

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN,
ĐẠI HỌC QUỐC GIA HÀ NỘI



GPU-accelerated 2D Convex Hull Computation

Ngày 8 tháng 12 năm 2023

Nhóm 6:

Dương Thanh Hải,
Mai Thanh Hảo,
Tôn Nữ Mai Khanh,
Hoàng Thảo Nguyên.

Giảng viên hướng dẫn:
Thầy Trần Hà Nguyên

Mục lục

1 Giới thiệu	4
2 Các phương pháp đã nghiên cứu	5
2.1 Thuật toán QuickHull song song	5
2.1.1 Ý tưởng chính	5
2.1.2 Nhận xét	5
2.2 Tiền xử lý dữ liệu	6
2.2.1 Ý tưởng chính	6
2.2.2 Nhận xét	7
3 Phương pháp đề xuất	7
3.1 Background	7
3.1.1 Bài toán tìm bao lồi	7
3.1.2 Các thuật toán giải bài toán tìm bao lồi	8
3.2 Phương pháp giải bài toán cụ thể	11
3.3 Phân tích thiết kế	14
3.3.1 Tiền xử lý các điểm lần thứ nhất	15
3.3.2 Tiền xử lý các điểm lần thứ hai	15
4 Kết quả thí nghiệm	16
4.1 Bộ dữ liệu	16
4.2 So sánh	16
4.2.1 Hiệu quả của xử lý dữ liệu trên GPU	16
4.2.2 Hiệu quả của quá trình tiền xử lý	17
5 Ứng dụng và Giao diện người dùng	17
6 Kết luận	20
7 Lời cảm ơn	20

Tóm tắt

Báo cáo trình bày cách triển khai phương pháp Sorting-based Preprocessing Approach (SPA) để loại bỏ các điểm nằm bên trong và trình bày một thuật toán thực hiện trên GPU, được gọi là CudaChain, để tìm bao lồi của các tập điểm trên mặt phẳng thông qua việc sử dụng thuật toán SPA. Nhóm cũng đã áp dụng phương pháp trên để tính toán bao lồi cho một tập hợp điểm trên mặt phẳng và áp dụng thuật toán tìm bao lồi trong việc xác định vị trí của trạm phát sóng. Đồng thời nhóm đã đưa ra một số phương pháp tính toán bao lồi tuần tự và song song đã có trước đó để chỉ ra sự khác biệt giữa các phương pháp đã có và phương pháp nhóm đề xuất. Để đánh giá kết quả của thuật toán, nhóm đã chạy các thuật toán trên 7 bộ dữ liệu gồm 100.000 điểm, 200.000 điểm, 500.000 điểm, 1.000.000 điểm, 2.000.000 điểm, 5.000.000 điểm, 10.000.000 điểm. Kết quả chỉ ra rằng: Phương pháp được giới thiệu có thể đạt tốc độ nhanh hơn 5-6 lần so với phương pháp QuickHull trong trường hợp tốt nhất.

1 Giới thiệu

Bài toán tìm bao lồi là một trong những bài toán đặc biệt quan trọng trong lĩnh vực hình học tính toán bởi các ứng dụng đa dạng của nó như: nhận dạng mẫu, xử lý ảnh, tìm đường đi cho robot, số liệu thống kê,... Hơn nữa bài toán tìm bao lồi còn được áp dụng để tìm ra các bài toán tính toán hình học khác như tìm tam giác phân Delaunay, tìm đường kính của một tập hợp, tìm các lớp lồi của một tập hợp,... Vì các ứng dụng quan trọng của nó nên nhiều nhà khoa học đã nghiên cứu và đưa ra các thuật toán tìm bao lồi của một tập hợp. Điển hình như các thuật toán của Vasyl Tereshchenko, Yaroslav Tereshchenko và Dmytro Kotsu với thuật toán "Graham Scan" [3]. Qin, Jiayu and Mei, Gang and Cuomo, Salvatore and Sixu, Guo and Li, Yixuan với phương pháp tiền xử lý để tìm bao lồi [2]. Gang Mei và A. Zhang Jiayin với thuật toán Quickhull [1]. Hiện nay có rất nhiều nhà khoa học dựa vào các thuật toán đó để cải tiến và tăng tốc cho chúng nhằm xử lí các vấn đề ở tốc độ cao với số lượng lớn. Xuất phát từ lí do trên, báo cáo này đưa ra một số cải tiến nhằm tăng tốc cho việc tính toán khi tìm bao lồi của một tập hợp điểm trên mặt phẳng.

Để làm rõ ứng dụng của tính toán song song trong việc tìm bao lồi cho tập điểm phẳng, nhóm đã chạy thử các thuật toán của phương pháp đã có trên, bao gồm: thuật toán tuân tự QuickHull, thuật toán song song QuickHull và song song trong bước tiền xử lý. Từ đó phương pháp mà nhóm đề xuất để giải quyết bài toán tìm bao lồi là phương pháp tiền xử lý song song trên CUDA và thuật toán SPA.

Ở các phần còn lại của báo cáo, nhóm sẽ trình bày chi tiết về các phương pháp mà nhóm đã thử nghiệm và từ đó đưa ra phương pháp nhóm đề xuất để tìm bao lồi cùng với các số liệu và kết quả thử nghiệm để so sánh. Đồng thời nhóm cũng sẽ trình bày cụ thể về ứng dụng của phương pháp đó vào việc xác định vị trí của trạm phát sóng cùng giao diện người dùng của nó. Cuối cùng là phần kết luận của bài báo cáo về các ưu điểm và nhược điểm của phương pháp mà nhóm đưa ra cùng với hướng phát triển trong tương lai của chúng.

2 Các phương pháp đã nghiên cứu

Sau đây là phần báo cáo cụ thể về các thuật toán nhóm đã tìm hiểu và kết quả chi tiết của các thuật toán đó trên các bộ dữ liệu khác nhau.

2.1 Thuật toán QuickHull song song

2.1.1 Ý tưởng chính

Ý tưởng chính của thuật toán QuickHull song song đa phần sẽ giống thuật toán QuickHull tuần tự, điểm khác biệt lớn nhất là nếu QuickHull tuần tự, ta chia tập điểm ban đầu thành 2 tập con và tính khoảng cách, sử dụng đệ quy để chia tiếp các điểm còn lại thành các tập con thì trong thuật toán QuickHull song song, ta sẽ tính song song khoảng cách từ các điểm đến đường thẳng gần nó nhất và việc tính toán này xảy ra đồng thời trên tất cả các tập con của nó, không phải tuần tự như ở phần trên. Vậy nên độ phức tạp của thuật toán QuickHull cũng giảm đi khá nhiều.

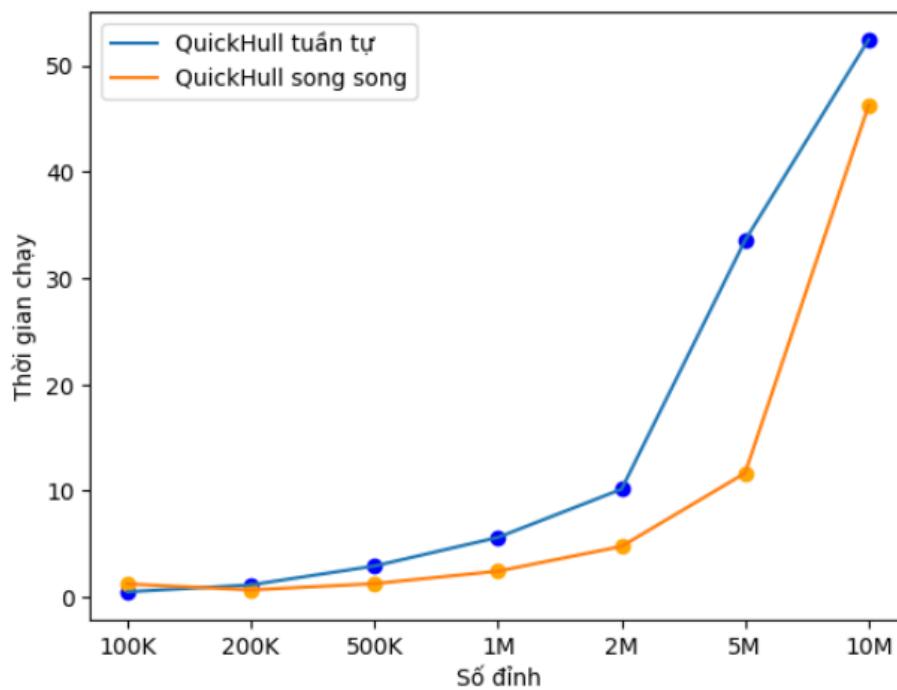
2.1.2 Nhận xét

Ưu điểm:

- Với bộ dữ liệu thực sự lớn thì thuật toán QuickHull song song vượt trội hơn rất nhiều so với thuật toán tuần tự.
- Tận dụng hiệu quả các tài nguyên phần cứng sẵn có, dẫn đến việc thực thi nhanh hơn.
- Cho phép thực hiện nhiều phép tính đồng thời và cải thiện hiệu quả tổng thể.

Nhược điểm:

- Độ phức tạp cao hơn
- Với số điểm chưa quá lớn thì việc sử dụng QuickHull tuần tự sẽ nhanh hơn và hiệu quả hơn so với sử dụng thuật toán song song. Cụ thể:



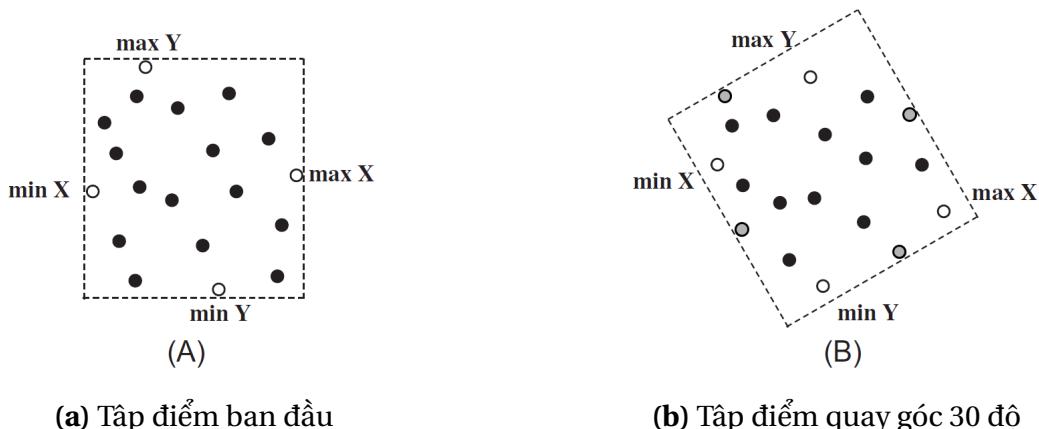
Hình 1: So sánh thời gian chạy của phương pháp QuickHull tuần tự và song song

2.2 Tiền xử lý dữ liệu

2.2.1 Ý tưởng chính

Một phương pháp tiếp cận song song sử dụng phương pháp tiền xử lý giúp cải thiện đáng kể tốc độ tính toán bao lồi là mô hình CudaCHPre2D. Việc tính toán được thực hiện trên GPU đơn và GPU kép và đã đạt được kết quả đáng kể trong việc cải thiện hiệu suất của quá trình tính toán bao lồi.

Ý tưởng cơ bản đầu tiên sau phương pháp được trình bày là loại bỏ các điểm nằm bên trong một đa giác lồi phẳng được tạo thành bởi 16 điểm cực đại, được xuất phát từ bốn nhóm điểm của (1) tập điểm gốc và (2) ba tập điểm được quay (ví dụ, 30, 45 và 60 độ). Bởi vì rõ ràng rằng các điểm cực đại của một tập điểm phẳng không thay đổi khi tất cả các điểm được quay với cùng một góc.



Hình 2: Ví dụ về tập điểm sau khi quay

Quá trình của thuật toán CudaCHPre2D bao gồm 3 bước:

- Bước 1: Xác định các điểm cực đại Ba kernel được thiết kế để quay tập điểm gốc, và mỗi luồng trong mỗi kernel đảm nhận việc tính toán vị trí mới của một điểm. Việc tìm điểm cực đại là để tìm giá trị tối thiểu và tối đa trong một vector, điều này có thể được thực hiện một cách hiệu quả bằng cách sử dụng giảm giá trị song song.

- Bước 2: Tạo một đa giác lồi Thuật toán chuỗi đơn tuyến của Andrew được chọn để tính toán bao lồi từ 16 điểm cực đại. Phần này của công việc được thực hiện trên CPU.

- Bước 3: Loại bỏ điểm nằm bên trong Các điểm nằm bên trong đa giác lồi được tạo ra ở Bước 2 sẽ bị loại bỏ. Mỗi luồng GPU có thể được gọi để kiểm tra xem một điểm có nằm trong đa giác lồi hay không.

2.2.2 Nhận xét

Ưu điểm:

- Việc thu nhỏ tập dữ liệu đầu vào bằng cách lọc bỏ các điểm được xác định chắc chắn nằm bên trong bao lồi giúp làm giảm đáng kể thời gian chạy của thuật toán QHull so với khi không có quá trình tiền xử lý.
- Cài đặt thuật toán đơn giản và trực quan

Nhược điểm:

- Thiết kế một bước xử lý trên CPU giữa hai bước xử lý trên GPU có thể tốn nhiều bộ nhớ khi phải sao chép dữ liệu từ device về host và ngược lại

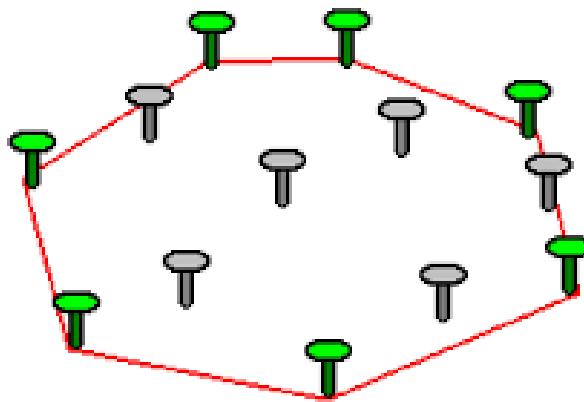
3 Phương pháp đề xuất

3.1 Background

3.1.1 Bài toán tìm bao lồi

Trong hình học tính toán (computational geometry), bao lồi (convex hull) của một tập điểm là tập lồi (convex set) nhỏ nhất (theo diện tích, thể tích, ...) mà tất cả các điểm đều nằm trong tập đó.

Theo một cách trực quan, nếu ta coi những điểm trong một tập hợp là những cái đinh đóng trên một tấm gỗ, bao lồi của tập điểm đó có viền ngoài tạo bởi sợi dây chun mắc vào những cái đinh sau khi bị kéo căng về các phía.



Hình 3: Ví dụ về bao lồi của một tập hợp

Bài toán cần giải quyết là tìm ra được tập điểm tạo thành bao lồi từ một tập phẳng các điểm sao cho thời gian thực hiện thuật toán được tối ưu nhất có thể. Khi tính toán bao lồi, một chiến thuật khá hiệu cho việc cải thiện hiệu suất tính toán là loại bỏ bớt đi những điểm nằm bên trong, được xác định từ bước tiền xử lý.

3.1.2 Các thuật toán giải bài toán tìm bao lồi

1. Thuật toán QuickHull

Ý tưởng chính của thuật toán QuickHull dựa vào phương pháp chia để trị, nó giống như thuật toán QuickSort. Cụ thể như sau:

Algorithm 1: Mã giả của thuật toán QuickHull.

Input : $Points_P$

Output : Các đỉnh của bao lồi (các đỉnh này đều thuộc P)

1 Chia tập hợp:

2 Tìm các điểm có tọa độ x_{max} và x_{min} , thêm 2 điểm đó vào bao lồi.

3 Nối 2 điểm trong bao lồi, ta được 1 đường thẳng (L) chia tập hợp P thành 2 nửa mặt phẳng là nửa trên và nửa dưới đường thẳng L.

4 Với mỗi điểm trên mặt phẳng, tính khoảng cách từ điểm đó đến đường thẳng L.

5 Các bước đệ quy:

6 for mỗi tập hợp con do

7 Lấy điểm có khoảng cách xa nhất tới đường thẳng L, cho chúng vào bao lồi.

8 Điểm đó và đường thẳng L tạo thành 1 tam giác. Các điểm nằm trong hình tam giác sẽ không thuộc bao lồi, ta xét tiếp các đỉnh nằm ngoài tam giác đó.

9 Với mỗi điểm nằm ngoài tam giác, ta được n tập hợp mặt phẳng tạo bởi nó và đường thẳng gần nó nhất. Sau đó ta tiếp tục tính khoảng cách từ các điểm tới mặt phẳng.

10 Lặp lại bước trên cho đến khi tất cả các điểm trong tập hợp P được xử lý.

Độ phức tạp của thuật toán:

Bởi vì thuật toán QuickHull có ý tưởng giống với thuật toán QuickSort. Nên độ phức tạp của nó trong trường hợp xấu nhất là $O(n^2)$. Trong trường hợp các đỉnh ban đầu là một tập hợp các đỉnh của bao lồi thì độ phức tạp của nó sẽ là $O(n \log n)$.

Kết quả thí nghiệm:

- Ngôn ngữ thực hiện: Python.

- Bộ dữ liệu thí nghiệm: 6 bộ dữ liệu được tạo random từ 100 điểm, 1000 điểm, 100.000 điểm, 200.000 điểm, 500.000 điểm và 1.000.000 điểm.

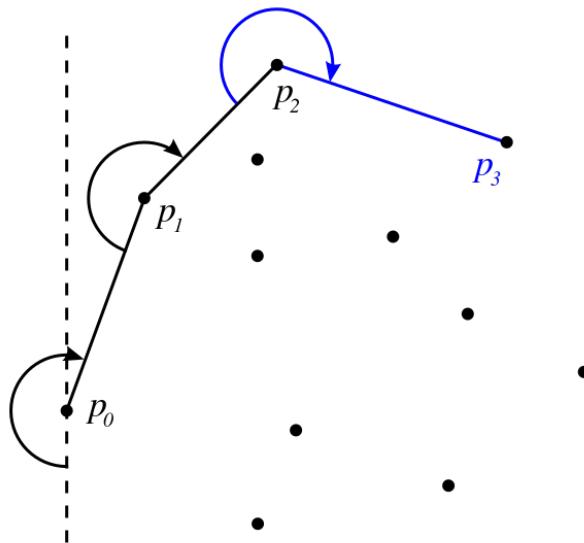
Kết quả cụ thể:

Total Points	Sequential Time(sec)	Parallel Time(sec)
100	0.001995563507080078	0.0000000000000000
1000	0.014003753662109375	0.0000000000000000
100000	1.1639657020568848	0.0000000000000000
200000	2.4665298461914062	0.0000000000000000
500000	6.061228275299072	0.0000000000000000
1000000	11.882169008255005	0.0000000000000000

Hình 4: Kết quả chạy QuickHull tuần tự

Với bộ dữ liệu nhỏ, có thể thấy QuickHull là một sự lựa chọn không tồi khi thời gian chạy của chúng khá ngắn, nhờ dựa trên phương pháp chia để trị, đồng thời dễ dàng có thể thực hiện song song trong một số trường hợp nhất định. Tuy nhiên, trong trường hợp xấu nhất độ phức tạp của thuật toán này lên đến $O(n^2)$. Quá trình cài đặt khá phức tạp và khó triển khai trong một số trường hợp cụ thể vì nó liên quan nhiều đến các phép tính đệ quy và hình học nên yêu cầu thực hiện phải thật cẩn thận.

2. **Thuật toán bọc gói quà (Gift wrapping)** Thuật toán bọc gói quà, hay còn gọi là thuật toán Jarvis march, là một trong những thuật toán tìm bao lồi đơn giản và dễ hiểu nhất. Tên thuật toán xuất phát từ sự tương tự của thuật toán với việc đi bộ xung quanh các điểm và cầm theo một dải băng gói quà.

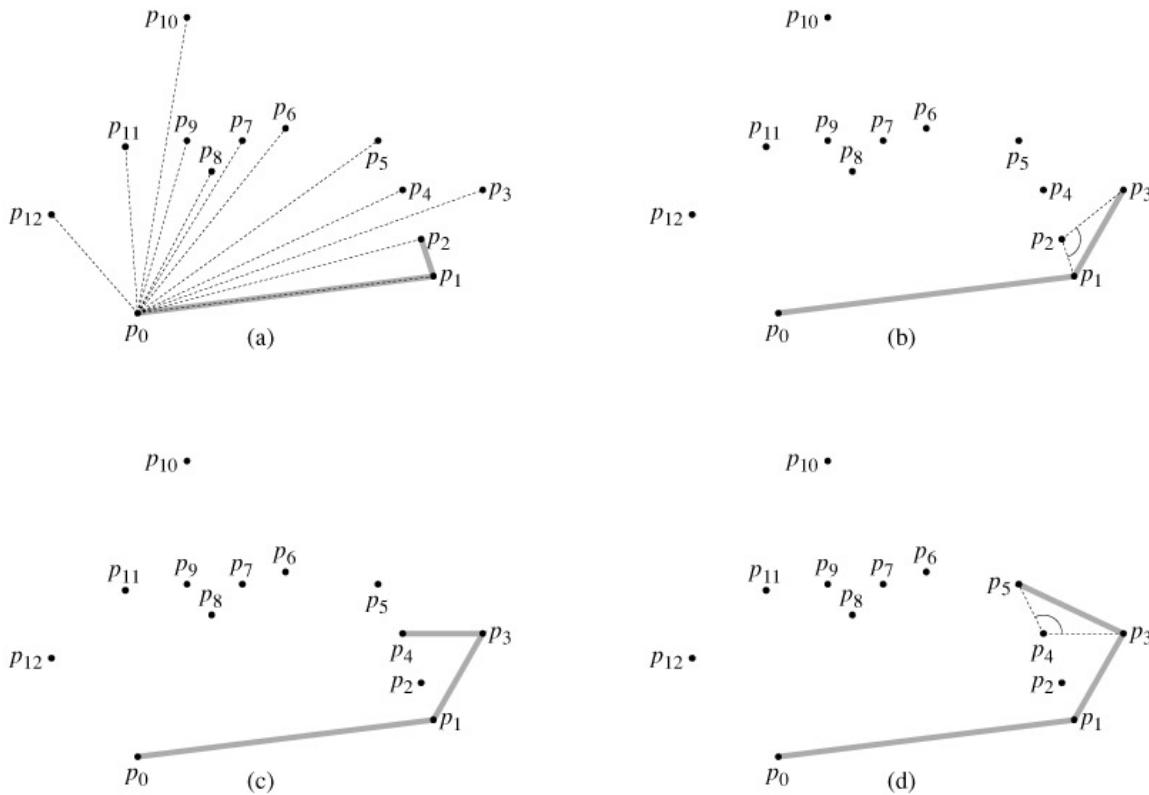


Hình 5: Thuật toán bọc gói quà

Thuật toán này được mô tả như sau:

- Bước đầu:
 - Chọn P là điểm trái nhất trong tập các điểm (để đảm bảo P nằm trong tập bao lồi).
 - Chọn hướng ta đang nhìn \vec{v} là từ hướng dưới lên trên.
- Tiếp theo, các bước sẽ lặp lại đến khi tìm được bao lồi:
 - Quay \vec{v} theo chiều kim đồng hồ cho đến khi ta nhìn thấy một điểm, gọi điểm đó là Q,
 - Cầm theo dải băng và đi đến điểm Q
 - Khi đến Q, thê \vec{v} thành \vec{PQ} và P thành Q
- Thuật toán kết thúc khi quay trở về điểm ban đầu. Lúc này ta đã đi đến tất cả các đỉnh của bao lồi theo chiều kim đồng hồ.

3. **Thuật toán Graham Scan** Thuật toán bắt đầu bằng việc xây dựng một đa giác đơn khép kín từ các điểm đã cho. Việc xây dựng này dùng một hàm: sắp các điểm theo khoá là giá trị của hàm theta tương ứng với góc giữa trực hoành và đường thẳng nối mỗi điểm chốt $p[1]$ (điểm có tung độ nhỏ nhất), và khi lần theo $p[1], p[2], \dots, p[N], p[1]$ ta sẽ được một đa giác khép kín.



Hình 6: Thuật toán Graham Scan

Độ phức tạp của thuật toán là $O(n \log(n))$. Trường hợp tốt nhất của thuật toán có thời gian chạy tuyến tính ($O(n)$) với tập điểm đã được sắp xếp.

3.2 Phương pháp giải bài toán cụ thể

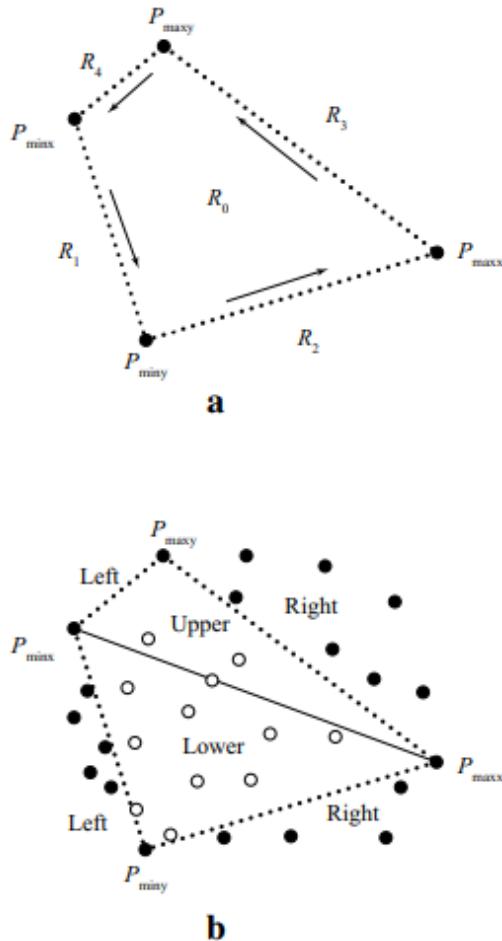
Trong báo cáo này, mục tiêu của nhóm là thiết kế và triển khai một thuật toán bao lồi thay thế và hiệu quả bằng cách tận dụng ưu thế tính toán của GPU. Các đóng góp trong việc xây dựng có thể được tóm tắt như sau: (1) đề xuất phương pháp Sorting-based Preprocessing Approach (SPA) để loại bỏ các điểm nằm bên trong; (2) trình bày một thuật toán thực hiện trên GPU, được gọi là CudaChain, để tìm bao lồi của các tập điểm trên mặt phẳng thông qua việc sử dụng thuật toán SPA.

Thuật toán tìm bao lồi đề xuất CudaChain gồm hai giai đoạn chính: (1) Hai bước tiền xử lý được thực hiện trên GPU và (2) bước tính toán cuối cùng cho ra bao lồi mong muốn thực hiện trên CPU.

Trong bước tiền xử lý, bên cạnh việc thực hiện chương trình trên GPU, nhóm em sử dụng một công cụ cho bước xác định những điểm cần loại bỏ, một chiến lược hiệu quả có thể giúp giảm bớt tập hợp đầu vào và đưa về bài toán đơn giản hơn mà không ảnh hưởng đến kết quả đầu ra. Cụ thể, (1) loại bỏ các điểm nằm bên trong và (2) tính toán bao lồi trên tập hợp điểm còn lại. Một số thuật toán cải tiến thuật toán bao lồi đã biết sẽ được trình bày trong phần dưới, được chứng minh là cải thiện thời gian thực thi tổng thể. Kết quả thử nghiệm cho thấy: (1) SPA có thể phát hiện và loại bỏ các điểm nằm bên trong một cách rất hiệu quả; và (2) CudaChain đạt được tăng tốc từ 5x đến 6x so với thực hiện Qhull nổi tiếng cho 20 triệu điểm.

Cụ thể hóa hơn, quy trình của thuật toán được đề xuất được liệt kê như sau:

1. Tìm bốn điểm cực đại có tọa độ x và y lớn nhất hoặc nhỏ nhất bằng phương pháp giảm giá trị

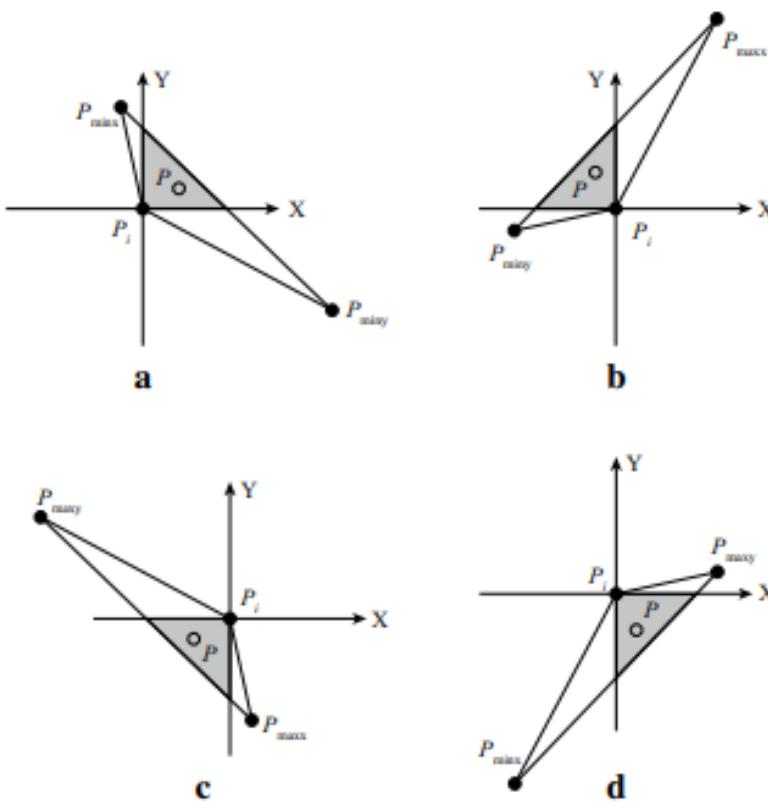


Hình 7: Phân phối các điểm và vòng đầu tiên của việc loại bỏ. a Bốn điểm cực đại; b Bốn khu vực con

song song, ký hiệu chúng là $P_{minx}, P_{maxx}, P_{miny}, P_{maxy}$

2. Xác định phân phối của tất cả các điểm song song và loại bỏ các điểm nằm bên trong tứ giác được tạo thành bởi $P_{minx}, P_{maxx}, P_{miny}, P_{maxy}$
3. Ký hiệu tập con của các điểm nằm trong bốn khu vực con, tức là phía dưới bên trái, phía dưới bên phải, phía trên bên phải và phía trên bên trái lần lượt là R_1, R_2, R_3 , và R_4
4. Sắp xếp R_1, R_2, R_3 , và R_4 một cách riêng lẻ theo phương pháp song song
5. Thực hiện SPA cho R_1, R_2, R_3 , và R_4 để loại bỏ các điểm nằm bên trong một cách hiệu quả hơn và tạo ra một chuỗi đơn giản cho các điểm còn lại trong mỗi khu vực con.
6. Tạo ra một đa giác đơn giản bằng cách kết nối bốn chuỗi đó theo hướng ngược chiều kim đồng hồ (CCW).
7. Tìm bao lồi của đa giác đơn giản bằng cách sử dụng thuật toán của Melkman (Melkman 1987).

Trong quy trình trên, chiến lược phổ biến nhất để loại bỏ điểm nằm bên trong được thực hiện trước tiên (tức là Bước 1 và 2); sau đó, những điểm còn lại được chia thành bốn tập con. Sau đó, mỗi tập con điểm được sắp xếp một cách riêng lẻ. Bước quan trọng trong quy trình này là vòng thứ hai của việc loại bỏ điểm nằm bên trong và việc tạo ra một chuỗi đơn giản cho mỗi tập con. Một đa giác đơn



Hình 8: Phân phối các điểm và vòng đầu tiên của việc loại bỏ. a Bốn điểm cực đại; b Bốn khu vực con

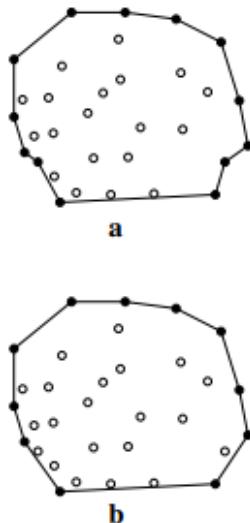
giản có thể dễ dàng được tạo ra bằng cách kết nối trực tiếp các chuỗi; và bao lồi dự kiến có thể được tìm thấy bằng cách sử dụng thuật toán của Melkman (Melkman 1987), được thiết kế đặc biệt để tính toán bao lồi của một đa giác đơn giản.

1. Bước 1: Phân phối điểm và vòng đầu tiên của việc loại bỏ. Chiến lược loại bỏ các điểm nằm bên trong một tứ giác được tạo thành bởi bốn điểm cực đại là trực tiếp và dễ hiểu; xem Hình 4. Không cần mô tả chiến lược này chi tiết hơn. Vấn đề đáng chú ý duy nhất là: để giảm chi phí tính toán, trong quá trình kiểm tra xem một điểm có phải là nằm bên trong hay không (tức là nằm trong khu vực R0 trong Hình 4a), phân phối của những điểm không nằm trong khu vực đó cũng có thể được xác định một cách dễ dàng.

Đối với tất cả các điểm, phương pháp đơn giản sau được áp dụng để xác định phân phối của chúng:

1. Nếu điểm P nằm bên phải của đường hướng $P_{minx}P_{miny}$, thì nó thuộc khu vực R1;
2. Ngược lại, nếu P nằm bên phải của đường hướng $P_{miny}P_{maxx}$, thì nó thuộc khu vực R2;
3. Ngược lại, nếu P nằm bên phải của đường hướng $P_{maxx}P_{maxy}$, thì nó thuộc khu vực R3;
4. Ngược lại, nếu P nằm bên phải của đường hướng $P_{maxy}P_{minx}$, thì nó thuộc khu vực R4;
5. Ngược lại, P thuộc khu vực R0.

Sau quy trình xác định ở trên, tất cả các điểm được phân bổ vào năm vùng. Những điểm ở vùng R0 là những điểm ở bên trong và cần được loại bỏ trực tiếp ở bước này, trong khi những điểm còn lại ở bốn vùng còn lại nên được xem xét để tính toán vùng bao lồi.



Hình 9: Hoàn chỉnh bao lồi cho đa giác đơn.a,Đa giác đơn, b,Bao lồi mong muốn

2. Bước 2: Vòng loại thứ hai và tạo đa giác đơn giản. Phần này sẽ mô tả một phương pháp tiền xử lý dựa trên sắp xếp mới mẻ mà đặc biệt áp dụng cho các điểm đã được sắp xếp trước đó. Phương pháp này được gọi là SPA. Các quy tắc để loại bỏ các điểm nội vi ở bốn vùng con, tức là dưới bên trái (R1), dưới bên phải (R2), trên bên phải (R3) và trên bên trái (R4).

Độ chính xác của SPA cho mỗi vùng con được minh họa trong Hình 5. Đối với vùng R1, điểm đầu tiên và điểm cuối cùng là P_{minx} và P_{miny} , tương ứng; xem Hình 3a. Giả sử điểm P_i đã được xác định là không phải là điểm nội vi, và bây giờ đang kiểm tra điểm P theo mối quan hệ giữa P_i và P. Vì tất cả các điểm trong vùng R1 đã được sắp xếp theo thứ tự tăng dần của tọa độ x, do đó $x_P > x_{Pi}$; và nếu $y_{Pi} > y_P$, thì điểm P phải nằm trong khu vực tam giác được đánh dấu; và rõ ràng nó cũng thuộc tam giác được hình thành bởi ba điểm P_i , P_{miny} và P_{minx} . Do đó, điểm P phải là một điểm nội vi và cần phải được loại bỏ. Độ chính xác của SPA đối với ba vùng còn lại cũng có thể được giải thích tương tự.

3. Bước 3: Tính toán bao lồi của đa giác đơn giản. Kết quả của bước trước là một đa giác đơn giản, đồng thời cũng là một ước lượng của bao lồi. Để tính toán bao lồi chính xác của tập điểm đầu vào, thuật toán nhanh được giới thiệu bởi Melkman (1987) được chọn để tính toán bao lồi của đa giác đơn giản. Bao lồi của đa giác đơn giản là bao lồi của tập dữ liệu đầu vào; xem Hình 5.

3.3 Phân tích thiết kế

Phần này sẽ nói cụ thể hơn về phương pháp được sử dụng. Chương trình được thực hiện kết hợp cả trên CPU (host) và GPU (device). Phần code trên CPU được sử dụng để tính convex hull từ tập điểm cho trước, được thiết kế tương đối đơn giản và dễ cài đặt. Vì vậy, nhóm sẽ tập trung làm rõ phần thiết kế code trên GPU.

Chương trình sử dụng ngôn ngữ Python, tận dụng các thư viện Numba, Cupy, Numpy để giảm thiểu thời gian chạy đồng thời đơn giản hóa việc lập trình các thuật toán song song.

3.3.1 Tiềm xử lý các điểm lần thứ nhất

Bước đầu tiên của quá trình loại bỏ các điểm đầu vào là xác định 4 điểm cực đại, là các điểm có tọa độ x/y lớn nhất và nhỏ nhất. Các điểm này có thể được xác định một cách nhanh chóng và hiệu quả bằng các hàm `cupy.argmin()`, `cupy.argmax()` (lấy ra chỉ số của phần tử nhỏ nhất hoặc lớn nhất) sử dụng thuật toán parallel reduction.

Một lõi kernel (`kernelPreprocess`) được thiết kế để xác định một điểm được phân vào vùng nào, gồm 1024 threads cho mỗi block. Mỗi luồng chịu trách nhiệm tính toán vị trí cho một điểm, kết quả thu được là một số nguyên được lưu vào một mảng $d_{pos}(n)$. Ví dụ, điểm P_i được xác định nằm trong vùng R_1 sẽ được có giá trị $d_{pos}[i] = 1$, các điểm nằm trong sẽ được lưu giá trị bằng 0.

3.3.2 Tiềm xử lý các điểm lần thứ hai

Trong quá trình này, chương trình được thiết kế (1) thực hiện phân tách các vùng thành các mảng riêng biệt, (2) sắp xếp các điểm trong từng vùng theo quy luật nhất định, (3) gọi tới một kernel để tiếp tục loại bỏ các điểm sử dụng phương pháp SFA, và (4) thực hiện gom nhôm các điểm còn lại sau khi xử lý.

Bước đầu tiên, phân tách các vùng sử dụng thuật toán Prefix Sum (Scan) của mảng d_{pos} với số các phần tử có trong mỗi vùng, kết quả trả về là một mảng chứa các chỉ số tương ứng trong d_{pos} với các giá trị của vùng cần phân tách. Sau đó, thực hiện so sánh các điểm theo quy luật trong bảng dưới đây.

Bảng 1: Quy luật sắp xếp các điểm tại từng vùng

Vùng	Điểm đầu	Điểm cuối	Thứ tự x	Thứ tự y
R1 (Trái dưới)	P_{minx}	P_{miny}	Tăng dần	Giảm dần
R2 (Phải dưới)	P_{miny}	P_{maxx}	Tăng dần	Tăng dần
R3 (Phải trên)	P_{maxx}	P_{maxy}	Giảm dần	Tăng dần
R4 (Trái trên)	P_{maxy}	P_{minx}	Giảm dần	Giảm dần

Tiếp theo, nhóm thiết kế 4 kernel, với mỗi kernel thực hiện loại bỏ các điểm trong mỗi vùng. Mỗi kernel chỉ bao gồm một block, mỗi thread trong block xử lý $(m + BLOCK_SIZE - 1)/BLOCK_SIZE$ điểm liên tiếp, trong đó m là tổng số điểm có trong một vùng, và $BLOCK_SIZE$ đại diện cho số thread được sử dụng trong block. Trong chương trình, $BLOCK_SIZE$ được gán giá trị bằng 1024. Sau quá trình lọc sử dụng SPA, các điểm được xác định nằm bên trong vùng tạo bởi 4 điểm cực đại sẽ được gán giá trị $d_{pos} = 0$. Lý do xây dựng một kernel với một block duy nhất là bởi sự phụ thuộc của các điểm trong quá trình so sánh và loại bỏ.

Bước cuối cùng là lấy ra danh sách chỉ số các phần tử khác 0 trong từng vùng và gộp các giá trị tương ứng đó vào mảng trả về, và được sao chép sang bộ nhớ CPU trong hàm main.

4 Kết quả thí nghiệm

Thuật toán được so sánh với thư viện Qhull với các tập dữ liệu có kích thước khác nhau. Cấu hình máy tính được sử dụng gồm lõi CPU Intel i5-3470, bộ nhớ 8GB, card đồ họa NVIDIA GeForce GT640 (GDDR5) với dung lượng RAM 1GB và 384 lõi.

4.1 Bộ dữ liệu

Để thí nghiệm chương trình, nhóm sử dụng bộ dữ liệu gồm các tập điểm có kích thước khác nhau từ 100.000 đến 20 triệu điểm. Một file dữ liệu sẽ có cấu trúc: Dòng đầu tiên gồm 2 giá trị nguyên lần lượt là số chiều của các điểm (trong trường hợp này là 2) và tổng số điểm có trong bộ dữ liệu. Mỗi dòng tiếp theo biểu diễn một bộ hai số thực tương ứng với tọa độ trục x và trục y của một điểm.

```
2 1000000
467 169
984 -257
33 572
-295 -307
214 642
-351 -952
562 -242
595 359
466 813
725 -241
922 -870
-79 827
597 -911
```

Hình 10: Ví dụ tập dữ liệu 1M điểm

4.2 So sánh

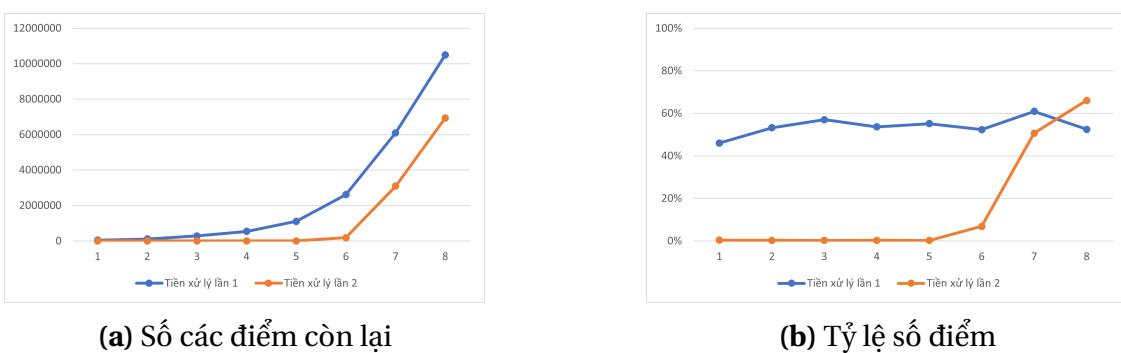
4.2.1 Hiệu quả của xử lý dữ liệu trên GPU

Kết quả thời gian chạy đối với hai tập dữ liệu được ghi trong bảng số liệu bên dưới. Với tập dữ liệu nhỏ, thời gian chạy của Qhull nhanh hơn, tuy nhiên hiệu năng không bằng với CudaChain đối với các tập dữ liệu có số điểm lớn. Tốc độ chạy của CudaChain đạt kết quả trung bình nhanh hơn xấp xỉ 3-4 lần và có thể lên tới 4-5 lần trong trường hợp tốt nhất. Kết quả thí nghiệm cũng cho thấy, khối lượng công việc thực hiện trên CPU ít hơn nhiều so với trên GPU.

Size	Qhull	CudaChain			Speedup	
		Total	GPU	CPU		
100k	15	25.5	24.2	1.3	5.10	0.59
200k	16	29.1	27.9	1.2	4.12	0.55
500k	47	40.4	38.4	2.0	4.95	1.16
1M	109	46.9	44.6	2.3	4.90	2.32
2M	202	83.5	81.2	2.3	2.75	2.42
5M	515	147.0	144.5	2.5	1.70	3.50
10M	1034	321.9	319.7	2.2	0.68	3.21
20M	2215	544.4	541.5	2.9	0.53	4.07

Hình 11: Hiệu năng của CudaChain so với thuật toán tuần tự trên cùng tập dữ liệu và môi trường

4.2.2 Hiệu quả của quá trình tiền xử lý



Hình 12: Hiệu quả của hai vòng tiền xử lý

Quá trình tiền xử lý bao gồm hai giai đoạn. Để đánh giá hiệu quả của việc thu nhỏ tập điểm đầu vào, nhóm thực hiện đếm số các điểm còn lại sau mỗi vòng xử lý. Kết quả cho thấy thuật toán SPA đã giúp làm giảm đáng kể số lượng điểm và cải thiện hiệu năng của toàn bộ mô hình thuật toán CudaChain.

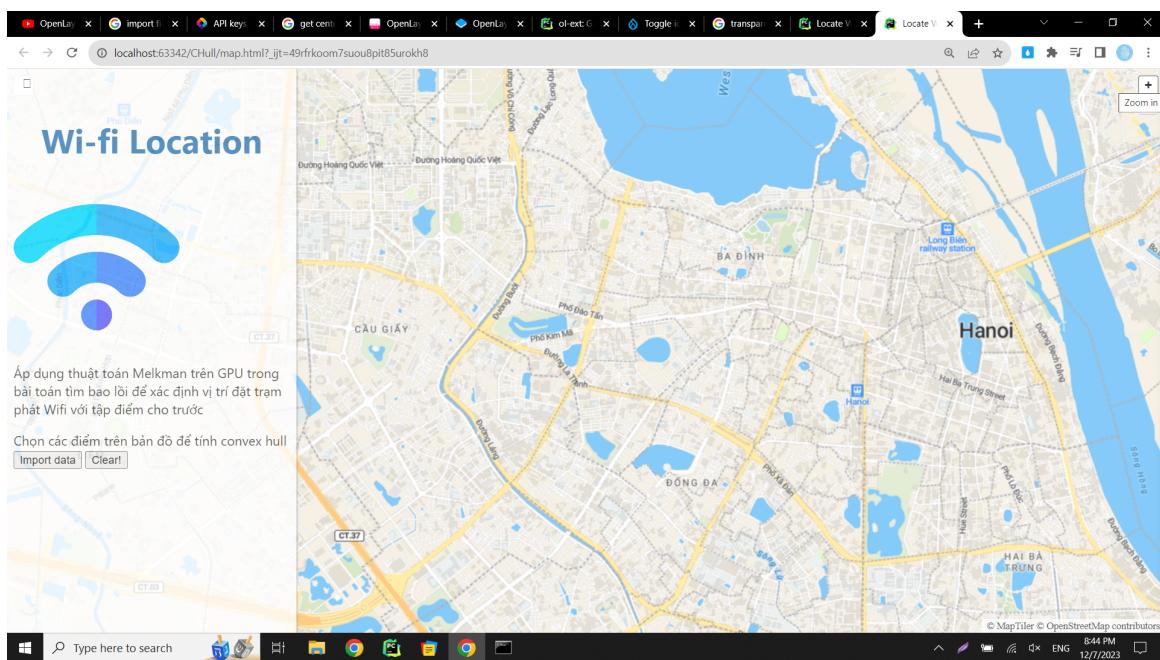
5 Ứng dụng và Giao diện người dùng

5.1. Cơ sở ứng dụng

Đầu tiên, với đầu vào là một tập điểm các vị trí trên bản đồ, nhóm sẽ áp dụng phương pháp đã đề xuất ở trên để xác định bao lồi cho tập hợp các điểm đó. Sau khi đã có tọa độ các đỉnh của bao lồi, vị trí đặt trạm phát sóng sẽ là tâm bao lồi đó do tổng khoảng cách từ tâm đến tập đỉnh đầu vào sẽ là nhỏ nhất. Vì vậy, tâm của bao lồi sẽ là vị trí thích hợp để đặt trạm phát sóng.

5.2. Tính năng

Giao diện chính của website:



Hình 13: Giao diện chính

1. Input

Người dùng có thể chọn 1 trong 2 cách sau để nhập tập điểm đầu vào:

a. Nhập trực tiếp trên bản đồ

Người dùng có thể chọn trực tiếp các điểm tọa độ trên bản đồ bằng cách click chuột trực tiếp vào bản đồ trên website. Đồng thời, bản đồ cũng sẽ tạo ra bao lồi ngay khi người dùng chọn hơn 2 điểm trên bản đồ.

b. Tải lên file GeoJSON

Người dùng cũng có thể upload file dữ liệu chứa tọa độ các điểm đầu vào mong muốn với định dạng là file GeoJson. File GeoJSON sẽ chứa các thông tin về một vị trí trên bản đồ, bao gồm tọa độ.

Ví dụ về 1 file GeoJSON:

```
chicago-parks.geojson
1  {
2    "features": [
3      {
4        "type": "Feature",
5        "properties": {
6          "title": "Lincoln Park",
7          "description": "A northside park that is home to the Lincoln Park Zoo"
8        },
9        "geometry": {
10          "coordinates": [
11            -87.637596,
12            41.940403
13          ],
14          "type": "Point"
15        }
16      },
17      {
18        "type": "Feature",
19        "properties": {
20          "title": "Burnham Park",
21          "description": "A lakefront park on Chicago's south side"
22        },
23        "geometry": {
24          "coordinates": [
25            -87.603735,
26            41.829985
27          ],
28          "type": "Point"
29        }
30      },
31    ]
32  }
```

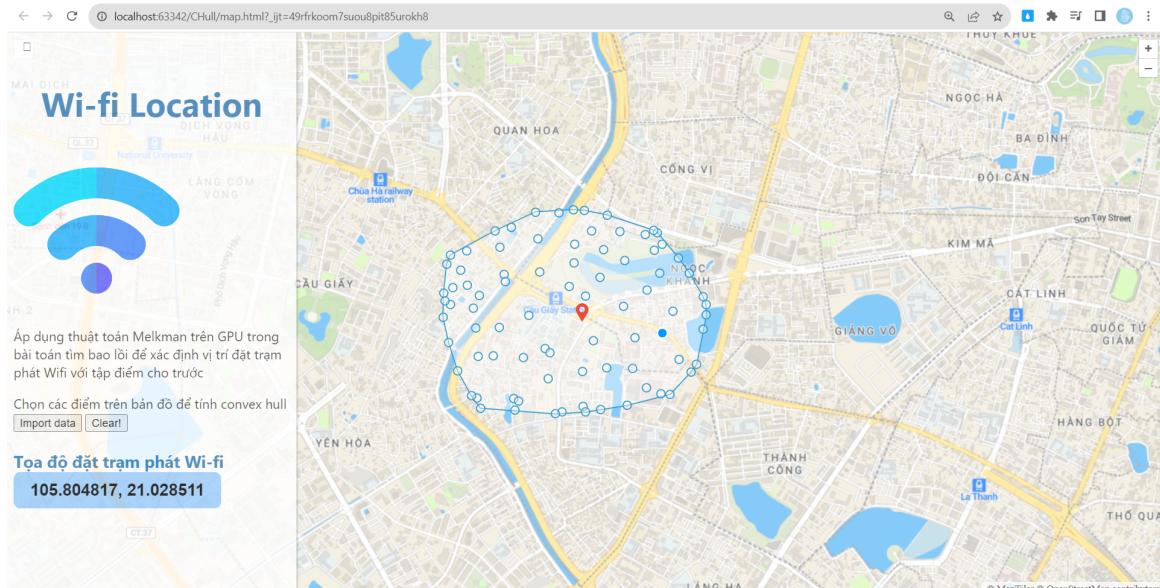
Hình 14: Ví dụ file chicago-parks.geojson

2. Clear

Khi muốn xóa tập điểm hiện tại để tìm một bao lồi cho tập hợp điểm mới, ta click vào nút "Clear!" để xóa toàn bộ các điểm đang có.

3. Wi-fi Location

Sau khi đã tính toán và hiện ra bao lồi từ một tập điểm, website sẽ trả về vị trí tâm của bao, hay chính là vị trí đặt trạm Wi-fi. Điểm này sẽ được đánh dấu bằng một định vị màu đỏ trên bản đồ



Hình 15: Tọa độ của trạm phát sóng wi-fi

6 Kết luận

Nhìn chung, bài báo cáo đã trình bày cách triển khai thuật toán CudaChain để giải quyết bài toán tìm bao lồi của một tập hợp điểm trên mặt phẳng cho trước. Trong quá trình triển khai, nhóm đã đề xuất phương pháp Sorting-based Preprocessing Approach (SPA) để loại bỏ các điểm nằm bên trong. Bên cạnh đó, nhóm cũng đã trình bày một thuật toán thực hiện trên GPU, được gọi là CudaChain, để tìm bao lồi của các tập điểm trên mặt phẳng thông qua việc sử dụng thuật toán SPA.

Nhóm cũng đã đánh giá hiệu suất triển khai của mình bằng cách so sánh kết quả thu được khi thực thi mã code có hoặc không sử dụng quy trình tiền xử lý (QuickHull). Từ những kết quả số liệu và biểu đồ so sánh như đã trình bày ở các phần trên, có thể dễ dàng nhận ra rằng việc triển khai thuật toán đề xuất của nhóm có thể đạt được tốc độ tăng lên tới 5-6 lần so với QuickHull trong trường hợp tốt nhất và 3-4 lần trong trường hợp trung bình.

Về mặt ứng dụng của thuật toán, nhóm cũng đã áp dụng thuật toán tìm bao lồi do nhóm đề xuất để xác định vị trí đặt trạm phát sóng thích hợp. Giao diện người dùng của ứng dụng cũng đã được triển khai và báo cáo cụ thể trong các phần trên.

Trong tương lai, nhóm hi vọng sẽ phát triển phương pháp để cải thiện nhiều hơn nữa trong việc tiền xử lý để cải thiện tốc độ tính toán cao hơn cũng như cải tiến được giao diện cho ứng dụng được hiệu quả hơn.

7 Lời cảm ơn

Trong thời gian học tập và hoàn thiện bài báo cáo này, nhóm chúng em xin chân thành cảm ơn sự giúp đỡ nhiệt tình của thầy Trần Hà Nguyên vì đã giúp đỡ, cung cấp nhiều thông tin quý báu và tạo điều kiện cho bọn em xuyên suốt quá trình học tập. Mặc dù nhóm đã cố gắng hoàn thiện bài báo cáo này, tuy nhiên do thời gian, kiến thức và kinh nghiệm có hạn nên bài làm của nhóm còn có nhiều thiếu sót trong việc trình bày, đánh giá và đề xuất ý kiến. Em rất mong nhận được sự thông cảm và đóng góp ý kiến của thầy và các bạn. Em xin chân thành cảm ơn.

Tài liệu

- [1] Zhang Jiayin et al. “A Novel Implementation of QuickHull Algorithm on the GPU”. In: *arxiv* (Jan. 2016).
- [2] Jiayu Qin et al. “CudaCHPre2D: A straightforward preprocessing approach for accelerating 2D convex hull computations on the GPU”. In: *Concurrency and Computation Practice and Experience* 32 (Apr. 2019). DOI: 10.1002/cpe.5229.
- [3] Vasyl Tereshchenko, Yaroslav Tereshchenko, and Dmytro Kotsur. “Point triangulation using Graham’s scan”. In: *Fifth International Conference on the Innovative Computing Technology (INTECH 2015)*. 2015, pp. 148–151. DOI: 10.1109/INTECH.2015.7173370.