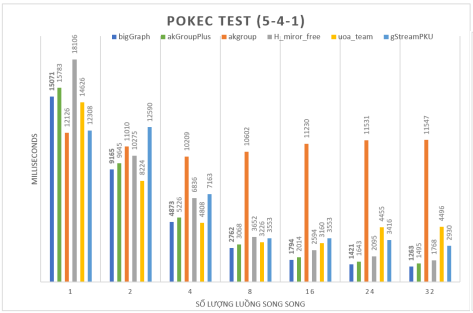
(a) Thời gian thi hành (b) Hệ số tăng tốc

Hình 2.6: Kết quả thực nghiệm với bộ dữ liệu Sigmod 5-4-1

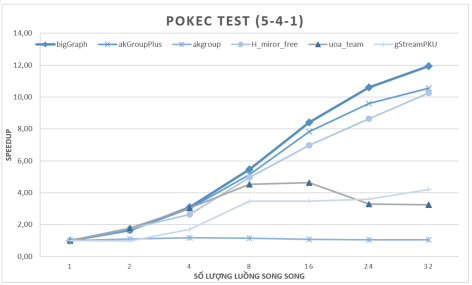


(a) Thời gian thi hành (b) Hệ số tăng tốc

Hình 2.7: Kết quả thực nghiệm với bộ dữ liệu Pokec 8-1-1

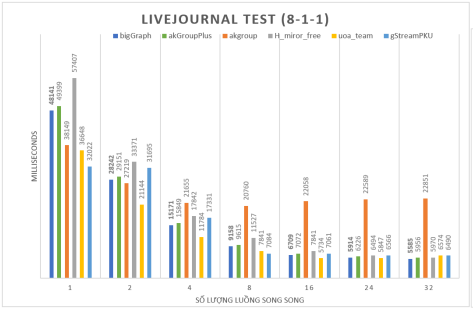


(a) Thời gian thi hành

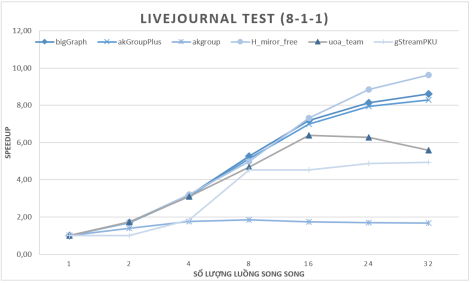


(b) Hệ số tăng tốc

Hình 2.8: Kết quả thực nghiệm với bộ dữ liệu Pokec 5-4-1

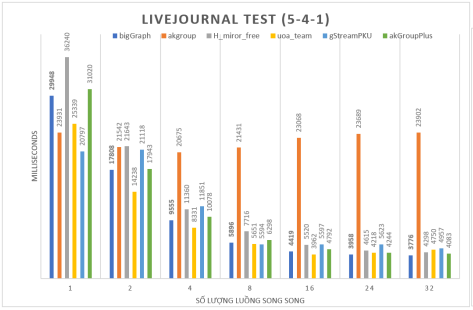


(a) Thời gian thi hành

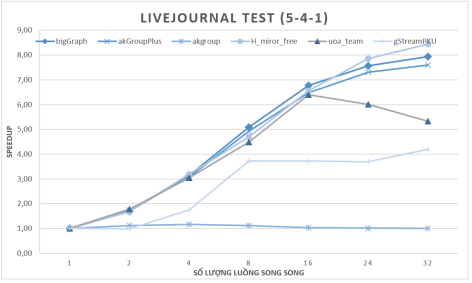


(b) Hệ số tăng tốc

Hình 2.9: Kết quả thực nghiệm với bộ dữ liệu LiveJournal 8-1-1



(a) Thời gian thi hành



(b) Hệ số tăng tốc

Hình 2.10: Kết quả thực nghiệm với bộ dữ liệu LiveJournal 5-4-1

Giải pháp bigGraph nhìn chung có thời gian thi hành hiệu quả tốt nhất so với

các giải pháp còn lại khi tiến hành với số lượng lớn luồng song song; kể cả giải pháp akGroupPlus đã nêu trên.

Chương 3: NÂNG CAO HIỆU NĂNG QUÁ TRÌNH TÍNH ĐỘ TRUNG TÂM TRÊN MẠNG XÃ HỘI

3.1 Tính độ trung tâm gần

Quá trình tính độ trung tâm gần của các đỉnh trong mạng/đồ thị $G = (V, E)$ có thể được minh hoạ bằng giải thuật giả mã được miêu tả trong Giải thuật 3.1.

Thuật toán 3.1: Giải thuật cơ bản tính độ trung tâm gần

```
   Input: G = (V, E); Γ_G(v) biểu diễn danh sách đỉnh liền kề của v
   Output: CC[.] for all v ∈ V
 1 CC[v] ← 0, ∀v ∈ V ;
 2 Sum[v] ← 0, ∀v ∈ V ;
 3 foreach s ∈ V do
 4 |   FC[s] ← 0;
 5 |   Q ← empty queue;
 6 |   Q.push(s);
 7 |   dst[s] ← 0;
 8 |   CC[s] ← 0;
 9 |   dst[v] ← −1, ∀v ∈ V s;
10 |   while Q is not empty do
11 |   |   v ← Q.pop();
12 |   |   forall w ∈ Γ_G(v) do
13 |   |   |   if dis[w] = ∞ then
14 |   |   |   |   Q.push(w);
15 |   |   |   |   dst[w] ← dst[v] + 1;
16 |   |   |   |   Sum[v] ← dst[w];
17 |   |   |   end
18 |   |   end
19 |   end
20 |   CC[s] ← 1/Sum[s] ;
21 end
22 return CC[.];
```

3.1.1 Tính độ trung tâm giữa

Độ trung tâm giữa BC là một trong những độ đo được sử dụng rộng rãi trong phân tích đồ thị nói chung để có thể xác định những nốt quan trọng trong đồ thị. Độ đo này đã được áp dụng để phân tích đồ thị/mạng trong nhiều lĩnh vực khác nhau như trong vận tải, sinh học - y tế, phân tích mạng xã hội để phát hiện cộng đồng, phát hiện nguy cơ khủng bố....

Giả mã của thuật toán Brandes với đồ thị không trọng số được minh hoạ như

trong giải thuật 3.2 dưới đây:

Thuật toán 3.2: Giải thuật cơ bản tính độ trung tâm giữa

Input: $G = (V, E)$, được tổ chức như mảng vector $Edges[][]$
Data: queue $Q \leftarrow empty$, stack S khởi tạo rỗng và có thể chứa được $|V|$ đỉnh ;
$dist[v]$: lưu khoảng cách từ đỉnh nguồn đến v ;
$Pred[v]$: chứa danh sách các đỉnh trên đường đi ngắn nhất từ đỉnh nguồn đến v ;
$\sigma[v]$: số đường đi ngắn nhất từ đỉnh nguồn đến v ;
$\delta[v]$: số đường đi ngắn nhất từ nguồn đi qua v ;
Output: $BC[.]$ với mọi $v \in V$

```
1  foreach s ∈ V do
       /* Pha 1. Duyệt theo kiểu bài toán SSSP                          */
2      foreach v ∈ V do Pred[v] ← empty list ;
3      foreach v ∈ V do dist[v] ← ∞; σ[v] ← 0 ;
4      dist[s] ← 0; σ[s] ← 1; Q.push(s) ;
5      while Q not empty do
6          v ← Q.pop(); S.push(v) ;
7          foreach w ∈ Edges[v] do
               /* w chưa được duyệt                                     */
8              if dist[w] == ∞ then dist[w] ← dist[v] + 1; Q.push(w) ;
               /* (v,w) nằm trên đường đi ngắn nhất                     */
9              if dist[w] == dist[v] + 1 then σ[w] ← σ[w] + σ[v]; Pred[w].push_back(v);
10         end
11     end
       /* Pha 2: Tích luỹ (tính ngược lại đỉnh đã được duyệt)           */
12     foreach v ∈ V do δ[v] ← 0;
13     while S not empty do
14         w ← S.pop() ;
15         for v ∈ Pred[w] do δ[v] ← δ[v] + σ[v]/σ[w] .(1 + δ[w]);
16         if w ≠ s then BC[w] ← BC[w] + δ[w];
17     end
18 end
19 return BC[.] ;
```

3.2 Nâng cao hiệu năng tính độ trung tâm

3.2.1 Cấu trúc dữ liệu phù hợp

Dữ liệu của mạng xã hội quy mô lớn $G = (V, E)$ sẽ được biểu diễn theo phương pháp sử dụng các danh sách đỉnh liền kề như đã trình bày ở chương trước.

3.2.2 Giải thuật song song tính độ trung tâm gần

Thi hành song song các phép tính độ trung tâm gần trên các đỉnh khác nhau chứ không phải song song quá trình duyệt và tính đường đi ngắn nhất từ một đỉnh đến tất cả các đỉnh còn lại (SSSP).

Giải thuật: 3.3

Thuật toán 3.3: Giải thuật song song tính độ trung tâm gần

```
   Input: G = (V, E) represented by Edges[][]
   Output: CC[.] for all v ∈ V
 1 CC[v] ← 0, Sum[v] ← 0, Maps[v] ← 0∀v ∈ V ;
   /* Thi hành song song sử dụng thư viện Cilk Plus                          */
 2 for s = 1 to Edges.size() do
 3     Q ← empty queue; Q.push(s);
 4     SetBit(s, Maps) ;                          /* Đánh dấu s đã được duyệt */
 5     CC[s] ← 0; FC[s] ← 0; dst ← 0 ;
 6     while Q is not empty do
 7         dst ← dst + 1;
           /* Duyệt các nút cùng mức từ s trong hàng đợi Q                   */
 8         while Q is not empty do
 9             v ← Q.pop();
               /* Duyệt các nút liền kề với s                                */
10             forall w ∈ Edges[s] do
                   /* nếu chưa duyệt                                         */
11                 if !TestBit(w) then
12                     Q.push(w, dst) ;        /* lưu lại khoảng cách từ v đến w */
13                     SetBit(w, Maps) ;              /* đánh dấu w đã duyệt */
14                 end
15             end
16         end
17         Q.nextLevel() ;                          /* chuyển đến mức kế tiếp */
18     end
19     if Sum[s] ≠ 0 then CC[s] ← 1/Sum[s] ;
20 end
21 return CC[.];
```

3.2.3 Giải thuật song song tính độ trung tâm giữa

Cách tiếp cận tính toán song song của chúng tôi dựa trên việc thi hành song song các phép tính độ trung tâm giữa trên các đỉnh khác nhau chứ không phải song song quá trình duyệt và tính đường đi ngắn nhất từ một đỉnh đến tất cả các đỉnh còn lại (SSSP).

Giải thuật: 3.4

Thuật toán 3.4: Giải thuật song song tính độ trung tâm giữa

Input: $G = (V, E)$, được tổ chức như mảng vector $Edges[][]$
Data: queue $Q \leftarrow$, stack S khởi tạo rỗng và có thể chứa được $|V|$ đỉnh ;
$dist[v]$: lưu khoảng cách từ đỉnh nguồn đến v ;
$Pred[v]$: chứa danh sách các đỉnh trên đường đi ngắn nhất từ đỉnh nguồn đến v ;
$\sigma[v]$: số đường đi ngắn nhất từ đỉnh nguồn đến v ;
$\delta[v]$: số đường đi ngắn nhất từ nguồn đi qua v ;
$reducerBC[v]$: vector chứa giá trị BC của đỉnh v cho phép cập nhật tương tranh khi thi hành song song với thư viện Cilk Plus ;
Output: $BC[.]$ với mọi $v \in V$

```
/* Thi hành song song sử dụng thư viện Cilk Plus                          */
1  for s = 0 to Edges.size() do
       /* Pha 1. Duyệt theo kiểu bài toán SSSP                           */
2      foreach v ∈ V do Pred[v] ← empty list ;
3      foreach v ∈ V do dist[v] ← ∞; σ[v] ← 0 ;
4      dist[s] ← 0; σ[s] ← 1; Q.push(s) ;
5      while Q not empty do
6          v ← Q.pop(); S.push(v) ;
7          foreach w ∈ Edges[v] do
               /* w chưa được duyệt                                      */
8              if dist[w] == ∞ then dist[w] ← dist[v] + 1; Q.push(w) ;
               /* (v,w) nằm trên đường đi ngắn nhất                      */
9              if dist[w] == dist[v] + 1 then σ[w] ← σ[w] + σ[v]; Pred[w].push_back(v);
10         end
11     end
       /* Pha 2: Tích luỹ (tính ngược lại đỉnh đã được duyệt)            */
12     foreach v ∈ V do δ[v] ← 0;
13     while S not empty do
14         w ← S.pop() ;
15         for v ∈ Pred[w] do δ[v] ← δ[v] + σ[v]/σ[w].(1 + δ[w]);
           /* Đảm bảo thi hành tương tranh chính xác                     */
16         if w ≠ s then reducerBC[w] ← reducerBC[w] + δ[w];
17     end
18 end
19 reducerBC.move_out(BC) ;                    /* Trả kết quả về cho vector BC */
20 return BC[.] ;
```

3.3 Thực nghiệm và đánh giá

3.3.1 Môi trường thử nghiệm, đánh giá

Giải thuật song song tính độ trung tâm gần CC được cài đặt bằng ngôn ngữ C++ trên hệ thống tính toán hiệu năng cao, với cấu hình 2 x Intel(R) Xeon(R) CPU E5-2697 v4 @ 2.30GHz (45MB Cache, 18-cores per CPU), bộ nhớ chính 128GB, CentOS Linux release 7.2.1511, gcc 6.3.0. Hệ thống tính toán này được cấu hình cho phép thi hành tối đa 36 luồng song song (do cấu hình tắt chức năng hyperthreading trên 2 CPU).

3.3.2 Dữ liệu thực nghiệm

Thu thập các bộ dữ liệu mạng xã hội được công bố từ hai tổ chức lớn: SNAP (Stanford Large Network Dataset Collection) và Aminer Datasets for Social Network Analysis,bao gồm: gemsec-Facebook Politician, gemsec-Facebook Artist, ego-Facebook,com-DBLP, com-Youtube, Flickr.

Bảng 3.1: Thông tin thống kê về các dữ liệu mạng xã hội thử nghiệm

Bộ dữ liệu	**Số cạnh**	**Số đỉnh**	**Đường kính**
ego-Facebook (DS1)	88.234	4.039	8
gemsec-Facebook Politician (gọi tắt DS2)	41.729	5.908	14
gemsec-Facebook Artist (DS3)	819.306	50.515	11
DBLP (DS4)	1.049.866	425.957	23
Youtube (DS5)	2.987.624	1.157.828	24
Flickr (DS6)	9.114.557	214.626	10

3.3.3 Kết quả thực nghiệm và đánh giá

3.3.3.1 Giải pháp nâng cao hiệu năng tính độ trung tâm gần

Hai bộ công cụ tiêu biểu chúng tôi chọn để so sánh là: TeexGraph và NetworKit. Kết quả thực nghiệm của chúng tôi được tổng hợp dựa trên việc tính thời gian thi hành trung bình của cả 10 lần chạy thử nghiệm đối với mỗi giải pháp và được thể hiện ở bảng sau:

Bảng 3.2: Thời gian (giây) và hệ số tăng tốc của bigGraph khi tính độ trung tâm gần

Số luồng	DS1		DS2		DS3	
	Thời gian	Speedup	Thời gian	Speedup	Thời gian	Speedup
1	1,031	1,00	1,546	1,00	195,276	1,00
2	0,552	1,87	0,819	1,89	97,527	2,00
4	0,306	3,37	0,415	3,72	49,136	3,97
8	0,169	6,11	0,223	6,95	26,418	7,39
16	0,095	10,82	0,129	12,03	14,491	13,48
32	0,054	19,07	0,069	22,48	7,481	26,10
36	0,052	19,89	0,060	25,96	6,648	29,37

Kết quả minh hoạ ở bảng trên cho thấy càng nhiều luồng song song chúng ta có thể tiến hành, thời gian thi hành tính độ trung tâm gần càng ngắn. Từ đó, chúng tôi quyết định chọn số luồng song song là 36 luồng (số luồng tối đa trên hệ thống tính toán của chúng tôi) khi thi hành so sánh đánh giá hiệu năng giữa bigGraph và NetworKit, TeexGraph.

Bảng sau minh hoạ thời gian trung bình của 10 lần thi hành quá trình tính độ trung tâm gần của cả ba giải pháp nêu trên:

Bảng 3.3: Thời gian tính độ trung tâm gần (giây)

Bộ dữ liệu	**Networkit**	**Teexgraph**	**bigGraph**
ego-Facebook	0,468	0,052	**0,032**
gemsec-Facebook Politician	1,192	0,071	**0,056**
gemsec-Facebook Artist	182,890	9,808	**6,405**
DBLP	3363,286	326,659	**153,753**
Flickr		540,944	**309,058**
Youtube	147.924,400	4.418,191	**2.168,677**

Bảng 3.4 cho thấy hệ số tăng tốc của bigGraph đối với cả 6 bộ dữ liệu đều nhanh hơn từ 1,27 đến 2,12 và từ 14,78 đến 68,21 so với 2 bộ công cụ còn lại, lần lượt là TeexGraph và NetworKit.

Bảng 3.4: Hệ số tăng tố của bigGraph so với Teexgraph và Networkit khi tính độ trung tâm gần

Bộ dữ liệu	**Teexgraph/bigGraph**	**Networkit/bigGraph**
ego-Facebook	1,66	14,78
gemsec-Facebook Politician	1,27	21,23
gemsec-Facebook Artist	1,53	28,56
DBLP	2,12	21,87
Flickr	1,75	
Youtube	2,04	68,21

3.3.3.2 Giải pháp nâng cao hiệu năng tính độ trung tâm giữa

Tiến hành cài đặt giải pháp song song hoá quá trình tính độ trung tâm giữa bằng ngôn ngữ C++ và sử dụng thư viện lập trình song song theo mô hình luồng CilkPlus, so sánh với cả hai bộ công cụ TeexGraph và NetworKit. Cả hai bộ công cụ này và bigGraph đều được cài đặt trên hạ tầng phần cứng đã được đề cập ở phần trên.

Kết quả thực nghiệm của chúng tôi được tổng hợp dựa trên việc tính thời gian thi hành trung bình của cả 10 lần chạy thử nghiệm đối với mỗi giải pháp và được thể hiện ở bảng sau:

Bảng 3.5: Thời gian (giây) và hệ số tăng tốc của bigGraph khi tính độ trung tâm giữa

Số luồng	DS1		DS2		DS3	
	Thời gian	Speedup	Thời gian	Speedup	Thời gian	Speedup
1	3,03	1,00	8,20	1,00	1129,47	1,00
2	2,52	1,20	7,44	1,10	832,76	1,36
4	1,51	2,01	4,52	1,82	556,46	2,03
8	0,92	3,29	2,61	3,15	330,64	3,42
16	0,54	5,62	1,60	5,14	196,46	5,75
32	0,28	10,79	0,89	9,19	118,92	9,50
36	0,23	13,26	0,74	11,01	99,85	11,31

Bảng sau minh hoạ thời gian trung bình của 10 lần thi hành quá trình tính độ trung tâm giữa của cả ba giải pháp nêu trên:

Bảng 3.6: Thời gian tính độ đo BC (giây)

Bộ dữ liệu	**bigGraph**	**Teexgraph**	**Networkit**
ego-Facebook	**0,23**	0,31	0,56
gemsec-Facebook Politician	**0,84**	0,84	1,70
gemsec-Facebook Artist	**99,85**	110,58	234,12
DBLP	**2345,62**	2694,78	4823,47
Flickr	**3.506,34**	4.447,93	7.694,61
Youtube	**56.071,60**	68.744,80	90.522,30

Hệ số tăng tốc của bigGraph đối với cả 6 bộ dữ liệu đều nhanh hơn từ 1,11 đến 1,35 và từ 2,06 đến 2,44 so với hai công cụ còn lại, lần lượt là TeexGraph và NetworKit.

Bảng 3.7: Hệ số tăng tốc của bigGraph so với Teexgraph và Networkit khi tính độ đo BC

Bộ dữ liệu	**Teexgraph/bigGraph**	**Networkit/bigGraph**
ego-Facebook	1,35	2,44
gemsec-Facebook Politician	1,13	2,28
gemsec-Facebook Artist	1,11	2,34
DBLP	1,15	2,06
Flickr	1,27	1,61
Youtube	1,23	2,19

KẾT LUẬN CHUNG

Các đóng góp chính

Luận án tập trung nghiên cứu ứng dụng lý thuyết đó để giải quyết bài toán nâng cao hiệu năng xử lý các phép toán đồng thời trên đồ thị. Với định hướng đó, chúng tôi đã tiến hành xác lập mục tiêu và các nội dung nghiên cứu chính của luận án. Qua các kết quả cả về lý thuyết lẫn thực nghiệm đã được trình bày cụ thể trong luận án, có thể khẳng định rằng toàn bộ mục tiêu và các nội dung nghiên cứu đề ra đã được hoàn thành, với các đóng góp chính bao gồm:

1. Mô hình hoá quá trình xử lý các phép toán tương tranh trên đồ thị quy mô lớn dựa vào lịch thi hành các phép toán đồng thời và dựa vào cấu trúc dữ liệu phù hợp; từ đó cho phép nâng cao hiệu năng bộ nhớ đệm cache và giảm thời gian truy xuất đến bộ nhớ chính.

2. Đề xuất ba giải pháp (akgroup, akgroupPlus và bigGraph) để nâng cao hiệu năng thi hành các truy vấn đồng thời trên đồ thị động quy mô lớn với khả năng thi hành song song cả các truy vấn duyệt đồ thị lẫn cập nhật đồ thị.

3. Nâng cao hiệu năng quá trình tính hai độ đo trung tâm: độ trung tâm gần và độ trung tâm giữa trên đồ thị mạng xã hội quy mô lớn, với giải pháp BigGraph được xây dựng dựa trên việc tổ chức dữ liệu đồ thị phù hợp và song song hoá các phép tính SSSP trên mỗi thành viên của mạng.

Toàn bộ các công trình đã công bố của chúng tôi đều được xuất bản trong các kỷ yếu hội thảo, tạp chí có chỉ mục trong SCOPUS và WoS.

Hạn chế của luận án

Do những hạn chế cả về thời gian, về nguồn lực lẫn sự giới hạn về không gian nghiên cứu, luận án còn một số điểm hạn chế sau:

- Các nội dung nghiên cứu của luận án còn chưa thể tiến hành với những đồ thị/mạng xã hội có quy mô siêu lớn, chẳng hạn như Facebook với số lượng đỉnh hơn hai tỷ và hơn nghìn tỷ cạnh.

- Một số độ đo trung tâm như độ trung tâm vector riêng, độ trung tâm xếp hạng trang... còn chưa được nghiên cứu, cài đặt trong phần tính các độ trung tâm.

- Chưa đưa được kết quả nghiên cứu của luận án áp dụng cụ thể vào các bài toán thực tế.

Hướng phát triển tương lai

Toàn bộ những hạn chế chế đã nêu trên đều được chúng tôi xác định sẽ là những nghiên cứu được chú trọng tiến hành trong thời gian tới.

Trong thời gian tới, những đồ thị quy mô siêu lớn cũng sẽ được chúng tôi quan tâm nghiên cứu để có thể làm chủ được những công nghệ xử lý đối với quy mô dữ liệu đồ thị siêu lớn đó. Đối với các độ đo trung tâm cũng như các phép toán phân tích mạng xã hội, chúng tôi cũng sẽ tiến hành cài đặt trong thời gian tới để hình thành nên bộ công cụ hoàn chỉnh phục vụ cho các lớp bài toán duyệt và phân tích đồ thị/mạng xã hội. Ngoài ra, với những tiềm năng của CSDL đồ thị, trong tương lai gần, chúng tôi sẽ hướng đến những nghiên cứu chuyên sâu hơn trong việc tối ưu hoá xử lý các truy vấn trên đồ thị thuộc tính; cho phép xử lý hiệu quả hơn các truy vấn trên đồ thị động, thay đổi theo thời gian (evolving graph database)...