CẢI THIỆN TỐC ĐỘ TÌM KIẾM CỦA MÔ HÌNH ĐỒ THỊ BT-GRAPH DỰA TRÊN NỀN TẢNG CUDA

Lương Hoàng Hướng[1], Nguyễn Hải Thanh[2], Huỳnh Xuân Hiệp[3]

[1] Trung tâm Công nghệ phần mềm, Đại học Cần Thơ

[2] Vụ Khoa học, Công nghệ và Môi trường, Bộ Giáo dục và Đào tạo Việt Nam

[3] Khoa Công nghệ thông tin và Truyền thông, Nhóm nghiên cứu liên ngành DREAM-CTU/IRD, Đại học Cần Thơ

lhhuong@ctu.edu.vn, nhthanh@{moet.gov.vn, moet.edu.vn}, hxhiep@ctu.edu.vn

TÓM TẮT *- BT-Graph (Graph Model based on Ball Tree Structure) là một mô hình đồ thị được xây dựng dựa trên cấu trúc balltree, giúp mô hình hóa hệ thống mạng giám sát các bẫy đèn tự động và hỗ trợ tìm kiếm vị trí địa lý. Khi số lượng vị trí địa lý lớn và không gian địa lý tìm kiếm mở rộng thì cần phải cải thiện tốc độ tìm kiếm của mô hình đồ thị BT-Graph. Trong bài viết này, chúng tôi đề xuất một hướng tiếp cận mới trong việc cải thiện tốc độ tìm kiếm của mô hình đồ thị BT-Graph bằng phương pháp song song hóa thuật toán tìm kiếm dựa trên nền tảng CUDA NVIDIA. Các thực nghiệm được triển khai trên hai thuật toán tìm kiếm k-láng giềng gần nhất và tìm kiếm đường đi ngắn nhất dựa trên mô hình đồ thị BT-Graph và cho thấy sự cải thiện tốt về thời gian tìm kiếm.*

Từ khóa *- CUDA, BT-Graph, vị trí địa lý, mạng giám sát bẫy đèn tự động, song song.*

I. GIỚI THIỆU

BT-Graph (Graph Model based on Ball Tree Structure) [11] là một mô hình đồ thị được xây dựng dựa trên cấu trúc balltree [21] [11]. BT-Graph không chỉ giúp mô hình hóa hệ thống mạng giám sát các bẫy đèn tự động bằng cách đề xuất bán kính hoạt động cho các cảm biến tự động, mà còn hỗ trợ tìm kiếm vị trí địa lý [11]. Tuy nhiên, khi số lượng vị trí địa lý lớn và không gian địa lý tìm kiếm mở rộng thì cần phải cải tiến tốc độ tìm kiếm của mô hình đồ thị BT-Graph.

A) B) C)

Hình 1. A) Tập hợp điểm, B) Cấu trúc Balltree, C) Mô hình đồ thị BT-Graph

Ngày nay, việc sử dụng Graphic Processing Units (GPUs) [28] đóng vai trò quan trọng trong xử lý các ứng dụng đòi hỏi cần phải xử lý song song. Ngoài ra GPUs cũng hỗ trợ tốt trong việc xử lý đồ thị mà không cần phải giảm độ phức tạp của mô hình đồ thị. Đã có nhiều nghiên cứu về việc cho thấy hiệu suất cao giữa xử lý song song trên GPUs và xử lý tuần tự trên CPU [1] [2] [3] [7] [10].

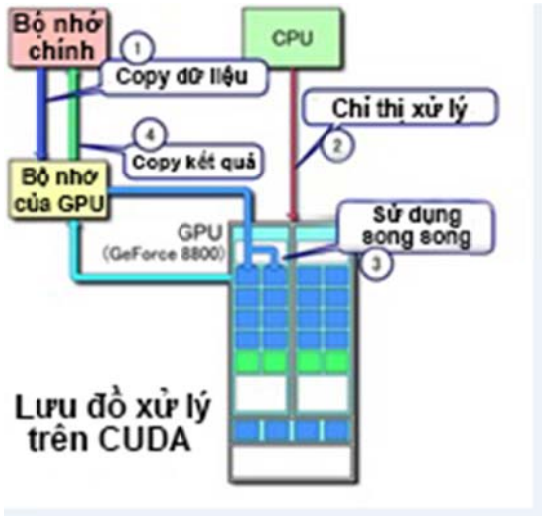
Trong bài viết này, chúng tôi đề xuất một hướng tiếp cận mới trong việc cải thiện tốc độ tìm kiếm của mô hình đồ thị BT-Graph bằng phương pháp song song hóa thuật toán tìm kiếm dựa trên nền tảng GPUs CUDA NVIDIA [6] [15] [16] [26]. Các thực nghiệm được triển khai dựa trên hai thuật toán: tìm kiếm k-láng giềng gần nhất [8] [13] [19] [23] [24] [25] và tìm kiếm đường đi ngắn nhất [5] [9] [17] [18] [27] [29].

Bài viết được chia thành năm phần. Phần thứ nhất giới thiệu về mô hình BT-Graph và tìm kiếm vị trí địa lý dựa trên mô hình. Phần thứ hai trình bày về CUDA NVIDIA. Phần thứ ba trình bày về cải thiện tốc độ tìm kiếm của mô hình đồ thị BT-Graph dựa trên nền tảng CUDA. Phần thứ tư trình bày về các thực nghiệm. Phần cuối cùng là phần kết luận.

II. CUDA NVIDIA

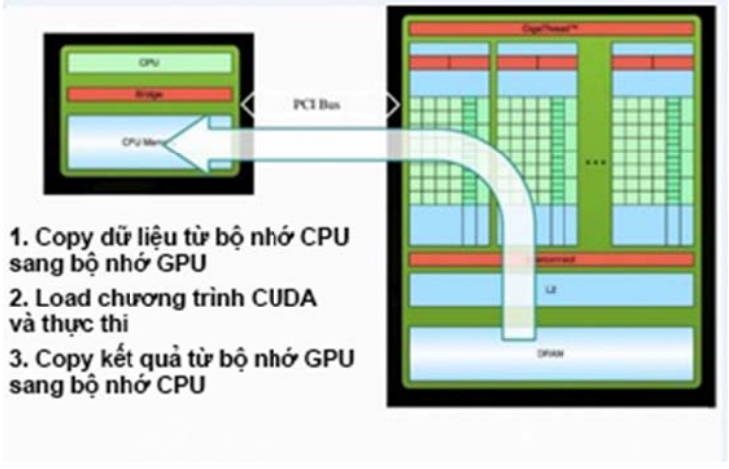
CUDA [26] là một mô hình lập trình và là một nền tảng tính toán song song được phát triển bởi Công ty NVIDIA. CUDA cung cấp khả năng kết hợp giữa kiến trúc phần cứng và phần mềm. CUDA có khả năng tăng đáng kể hiệu suất tính toán bằng cách khai thác sức mạnh của đơn vị xử lý đồ họa – Graphis Processing Units (GPUs).

GPUs [6] [16] [28] hỗ trợ đa luồng khổng lồ - nhiều lõi, với số lượng lên đến hàng trăm lõi và hàng ngàn luồng. Với số lượng lớn các lõi GPUs cung cấp một khả năng xử lý dữ liệu song song, chính vì điều đó GPUs được sử dụng rộng rãi trong xử lý song song. GPUs được sử dụng để giải quyết nhiều vấn đề phức tạp trong mô hình hóa và mô phỏng như: mô phỏng khí hậu, dịch bệnh,…



Hình 2. Lưu đồ xử lý của CUDA

CUDA cung cấp một tập hợp các thư viện mở rộng hỗ trợ lập trình viên trong việc phát triển các thuật toán song song. Cả CPU và GPU đều tham gia vào quá trình tính toán. Các tính toàn tuần tự sẽ được thực thi trên CPU, trong khi các tính toán song song sẽ do GPU xử lý với bộ nhớ riêng biệt.



Hình 3. GPU và CUDA giao tiếp với bộ cấp phát bộ nhớ

III. CẢI THIỆN TỐC ĐỘ TÌM KIẾM CỦA MÔ HÌNH ĐỒ THỊ BT-GRAPH DỰA TRÊN CUDA

A. Thuật toán tìm kiếm k-láng giềng gần nhất của mô hình đồ thị BT-Graph dựa trên CUDA

Tìm kiếm k-láng giềng gần nhất trên mô hình đồ thị BT-Graph (Search based on BT-Graph, viết tắt là BTS) [11] được thể hiện như là phương pháp tìm kiếm k vị trí địa lý gần nhất được áp dụng trên hệ thống mạng các bẫy đèn tự động tại một không gian địa lý xác định. Với V là tập hợp các vị trí địa lý (bẫy đèn), Q chứa các điểm láng giềng của truy vấn q trong V, k là số điểm gần nhất cần tìm. Quá trình tìm kiếm được bắt đầu từ nút gốc, trong suốt quá trình tìm kiếm giải thuật sẽ tính toán lại Q. Tại mỗi nút B đang xét, giải thuật thực hiện một trong ba trường hợp, cuối cùng trả về Q chứa k vị trí có cùng điều kiện gần nhất của truy vấn q. Trường hợp một nếu khoảng cách từ điểm truy vấn q đến nút đang xét B lớn hơn D, bỏ qua B và trả kết quả là Q. Trường hợp hai nếu B là nút lá, duyệt qua tất cả các điểm $x \in B$ và cập nhật lại Q. Trường hợp ba nếu B là một nút trong, gọi đệ quy thuật toán tìm kiếm cho hai nút con của B là con trái và con phải. Chi tiết giải thuật được mô tả như trong [11].

Tuy nhiên, khi số lượng vị trí địa lý lớn và không gian tìm kiếm mở rộng thì cần cải thiện tốc độ tìm kiếm của BTS. Ngoài ra khi số lượng điểm truy vấn lớn cũng cần phải cải thiện tốc độ tìm kiếm. Vì vậy, chúng tôi đề xuất hai trường hợp giải quyết bài toán tìm kiếm k-láng giềng gần nhất bằng phương pháp song song hóa.

Trường hợp một chúng tôi đề xuất sử dụng thuật toán vét cạn áp dụng cho tìm kiếm k-láng giềng dựa trên nền tảng CUDA (gọi tắt là BF-kNNCUDA) [2] [3] [8] [10] [19] [23] [25] và được cài đặt với hai module chính. Module một thực hiện tính toán song song khoảng cách từ điểm truy vấn đến tất cả các điểm trong tập dữ liệu – Thực hiện tại GPU. Module hai thực hiện sắp xếp các khoảng cách tính toán được theo thứ tự tăng dần và chọn ra k-khoảng cách nhỏ nhất (gần nhất) – Thực hiện tại CPU.

Giải thuật 1: BF-kNNCUDA

//Xử lý tại CPU

1. Nạp dữ liệu vào bộ nhớ dùng chung
2. Thiết lập chỉ số k
3. Sao chép dữ liệu từ CPU vào GPU

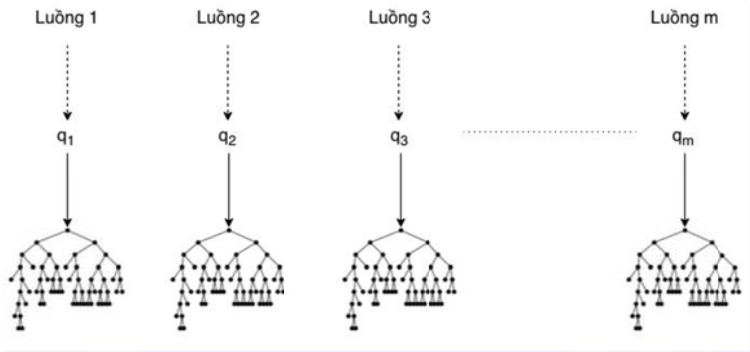
//Xử lý tại GPU

4. Tính toán khoảng cách từ điểm truy vấn q đến tất cả điểm $v \in V$
5. Sao chép dữ liệu từ GPU về CPU

//Xử lý tại CPU

6. Sắp xếp các khoảng cách tính được theo thứ tự tăng dần
7. Lấy 'k' khoảng cách đầu tiên thể hiện cho các điểm gần nhất
8. Giải phóng bộ nhớ CUDA

Trường hợp hai chúng tôi đề xuất với mỗi lần thực hiện BTS trên một điểm truy vấn thì sẽ do một luồng trong CUDA xử lý [4]. Phương pháp này chúng tôi gọi là BTSCUDA. Ý tưởng thuật toán được thể hiện trong hình 4 và giải thuật 2.



Hình 4. BTSCUDA với mỗi điểm truy vấn sẽ do một luồng trong CUDA xử lý

Giải thuật 2: BTSCUDA

//Xử lý tại CPU

1. Tạo cấu trúc cây T
2. Tạo mảng chứa các điểm truy vấn qArrray
3. Sao chép cây T và mảng qArrray từ CPU vào GPU

//Xử lý tại GPU

4. Với mỗi điểm truy vấn
5. Tìm kiếm trên cây T với phương pháp BFS
6. Trả kết quả là danh sách k-láng giềng gần nhất của mỗi điểm truy vấn
7. Sao chép kết quả tất cả danh sách k-láng giềng tìm được từ GPU về CPU

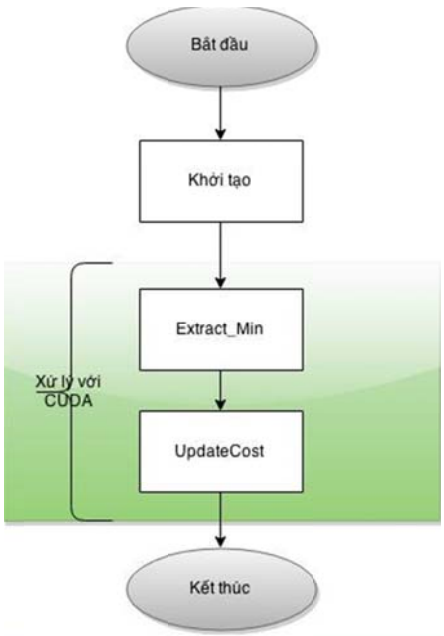
B. Thuật toán tìm kiếm đường đi ngắn nhất của mô hình đồ thị BT-Graph dựa trên CUDA

Thuật toán Dijkstra [5] [26] là thuật toán tìm kiếm đường đi ngắn nhất bằng phương pháp duyệt qua đồ thị và tìm kiếm đường đi sao cho chi phí duyệt từ đỉnh bắt đầu đến đỉnh kết thúc là nhỏ nhất. Thuật toán Dijkstra tuần tự được xây dựng trên cơ sở gán cho các đỉnh của đồ thị các nhãn tạm thời. Các nhãn này được thay đổi theo mỗi bước lặp tính toán. Có hai nhãn là cố định và tạm thời. Ở mỗi bước lặp sẽ thay đổi một nhãn tạm thời thành nhãn cố định. Một nút được đánh dấu là nhãn cố định sẽ cho kết quả là đường đi ngắn nhất từ đỉnh tới nút đó. Thuật toán bao gồm ba bước cơ bản: khởi tạo (Initialization), tìm giá trị nhỏ nhất (Extract_Min) và cập nhật giá trị (UpdateCost).

Giải thuật 3: Thuật toán Dijkstra

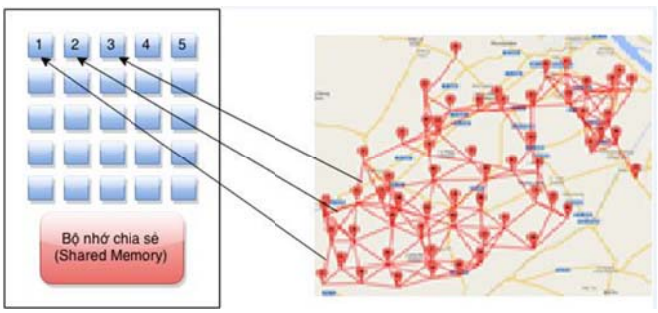
```
    //Initial
1.  foreach v ∈ V do
2.          d[v] ← ∞
3.      end
4.      d[S] ← 0
5.      Q ← V
6.  while Q ≠ ∅ do
      //Extract_Min
7.    u ← { v: v ∈ Q ^ ∀w ∈ Q, d[v] ≤ d[w] }
8.    if (d[u] = ∞) then
9.      break;
10.   Remove u from Q
      //UpdateCost
11.   foreach (u, v) ∈ V do
12.     if ((d[u] + c(u, v)) < d[v]) then
13.       d[v] ← d[u] + c(u, v)
14.   end
15. end
```

Để song song hóa thuật toán Dijkstra, chúng tôi đề xuất mỗi cạnh của đồ thị BT-Graph sẽ tương ứng với một luồng trong CUDA. Quá trình xử lý song song được thể hiện như lưu đồ ở hình 5. Trong đó, hai bước Extract_Min và UpdateCost được cài đặt bằng CUDA.



Hình 5. Lưu đồ song song thuật toán Dijkstra trên mô hình đồ thị BT-Graph sử dụng CUDA

Quá trình cấp phát bộ nhớ GPUs cho các cạnh của đồ thị BT-Graph được thể hiện như hình 6.



Hình 6. Cấp phát bộ nhớ lưu trữ cho mô hình đồ thị BT-Graph

Thuật toán Dijkstra cài đặt với CUDA (gọi tắt là DijkstraCUDA) được thể hiện như sau:

Giải thuật 4: Thuật toán DijkstraCUDA

Bước 1:

1. Khởi tạo ma trận trọng số, điểm bắt đầu và kết thúc

Bước 2:

2. Cấp phát bộ nhớ trong CUDA tương ứng với số cạnh của mô hình đồ thị BT-Graph

Bước 3:

3. Sao chép dữ liệu từ CPU sang GPU

Bước 4:

4. Tìm kiếm đỉnh u tự do sao cho chi phí đi từ đỉnh xuất phát S đến u là nhỏ nhất
5. Nếu không tìm thấy u thỏa điều kiện trên thì thoát:
   + Hoặc là tìm thấy đường đi.
   + Hoặc không tìm thấy đường đi.
6. Nếu tìm thấy đỉnh u thỏa điều kiện, dùng đỉnh u xét các đỉnh tự do khác.
7. Dùng CUDA để cấp các luồng cho việc thực hiện tính toán giữa đỉnh u và các đỉnh còn lại.

Bước 5:

8. Sao chép dữ liệu từ GPU về CPU

Bước 6:

9. Giải phóng CUDA

IV. THỰC NGHIỆM

Trong phần thực nghiệm này chúng tôi tiến hành so sánh tốc độ tìm kiếm trên mô hình đồ thị BT-Graph giữa thuật toán tuần tự trên CPU và thuật toán song song trên CUDA. Máy tính được sử dụng cho phần thực nghiệm này là Desktop (Intel Core i3-3220 3.3 GHz, bộ nhớ 4GB DDR3, card đồ họa NVIDIA GeForce GTX 660 với bộ nhớ 2GB GDDR5) chạy hệ điều hành Ubuntu 14.04 64 bits.

A. Dữ liệu thực nghiệm

Dữ liệu sử dụng cho thực nghiệm được chia làm hai loại, loại một dùng cho thực nghiệm tìm kiếm k-láng giềng và loại hai dùng cho tìm kiếm đường đi ngắn nhất Dijkstra. Các dữ liệu được sinh ra ngẫu nhiên từ chương trình thực nghiệm.

Cấu trúc thông tin của dữ liệu loại một bao gồm hai danh sách. Danh sách thứ nhất chứa các điểm dữ liệu ban đầu gồm ba cột và số dòng là tùy chọn. Mỗi dòng mô tả thông tin một điểm dữ liệu. Cột một chứa tên của điểm dữ liệu, cột hai chứa giá trị x trong không gian hai chiều và cột ba chứa giá trị y trong không gian hai chiều. Danh sách thứ hai chứa các điểm truy vấn cũng gồm ba cột và số dòng là tùy chọn. Mỗi dòng mô tả thông tin một điểm truy vấn. Cột một chứa tên của điểm dữ liệu, cột hai và cột ba chứa giá trị x và y trong không gian hai chiều.

Cấu trúc thông tin của dữ liệu loại hai dùng để mô tả số đỉnh, cạnh và thông tin các cạnh của mô hình đồ thị BT-Graph. Dòng thứ nhất chứa hai giá trị, số đỉnh và số cạnh của mô hình đồ thị BT-Graph. Các dòng còn lại, mỗi dòng chứa ba giá trị - đỉnh đầu, đỉnh cuối và trọng số của cạnh được tạo bởi hai đỉnh đó.

B. Công cụ thực nghiệm và phương pháp thực nghiệm

Trong thực nghiệm này, chúng tôi đã xây dựng một công cụ dựa trên nền tảng NetGen [14] với tên gọi là: GLS (Geographical Location Search) [11], cho phép xây dựng mô hình đồ thị BT-Graph [28] dựa trên tập dữ liệu cho trước. Ngoài ra, chương trình còn cho phép tính toán, thực thi chương trình CUDA thông qua thư viện DLL&C của Smalltalk và so sánh thời gian thực thi của các thuật toán.

Trong cùng một thuật toán và một tập dữ liệu, chúng tôi tiến hành thực thi chương trình nhiều lần và lấy kết quả trung bình về thời gian thực thi của thuật toán đó nhằm mục đích để thu được kết quả tương đối chính xác.

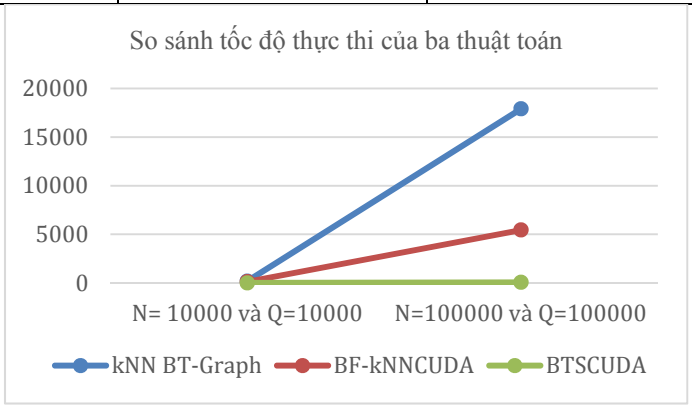
C. So sánh hiệu suất của thuật toán tìm kiếm k-láng giềng

Yêu cầu đặt ra cho phần thực nghiệm này là so sánh được tốc độ tìm kiếm của thuật toán kNN truyền thống [11] trên mô hình đồ thị BT-Graph và thuật toán kNN dựa trên CUDA [26].

Thực nghiệm được tiến hành trên tập dữ liệu được mô tả như trong phần dữ liệu thực nghiệm. Chúng tôi tiến hành thực thi chương trình 10 lần trên mỗi thuật toán và lấy giá trị trung bình theo thời gian thực thi. Kết quả thực nghiệm của chương trình sau khi đã lấy trung bình được thể hiện như trong bảng 1. Trong đó, N là số lượng điểm dữ liệu ban đầu, Q là số lượng điểm truy vấn.

Bảng 1. Bảng so sánh tốc độ thực thi của các thuật toán (Đơn vị tính bằng mili giây)

	N =10000 với Q= 10000	N = 100000 với Q=100000
kNN BT-Graph	179.46	17898.13
BF-kNNCUDA	123.47	5440.29
BTSCUDA	1.12	53.478



Hình 7. Biểu đồ so sánh tốc độ thực thi của ba thuật toán

Dựa vào bảng thực nghiệm 1 và biểu đồ so sánh ở hình 7, chúng ta thấy rằng thuật toán BF-kNNCUDA cải thiện tốc độ tìm kiếm hơn gấp khoảng 2.37 lần so với thuật toán kNN BT-Graph. Trong khi đó thuật toán BTSCUDA cải thiện tốc độ gấp khoảng 247.46 lần so với thuật toán kNN BT-Graph và cải thiện tốc độ gấp khoảng 105.98 lần so với BF-kNNCUDA.

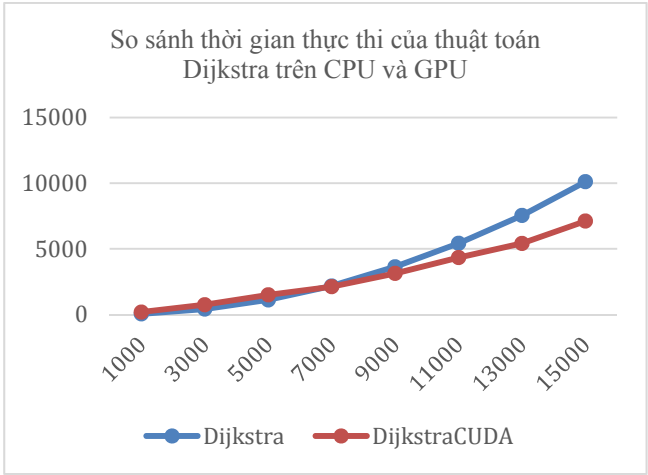
D. So sánh hiệu suất của thuật toán tìm kiếm đường đi ngắn nhất

Yêu cầu đặt ra cho phần thực nghiệm này là so sánh được tốc độ tìm kiếm của thuật toán Dijkstra truyền thống trên mô hình đồ thị BT-Graph và thuật toán DijkstraCUDA. Trong phần này, chúng tôi vẫn tiến hành thực thi chương trình 10 lần cho từng thuật toán và sau đó lấy kết quả trung bình trên tổng số lần thực thi đó.

Dữ liệu đầu vào là tập dữ liệu được mô tả như trong phần dữ liệu thực nghiệm. Sau đó xây dựng ma trận trọng số trên mô hình đồ thị BT-Graph. Tiến hành tìm kiếm trên ma trận trọng số đó bằng hai phương pháp: tuần tự và song song với CUDA. Kết quả đầu ra của chương trình là thời gian thực thi của hai phương pháp. Kết quả thực nghiệm với số lượng đỉnh của đồ thị khác nhau được thể hiện như bảng 2.

Bảng 2. Thể hiện kết quả thực nghiệm của hai thuật toán (Đơn vị tính là mili giây)

	1000	3000	5000	7000	9000	11000	13000	15000
Dijkstra	47	413	1116	2173	3641	5422	7545	10111
DijkstraCUDA	189	753	1498	2120	3130	4338	5414	7118



Hình 8. Biểu đồ so sánh tốc độ thực thi của thuật toán Dijkstra trên CPU và GPU

Từ bảng thực nghiệm 2 và biểu đồ so sánh thuật toán Dijkstra trên CPU và GPU trong hình 8, ta thấy khi số lượng đỉnh trong một đồ thị nhỏ và nằm trong khả năng xử lý của CPU thì khi đó CPU sẽ có thời gian thực thi nhanh hơn so với GPU. Nguyên nhân là do phải sao chép dữ liệu từ CPU sang GPU và tiến hành xử lý tính toán sau đó sao chép kết quả

ngược về GPU. Tuy nhiên, khi số lượng đỉnh của một đồ thì đủ lớn, thì chúng ta có thể thấy thời gian xử lý của GPU được cải thiện đáng kể. Cụ thể như trong như trong bảng 2 và hình 8, khi số lượng đỉnh nhỏ hơn 9000 thì CPU xử lý nhanh hơn GPU. Nhưng khi số lượng đỉnh đủ lớn (lớn hơn 9000) thì tốc độ của GPU nhanh hơn so với CPU.

V. KẾT LUẬN

BT-Graph (Graph Model based on Ball Tree Structure) [11] là một mô hình đồ thị được xây dựng dựa trên cấu trúc balltree, giúp mô hình hóa hệ thống mạng giám sát các bẫy đèn tự động và hỗ trợ tìm kiếm vị trí địa lý. Khi số lượng vị trí địa lý lớn và không gian địa lý tìm kiếm mở rộng thì cần phải cải thiện tốc độ tìm kiếm của mô hình đồ thị BT-Graph. Chính vì vậy chúng tôi đã đề xuất một hướng tiếp cận mới trong cải thiện tốc độ tìm kiếm của mô hình đồ thị BT-Graph bằng phương pháp song song hóa dựa trên nền tảng CUDA.

Trên nền tảng CUDA, chúng tôi đã tiến hành song song hóa hai thuật toán tìm kiếm của mô hình đồ thị BT-Graph – BF-kNNCUDA, BTSCUDA và DijkstraCUDA. Từ đó, công cụ GLS được tạo ra nhằm xây dựng mô hình đồ thị BT-Graph, đồng thời cho phép tính toán hiệu suất hoạt động của các thuật toán và đưa ra kết quả so sánh.

Kết quả thực nghiệm cho thấy việc song song hóa thuật toán tìm kiếm của mô hình đồ thị BT-Graph cho một kết quả tốt về thời gian tìm kiếm. Trong thời gian tới, chúng tôi sẽ tiến hành song song hóa toàn bộ quá trình xây dựng mô hình đồ thị BT-Graph nhằm tiếp tục cải thiện tốc độ trong toàn bộ quá trình xây dựng mô hình.

VI. TÀI LIỆU THAM KHẢO

[1] Andrew O. Finley, Ronald E. McRoberts, "Efficient k-nearest neighbor searches for multi-source forest attribute mapping", *Remote Sesing of Environment (Volume 112, Issue 5)*, pp. 2203-2211, 2008.

[2] Carlo del Mundo, Mariam Umar, "A Mixed Hierarchical Algorithm for Nearest Neighbor Search", 2013.

[3] Deepika Verma, Namita Kakkar, Neha Mehan, "Comparison of Brute-Force and K-D tree Algorithm", *International Journal of Advanced Research in Computer and Communication Engineering (Volume 3, Issue 1)*, 2014.

[4] Dr. Mohammed Otair, "APPROXIMATE K-NEAREST NEIGHBOUR BASED SPATIAL CLUSTERING USING K-D TREE", *Internaltional Journal of Database Management Systems (Volume 5, Issue 1)*, pp. 97, 2013.

[5] E. W. Dijkstra, "A note on two problems in connexion with graphs", *Numerische Mathematik (volume 1, Issue 1)*, pp. 269-271, 1959.

[6] Ebook: CUDA, http://www.nvidia.com/docs/IO/116711/sc11-cuda-c-basics.pdf.

[7] Fabian Gieseke, Justin Heinermann, Cosmin Oancea, Christian Igel, "Buffer k-d Trees: Processing Massive Nearest Neighbor Queries on GPUs", *Proceeding of The 31st Internaltional Conference on Machine Learning*, pp. 172-180, 2014.

[8] Graham Nolan, "Improving the k-Nearest neighbour Algorithm with CUDA", *Technical report, School of CSSE, The University of Western Australia*, 2009.

[9] Gunjan Singla, Amrita Tiwari, Dhirendra Pratap Singh, "New Approach for Graph Algorithms on GPU using CUDA", *Internaltion Journal of Computer Application (Volume 72, Issue 18),* pp. 38-42, 2013.

[10] Kimikazu Kato, Tikara Hosino, "Solving k-Nearest Neighbor Problem on Multiple Graphics Processors", *CCGrid 2010 – 10th IEEE/ACM International Conference on Cluster, Cloud, and Grid Computing*, pp. 769-773, 2009.

[11] Lương Hoàng Hướng, Huỳnh Xuân Hiệp, "Mô hình đồ thị tìm kiếm vị trí địa lý dựa trên cấu trúc BallTree", Kỷ yếu hội thảo quốc gia lần thứ XVII, Trang 116 – 123, 2014.

[12] Mike Izbicki, Christian Shelton, "Faster Cover Trees", *Proceeding of The 32nd Internaltional Conference on Machine Learning*, pp. 1162-1170, 2015.

[13] N. Bhatia, "Survey of nearest neighbor techniques", *arXiv Prepr. arXiv 1007.0085 (Vol. 8, No. 2)*, pp. 302-305, 2010.

[14] NetGen website, "A generator of concurrent systems", http://wsn.univ-brest.fr/NetGen/News

[15] NVIDIA document, "CUDA SAMPLES", http://docs.nvidia.com/cuda/cuda-samples

[16] NVIDIA Website: About CUDA, https://developer.nvidia.com/about-cuda

[17] P. Harish, P. J. Narayanan, "Accelerating Large Graph Algorithm on the GPU using CUDA", *Proceeding HiPC'07 Proceeding of the 14th international conference on High performance computing (Volume 4873)*, pp. 197-208, 2007.

[18] Pedro J. Martin, Roberto Torres, Antonio Gavilanes, "CUDA Solution for the SSSP Problem", *Proceeding ICCS'09 Proceeding of the 9th International Conference in Computational Science: Part I (Volume 5544)*, pp. 904-913, 2009.

[19] Quansheng Kuang, Lei Zhao, "A Practical GPU Based KNN Algorithm", *Proceedings of the Second Symposium International Computer Science and Computational Technology*, pp. 151-155, 2009.

[20] S. Dhanabal and Dr. S. Chandramathi. "A Review of various k-Nearest Neighbor Query Processing Techniques", *International Journal of Computer Applications 31(7), Published by Foudation of Computer Science, New York, USA*, pp. 14-22, October 2011.

[21] S. M. Omohundro, "Five Balltree Construction Algorithms", *ICSI Technical Report*, 1989.

[22] T. Liu, A. W. Moore, A. G. Gray, and K. Yang, "An Investigation of Pratical Approximate Nearest Neighbor Algorithms", *In Proceedings of NIPS (TR-89-063)*, 2004.

[23] V. B. Nikam, B. B. Meshram, "PARALLEL KNN on GPU ARCHITECTURE USING OpenCL", *Internaltional Journal of Research in Engineering and Technology*, pp. 367–372, 2014.

[24] Vincent Garcia, Éric Debreuve, Frank Nielsen, Michel Barlaud, "K-NEAREST NEIGHBOR SEARCH: FAST GPU-BASED IMPLEMENTATIONS AND APPLICATION TO HIGH-DIMENSIONAL FEATURE MATCHING", *Image Processing (ICIP), 17th IEEE International Conference*, pp. 3757-3760, 2010.

[25] Vincent Garcia, Frank Nielsen, "Searching High-Dimensional Neighbours: CPU-Based Tailored Data-Structure Versus GPU-Based Brute-Force Method", *Lecture Notes in Computer Science, Springer Berlin Heidelberg (Volume 5496)*, pp. 425-436, 2009.

[26] Wiki: CUDA, http://en.wikipedia.org/wiki/CUDA

[27] Wiki: Dijkstra, http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

[28] Wiki: GPU, http://en.wikipedia.org/wiki/Graphics_processing_unit

[29] Yangzihao Wang, John Owens, "Large-Scale Graph Processing Algorithms on the GPU", *Technical Report*, 2013.

IMPROVING THE SEARCH SPEED OF BT-GRAPH MODEL BASED ON CUDA

Luong Hoang Huong, Nguyen Hai Thanh, Huynh Xuan Hiep

ABSTRACT *- BT-Graph (Graph Model based on Ball Tree structure) is a graph model which is built based on Balltree structure. This model is used to model the automatic BPH surveillance light trap network and search the geographical position. In some cases, when a number of geographical positions are large and searched space is expanded, it is necessary to improve search speed of BT-Graph model. In this paper, we propose a new approach to improve the search speed of BT-Graph model by using the parallel method based on CUDA NVIDIA. The experiments are deployed on k-Nearest Neighbor search algorithm (kNN) and shortest path search algorithm (Dijkstra). The results of this approach showed that search results are good.*