MỘT NGHIÊN CỨU TRƯỜNG HỢP ĐẠI DIỆN KHOA HỌC ỨNG DỤNG (GeoAc) SỬ DỤNG KHUNG ĐA DIỆN THẲNG THẲNG THẲNG

qua

Ravi Shankar

Một luận án

nộp một phần để hoàn thành

của các yêu cầu về mức độ

Thạc sĩ Khoa học Máy tính

Đại học Boise State

Tháng 8 năm 2021

TRƯỜNG CAO ĐẲNG ĐẠI HỌC BANG BOISE

ỦY BAN QUỐC PHÒNG VÀ PHÊ DUYỆT ĐỌC CUỐI CÙNG

của luận án được nộp bởi

Ravi Shankar

Tiêu đề luận án: Một nghiên cứu trường hợp về việc thể hiện các ứng dụng khoa học (GeoAc) bằng cách sử dụng Khung đa diện thưa thớt

Ngày thi vấn đáp cuối kỳ: 24 tháng 5 năm 2021

Những cá nhân sau đây đã đọc và thảo luận về luận án do sinh viên Ravi Shankar nộp, và họ đã đánh giá bài thuyết trình và trả lời các câu hỏi trong kỳ thi vấn đáp cuối cùng. Họ thấy rằng sinh viên đã vượt qua kỳ thi vấn đáp cuối cùng.

Tiến sĩ Catherine Olschanowsky	Chủ tịch, Ủy ban giám sát
Dylan Mikesell, Tiến sĩ	Thành viên, Ủy ban giám sát
Tiến sĩ Steven Cutchin	Thành viên, Ủy ban giám sát

Sự chấp thuận đọc cuối cùng của luận án đã được chấp thuận bởi Catherine Olschanowsky, Tiến sĩ, Chủ tịch Ủy ban giám sát. Luận án đã được chấp thuận bởi Cao đẳng sau đại học.

Dành tặng cho gia đình tôi.

LỜI CẢM ƠN

Tác giả xin bày tỏ lòng biết ơn tới cố vấn của mình, Tiến sĩ Catherine Olschanowsky vì sự ủng hộ, động viên và niềm tin không ngừng nghỉ của cô dành cho anh. sự kiên nhẫn, lời khuyên và động lực trong nhiều năm qua là vô giá.

Tác giả xin cảm ơn các thành viên ủy ban giám sát của mình là Tiến sĩ Dylan Mikesell và Tiến sĩ Steven Cutchin đã phản hồi và hỗ trợ.

Tác giả cũng muốn ghi nhận sự giúp đỡ và hợp tác của hiện tại và các cựu thành viên của phòng thí nghiệm Tối ưu hóa luồng dữ liệu ADaPT đặc biệt là Aaron Orenstein, Anna Rift, Tobi Popoola và Tiến sĩ Eddie Davis.

Tài liệu này dựa trên công trình được hỗ trợ một phần bởi Khoa học Quốc gia Nền tảng theo Số tài trợ 1849463 và 1563818. Bất kỳ ý kiến, phát hiện và kết luận hoặc khuyến nghị được nêu trong tài liệu này là của tác giả và không nhất thiết phản ánh quan điểm của Quỹ Khoa học Quốc gia. Điều này nghiên cứu sử dụng hỗ trợ điện toán hiệu suất cao của cụm máy tính R2 (DOI: 10.18122/B2S41H) do Trung tâm nghiên cứu máy tính của Đại học Boise State cung cấp Phòng.

TÓM TẮT

Phát hiện và dự báo sóng thần là một vấn đề khó khăn mà các nhà khoa học đang cố gắng giải quyết. Ước tính đường đi sớm và dự đoán chính xác thời gian đến và kích thước của sóng thần có thể cứu sống và giúp đánh giá tác động. Sóng thần động đất gây ra sự dịch chuyển của mặt đất và bề mặt biển đẩy lên bầu khí quyển. Sự nhiễu loạn khí quyển này lan truyền lên trên như một âm thanh sóng và cuối cùng chạm tới tầng điện ly. IonoSeis là một gói phần mềm mô phỏng sử dụng các kỹ thuật cảm biến từ xa tầng điện ly dựa trên vệ tinh để xác định tâm chấn của những trận động đất này.

Thời gian thực hiện của thành phần dò tia của IonoSeis ngăn cản việc sử dụng nó như một công cụ mô hình hóa thời gian thực. Một giải pháp được đề xuất là thay thế thành phần này bằng một gói dò tia mới hơn được Phòng thí nghiệm quốc gia Los Alamos phát triển có tên là GeoAc và song song hóa nó. Nghiên cứu này là một nghiên cứu điển hình sử dụng đa diện thưa thớt khung (SPF) để biểu diễn mã GeoAc hoạt động và do đó thúc đẩy yêu cầu đối với khuôn khổ tối ưu hóa SPF đang được tích cực phát triển.

MỤC LỤC

DANH SÁCH CÁC BẢNG

DANH SÁCH CÁC HÌ NH ẢNH

DANH SÁCH CÁC TỪ VIẾT TẮT

CDL – Ngôn ngữ dữ liệu chung

GNSS – Hệ thống vệ tinh định vị toàn cầu

IEGenLib – Thư viện trình tạo thanh tra/thực thi

IR – Biểu diễn trung gian

LANL – Phòng thí nghiệm quốc gia Los Alamos

NetCDF – Biểu mẫu dữ liệu chung của mạng

NUMA – Truy cập bộ nhớ không đồng nhất

ODE – Phương trình vi phân thường

PDFG – Đồ thị luồng dữ liệu đa diện

SAC – Mã phân tích địa chấn

SPF – Khung đa diện thưa thớt

WASP3D – Truyền âm thanh trong khí quyển gió

DANH SÁCH CÁC BIỂU TƯỢNG

Logic và

| Ký hiệu lý thuyết tập hợp - sao cho

{} Ký hiệu lý thuyết tập hợp - Một tập hợp

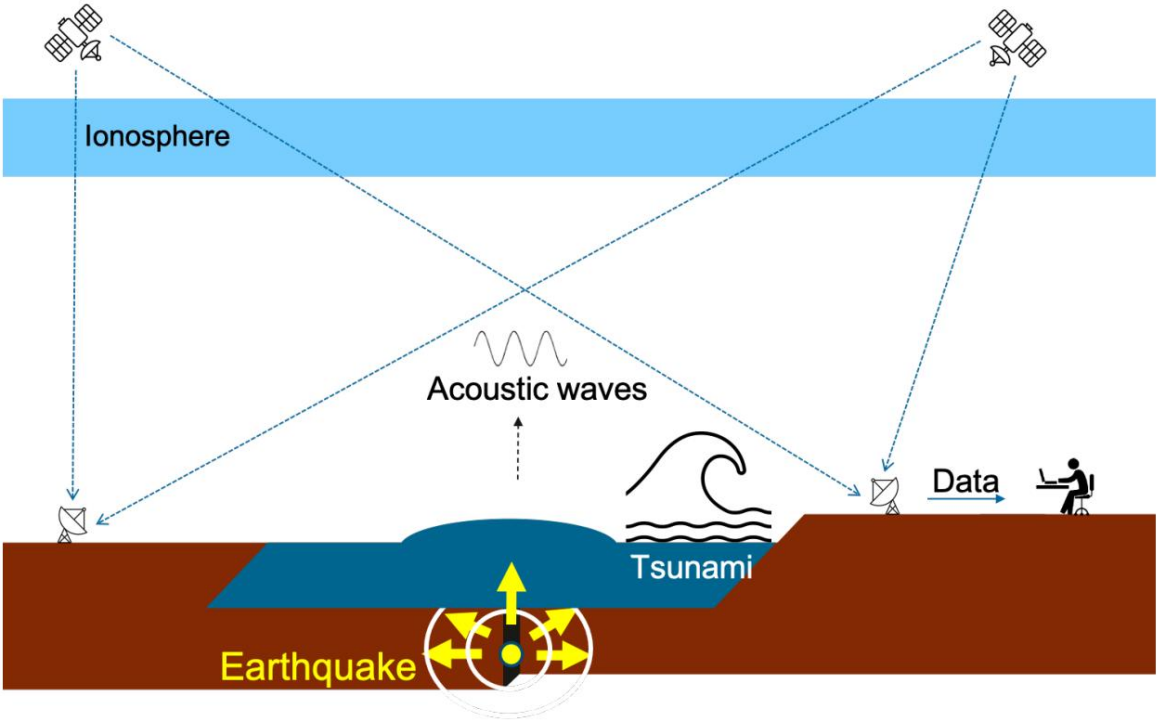
CHƯƠNG 1

GIỚI THIỆU

Phát hiện và dự báo sóng thần là một vấn đề khó khăn mà các nhà khoa học đang cố gắng giải quyết để giải quyết. Tính không thể đoán trước của sóng thần khiến cho việc cảnh báo sớm trở nên khó khăn và di tản các bờ biển có nguy cơ. Ước tính sớm và dự đoán chính xác thời gian đến và quy mô của sóng thần có thể cứu sống và giúp đánh giá tác động tòa nhà và cơ sở hạ tầng.

Phân tích dữ liệu cảm biến dựa trên vệ tinh có thể giúp dự đoán hướng đi của sóng thần. Động đất gây ra sóng thần gây ra sự dịch chuyển của mặt đất và bề mặt biển đẩy trên bầu khí quyển và lan truyền qua bầu khí quyển vào tầng điện ly. Hệ thống vệ tinh định vị toàn cầu (GNSS) giám sát các nhiễu loạn tầng điện ly trong được tạo ra bởi những hiện tượng như vậy. Các kỹ thuật cảm biến từ xa dựa trên vệ tinh như vậy có thể được sử dụng để ước tính sự biến dạng của bề mặt trái đất và dự đoán thời gian xảy ra sóng thần. IonoSeis [16] là một gói phần mềm mô phỏng tận dụng các kỹ thuật này để xác định tâm chấn của các trận động đất có thể gây ra sóng thần.

Có một số lĩnh vực chính mà IonoSeis có thể được cải thiện. Một trong những thành phần chính của nó nents là một gói dò tia ba chiều được gọi là Windy Atmospheric Sonic Sự lan truyền (WASP3D). WASP3D mô hình hóa sóng đến tầng điện ly từ tâm chấn của một trận động đất. Gói dò tia này hiện không đủ nhanh để đáp ứng nhu cầu của quy trình làm việc và ngăn cản ứng dụng giám sát thời gian thực. Có



Hình 1.1: Các kỹ thuật cảm biến từ xa dựa trên vệ tinh phát hiện các nhiễu loạn tầng điện ly do sóng thần gây ra.

cũng không có quy định nào để mô phỏng nhiều tâm chấn tiềm năng cùng một lúc. Một trong mục tiêu của luận án này thay thế thành phần này của IonoSeis bằng GeoAc [2] công cụ tạo mô hình.

GeoAc là một gói dò tia mới hơn được phát triển bởi Phòng thí nghiệm quốc gia Los Alamos tory (LANL) mô phỏng chính xác hơn hiện tượng lan truyền sóng so với WASP3D, mô hình dò tia hiện tại trong IonoSeis. Tuy nhiên, thời gian thực hiện của GeoAc cũng quá dài. Các ứng dụng khoa học như GeoAc thường bị giới hạn bởi bộ nhớ, yêu cầu một lượng lớn bộ nhớ và dành một lượng thời gian thực thi đáng kể đọc và ghi dữ liệu. Điều này có thể ảnh hưởng đáng kể đến hiệu suất. Bộ nhớ chia sẻ song song hóa đã được sử dụng trong GeoAc để đáp ứng nhu cầu của quy trình làm việc IonoSeis và có khả năng cho phép theo dõi nguy cơ sóng thần theo thời gian thực.

Một số khuôn khổ chuyển đổi đã được phát triển để tối ưu hóa bộ nhớ bị ràng buộc
ứng dụng. Tuy nhiên, chúng bị hạn chế về khả năng sử dụng và khả năng biểu đạt. Một
khuôn khổ như vậy là khuôn khổ đa diện thưa thớt. Các công cụ như SPF được sử dụng
để thể hiện các phép tính thưa thớt và áp dụng các phép biến đổi tự động. Trường hợp này
nghiên cứu thúc đẩy sự phát triển của một khuôn khổ tối ưu hóa SPF để cho phép sử dụng
của các chuyển đổi trình biên dịch để tối ưu hóa hiệu suất trở nên biểu cảm hơn và
do đó có thể sử dụng được nhiều hơn.

1.1 Phát biểu vấn đề

Thời gian thực hiện của thành phần dò tia của IonoSeis không đủ nhanh để
đáp ứng nhu cầu của quy trình làm việc và ngăn chặn việc sử dụng nó như một công cụ mô hình hóa thời gian thực.
đề xuất thay thế cho thành phần dò tia này bị ràng buộc bởi bộ nhớ và dành một
mất nhiều thời gian để đọc và ghi dữ liệu, do đó ảnh hưởng đến hiệu suất.

Các công cụ SPF thiếu khả năng biểu đạt để nắm bắt được hành vi thực sự của
ứng dụng khoa học.

1.2 Đóng góp

Luận án này trình bày một nghiên cứu điển hình thúc đẩy sự phát triển của một giải pháp tối ưu hóa SPF
Khung tối ưu hóa cho phép sử dụng trình biên dịch chuyển đổi
thông tin để tối ưu hóa hiệu suất trong các ứng dụng khoa học khác.

WASP3D, thành phần dò tia của IonoSeis được thay thế bằng OpenMP
việc triển khai công cụ dò tia âm thanh GeoAc của LANL. Công cụ đã được tùy chỉnh
để đáp ứng các yêu cầu của quy trình làm việc và tạo ra các tệp dữ liệu NetCDF cần thiết
bằng các bước tiếp theo trong IonoSeis.

Một nghiên cứu hiệu suất đã được tiến hành với các luồng khác nhau trên triển khai OpenMP Phân tích của GeoAc so sánh thời gian chạy và tốc độ với số lượng luồng khác nhau.

1.3 Tổ chức

Luận văn này được tổ chức thành sáu chương với số lượng mục khác nhau. Chương ter 2 cung cấp tổng quan về các công cụ phần mềm khoa học miền và bối cảnh thông tin về các khái niệm điện toán hiệu suất cao. Chương 3 trình bày chi tiết phần mềm nỗ lực kỹ thuật để song song hóa GeoAc bằng OpenMP và tùy chỉnh nó để đáp ứng nhu cầu của quy trình làm việc. Chương 4 đề cập đến những nỗ lực thể hiện GeoAc bằng cách sử dụng SPF và do đó thúc đẩy khuôn khổ tối ưu hóa. Chương 5 có nội dung đánh giá công trình liên quan. Chương 6 kết luận luận án này và cung cấp tóm tắt về công trình này.

CHƯƠNG 2

LÝ LỊCH

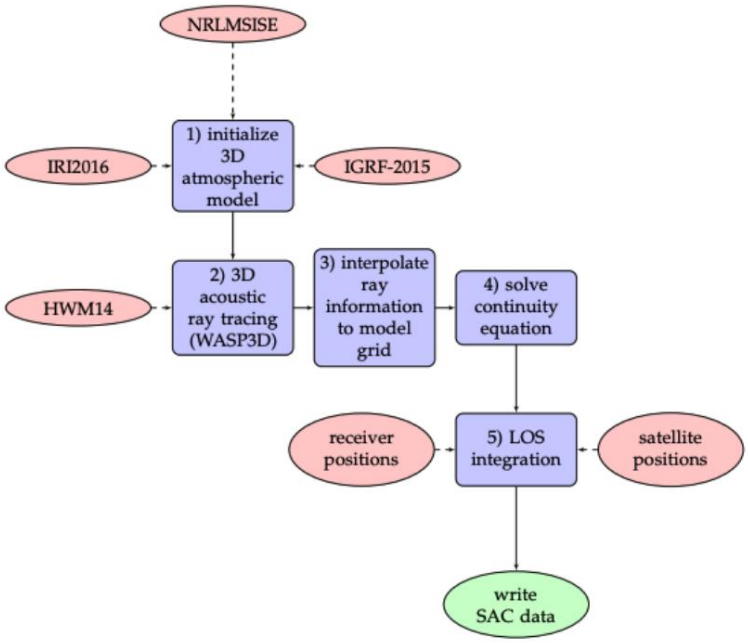
Chương này cung cấp tổng quan về các công cụ phần mềm khoa học miền
nghiên cứu này được xây dựng dựa trên thông tin cơ bản về máy tính hiệu suất cao
các khái niệm và khuôn khổ tối ưu hóa mà nghiên cứu điển hình này thúc đẩy. Phần 2.1
cung cấp tổng quan về gói phần mềm mô phỏng IonoSeis và chi tiết cách
thành phần dò tia WASP3D không đáp ứng được nhu cầu của quy trình làm việc và
ngăn cản ứng dụng giám sát thời gian thực. Phần 2.2 giới thiệu phần mềm GeoAc
gói được sử dụng để thay thế WASP3D và sau đó được tối ưu hóa. Phần 2.3 nói về
về bức tường trí nhớ và cách hiệu suất bộ nhớ là một hạn chế trong khoa học
tính toán. Phần 2.4 khám phá trình biên dịch đa diện và đa diện thưa thớt
khuôn khổ tối ưu hóa và giới thiệu API tính toán [21].

2.1 IonoSeis

Nhiều trận động đất gây ra sự dịch chuyển khối lượng đột ngột trên bề mặt trái đất. Khi
loại động đất này xảy ra dưới đại dương, có cường độ đủ mạnh,
và đáp ứng một số tiêu chí khác, sóng thần được tạo ra. Mặt đất hoặc mặt biển
sự dịch chuyển đẩy vào bầu khí quyển, từ đó tạo ra bầu khí quyển
nhiễu loạn. Sự nhiễu loạn này lan truyền lên trên như một sóng âm thanh cuối cùng
gây ra sự thay đổi cục bộ về mật độ electron của tầng điện ly. Điều hướng toàn cầu

Hệ thống vệ tinh (GNSS) giám sát các nhiễu loạn tầng điện ly do hiện tượng như vậy gây ra ena. Các phương pháp cảm biến từ xa dựa trên vệ tinh như vậy được sử dụng để ước tính trái đất biến dạng bề mặt và dự đoán thời gian xảy ra sóng thần.

IonoSeis là một gói phần mềm kết hợp nhiều cơ sở mã hiện có thành một gói đơn để mô hình hóa nhiễu loạn điện tử có nguồn gốc từ GNSS trong tầng điện ly do đến sự tương tác của bầu khí quyển trung tính và các hạt tích điện trong tầng điện ly. Chạy IonoSeis là một quá trình gồm 5 bước như được chỉ ra trong Hình 2.1. Phân tích địa chấn



Hình 2.1: Sơ đồ luồng cho khuôn khổ mô hình IonoSeis. Các hình bầu dục màu đỏ biểu thị đầu vào mô hình và các hộp màu xanh biểu thị các bước riêng lẻ, mỗi bước có tập lệnh BASH riêng với các tham số đầu vào cho bước đó [16].

Định dạng dữ liệu mã (SAC) được chỉ ra bởi hình bầu dục màu xanh lá cây ở bước 5 Hình 2.1 được sử dụng để lưu trữ chuỗi thời gian nhiễu loạn electron. Biểu mẫu dữ liệu chung của mạng (NetCDF) các tập tin được sử dụng để lưu trữ thông tin lưới giữa mỗi bước.

WASP3D là một trong những thành phần chính của IonoSeis và là một

phần mềm dò tia trong tọa độ hình cầu. Tuy nhiên, đây là một cơ sở mã cũ hơn và có một số hạn chế nhất định:

- Thời gian thực hiện không đủ nhanh để đáp ứng nhu cầu của IonoSeis và ngăn cản ứng dụng giám sát thời gian thực.
- Không có điều khoản nào cho phép mô phỏng nhiều tâm chấn tiềm tàng cùng một lúc. Hiện tại chỉ có thể mô phỏng được một tâm chấn duy nhất tại một thời điểm.
- WASP3D không thể hiện chính xác bản chất vật lý của quá trình truyền sóng.
  - Không tính đến sự tiêu tán năng lượng nhớt hoặc khuếch đại sóng do bầu khí quyển loãng ở độ cao lớn.
  - Nó không thể mô hình hóa nhiễu loạn tầng điện ly lớn hơn 500-600 km cho đến khi việc lấy mẫu khí quyển bằng cách bắn tia ở các khoảng cách không tuyến tính có thể được cải thiện.
  - Không mô hình hóa được hành vi dạng sóng phi tuyến tính.
  - Nó chỉ mô hình hóa trong tọa độ hình cầu. Mặc dù điều này hữu ích cho IonoSeis, các nhà khoa học muốn có thể mở rộng gói này để các phòng thí nghiệm quốc gia nghiên cứu về chất nổ và các nguồn âm thanh cục bộ khác có thể sử dụng Descartes phiên bản của mã.

Một đóng góp sơ bộ của luận án này thay thế WASP3D bằng GeoAc

2.2 Địa lý

GeoAc [2] là một công cụ phần mềm theo dõi tia được phát triển tại Phòng thí nghiệm quốc gia Los Alamos hùng biện. Phần mềm được viết bằng C++ và được sử dụng để mô hình hóa sự lan truyền của

sóng âm thanh qua khí quyển bằng cách sử dụng Runge–Kutta bậc bốn (RK4)
phương pháp. Mô hình GeoAc truyền bá trong mặt phẳng phương vị và bao gồm các phương pháp để
sử dụng hệ tọa độ Descartes cũng như hệ tọa độ hình cầu [2]. Một
của những đóng góp của luận án này là khám phá tính khả thi của việc sử dụng một tối ưu hóa
Phiên bản GeoAc để thực hiện dò tia tầng điện ly và có khả năng chạy theo thời gian thực.

Phân tích hiệu suất được mô tả trong Phần 3.1.1 chỉ ra rằng chức năng RK4
tion là hoạt động tốn kém nhất trong GeoAc và do đó được lựa chọn cho mục đích tiếp theo
phân tích và tối ưu hóa. Hình 3.2 cho thấy những điểm nghẽn tiềm ẩn về hiệu suất.
Phương pháp Runge–Kutta là một họ các giải pháp lặp thường được sử dụng để xấp xỉ
các giải pháp phù hợp cho các bài toán giá trị ban đầu của phương trình vi phân thường (ODE) [30,
1]. Phương pháp Runge-Kutta bậc bốn là phương pháp được sử dụng rộng rãi nhất để giải
ODE. Nó thường được sử dụng như một trình giải trong nhiều khuôn khổ và thư viện, cung cấp
sự cân bằng giữa chi phí tính toán và độ chính xác.

ODE là một phương trình vi phân xác định mối quan hệ giữa một yếu tố phụ thuộc
biến y và một biến độc lập x như thể hiện trong Phương trình 2.1. Thuật ngữ
'ordinary' chỉ ra rằng nó có đạo hàm thường chứ không phải đạo hàm riêng.

$$\frac{dy}{dx} + xy = 0 \tag{2.1}$$

ODE được sử dụng để mô hình hóa các vấn đề trong kỹ thuật và các ngành khoa học khác nhau liên quan đến
sự thay đổi của một số biến đối với một biến khác. Hầu hết các vấn đề này đòi hỏi
giải pháp của một bài toán giá trị ban đầu. Một bài toán giá trị ban đầu là một ODE cùng nhau
với một điều kiện ban đầu chỉ định giá trị của hàm chưa biết tại một số
điểm cho trước trong miền. GeoAc giải quyết các phương trình chi phối sự lan truyền âm thanh
qua bầu khí quyển trong giới hạn hình học bằng cách sử dụng thuật toán RK4.

2.3 Tối ưu hóa bộ nhớ

Bộ nhớ là một nút thắt quan trọng trong tính toán khoa học. Các ứng dụng khoa học là bị ràng buộc bởi bộ nhớ, thường đòi hỏi lượng lớn dữ liệu từ bộ nhớ. Do đó, phần lớn thời gian thực hiện của nó được dành cho việc đọc và ghi dữ liệu. Mặc dù vi xử lý tốc độ xử lý và bộ nhớ đang được cải thiện theo cấp số nhân, tốc độ cải thiện tốc độ bộ vi xử lý vượt xa tốc độ bộ nhớ, tạo ra cái được gọi là tường bộ nhớ [29]. Trong vài thập kỷ trở lại đây, tốc độ bộ xử lý đã tăng lên khoảng 80% mỗi năm [14], trái ngược với tốc độ bộ nhớ chỉ được cải thiện ở mức khoảng 7% mỗi năm [11]. Khoảng cách hiệu suất bộ xử lý-bộ nhớ ngày càng tăng này dẫn đến bộ xử lý dành một lượng thời gian đáng kể và do đó năng lượng, chờ đợi dữ liệu từ bộ nhớ. Kết quả là, hiệu suất của các ứng dụng khoa học mà bị ràng buộc bởi bộ nhớ bị ảnh hưởng đáng kể.

Tối ưu hóa bộ nhớ bao gồm một loạt các kỹ thuật được sử dụng để tối ưu hóa bộ nhớ cách sử dụng sau đó có thể được dùng để cải thiện hiệu suất của các ứng dụng khoa học. Vòng lặp là ứng cử viên hàng đầu cho việc tối ưu hóa vì chúng có thể cung cấp một nguồn tuyệt vời cho song song dữ liệu. Loop fusion là một kỹ thuật tối ưu hóa và chuyển đổi trình biên dịch nối hai hoặc nhiều câu lệnh từ các vòng lặp riêng biệt thành một vòng lặp duy nhất. Đọc-giảm sự hợp nhất liên quan đến việc hợp nhất các vòng lặp đọc từ cùng một vị trí bộ nhớ và cung cấp cơ hội để giảm số lần đọc cùng một dữ liệu. Điều này dẫn đến trong vị trí dữ liệu tốt hơn. Vị trí dữ liệu là thuộc tính nơi tham chiếu đến cùng một vị trí bộ nhớ hoặc các vị trí liền kề được sử dụng lại trong một khoảng thời gian ngắn [15]. Vị trí dữ liệu tốt hơn đảm bảo rằng dữ liệu được truy cập thường xuyên nhất có thể được truy xuất nhanh chóng. Sự kết hợp giữa nhà sản xuất và người tiêu dùng liên quan đến việc hợp nhất các vòng lặp trong đó một vòng lặp ghi một biến sau đó được vòng lặp thứ hai đọc do đó làm giảm dung lượng lưu trữ tạm thời

yêu cầu [6].

2.4 Khung tối ưu hóa

Phần này cung cấp tổng quan về trình biên dịch đa diện và đa diện thưa thớt
khuôn khổ tối ưu hóa và trình bày API tính toán, một công cụ tối ưu hóa SPF
khuôn khổ mà nghiên cứu điển hình này thúc đẩy.

2.4.1 Mô hình đa diện

Mô hình đa diện là một khuôn khổ toán học được sử dụng để biểu diễn và thao tác
các vòng lặp lồng nhau của các chương trình thực hiện các hoạt động lớn và phức tạp trong một chương trình nhỏ gọn
dạng. Các lần lặp lại của vòng lặp lồng nhau trong Hình 2.2 có thể được biểu diễn dưới dạng các điểm mạng
trên một đa diện như thể hiện trong Hình 2.3. Không gian lặp lại liên quan được biểu diễn
về mặt đồ họa như một không gian hai chiều (i,j) trong đó mỗi nút trong đồ thị biểu diễn
một sự lặp lại. Một không gian lặp lại mô tả một lồng vòng lặp được coi là một không gian affine
khi giới hạn dưới và trên của mỗi vòng lặp có thể được biểu thị dưới dạng hàm tuyến tính.

```
1 int phi_bounds = 3; 2
int theta_bounds = 3; 3
cho(int i = 1; i <= phi_bounds; i++){ 4 5 6 7 }
       đối với (int j = 1; j <= theta_bounds; j++){
            S1(i,j)
       }
```

Hình 2.2: Vòng lặp lồng nhau được đơn giản hóa từ cơ sở dữ liệu mã GeoAc.

Một vòng lặp lồng nhau có thể được biểu diễn bằng các thành phần sau:

- Miền lặp: Tập hợp các phiên bản được thực thi của mỗi câu lệnh trong vòng lặp.
  Các tập hợp này được biểu diễn bằng bất đẳng thức afin.

- Quan hệ: Tập hợp các lệnh đọc, ghi và có thể-ghi liên quan đến câu lệnh
  các trường hợp trong không gian lặp lại tới các vị trí dữ liệu.

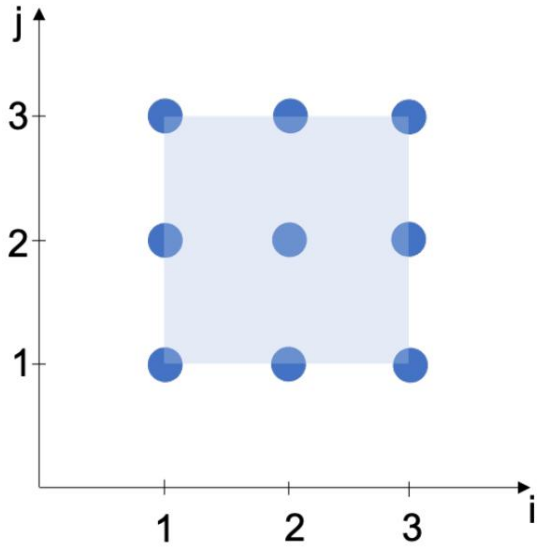
- Phụ thuộc: Tập hợp các phụ thuộc dữ liệu áp đặt các hạn chế đối với
  lệnh thi hành án.

- Lịch trình: Thứ tự thực hiện chung của mỗi trường hợp câu lệnh được biểu diễn
  bằng một tập hợp các cặp được sắp xếp theo thứ tự từ điển trong một không gian đa chiều [9].

Một phép biến đổi trong mô hình đa diện được biểu diễn bằng một tập hợp các hàm affine tions, một cho mỗi câu lệnh, được gọi là các hàm lập lịch. Một phép biến đổi affine là một phép ánh xạ tuyến tính bảo toàn các điểm, đường thẳng và mặt phẳng. Phép biến đổi afin có thể được áp dụng cho đa diện sau đó được chuyển đổi thành tương đương nhưng được tối ưu hóa mã. Khái niệm này cho phép trình biên dịch suy luận về biến đổi vòng lặp các tions như thao tác không gian lặp lại của các vòng lặp lồng nhau. Một hạn chế đối với đa diện mô hình là các ràng buộc trong không gian lặp lại và các phép biến đổi phải được afin.

2.4.2 Khung đa diện thưa thớt (SPF)

Khung đa diện thưa thớt mở rộng mô hình đa diện bằng cách hỗ trợ không gian lặp lại phi afin và biến đổi các phép tính bất thường bằng cách sử dụng uninter- các hàm được giả định [13]. Các hàm không được giải thích là các hằng số tượng trưng biểu diễn cấu trúc dữ liệu như mảng chỉ mục trong định dạng dữ liệu thưa thớt. Hằng số tượng trưng là những giá trị hằng số không thay đổi trong quá trình tính toán.
SPF biểu diễn các phép tính với các truy cập bộ nhớ gián tiếp và sắp xếp lại thời gian chạy các phép biến đổi với các tập hợp số nguyên và các mối quan hệ với các ràng buộc affine và con- các chủng liên quan đến các ký hiệu hàm chưa được giải thích [26, 27]. Sắp xếp lại dữ liệu thời gian chạy



Hình 2.3: Không gian lặp: các câu lệnh của vòng lặp lồng nhau trong Hình 2.2 có thể được xem như các điểm mạng trên một đa diện.

các kỹ thuật cố gắng cải thiện vị trí dữ liệu không gian và thời gian trong một vòng lặp bằng sắp xếp lại dữ liệu dựa trên thứ tự mà nó được tham chiếu trong vòng lặp [25].

Tập hợp và quan hệ trong SPF

Cốt lõi của SPF là các tập hợp và quan hệ. Không gian dữ liệu và lặp lại là được biểu diễn bằng các tập hợp, trong khi các hàm truy cập, sự phụ thuộc dữ liệu và các phép biến đổi được biểu diễn bằng các mối quan hệ [26].

Các tập hợp được chỉ định như thể hiện trong Phương trình 2.2, trong đó mỗi x là một bộ số nguyên và mỗi c là một ràng buộc. Độ lớn của tập hợp là m, là kích thước của các bộ.

$$s = \{[x1, \ldots, xm] \mid c1 \quad \ldots \quad cn\} \tag{2.2}$$

Các ràng buộc trong một tập hợp có thể là các đẳng thức hoặc bất đẳng thức là các biểu thức

chứa hằng số biểu tượng hoặc các hàm chưa được giải thích [26].

Mối quan hệ là một ánh xạ của các tập hợp số nguyên đầu vào tới các số nguyên đầu ra.
Các mối quan hệ được chỉ định như thể hiện trong Phương trình 2.3.

$$s = \{[x1, ..., xm] \quad [y1, ..., yn] \mid c1 \quad ... \quad cz\} \tag{2.3}$$

Mỗi x là một biến tuple đầu vào, mỗi y là một biến tuple đầu ra và c là một
sự hạn chế.

Phương trình 2.4 biểu diễn miền lặp của phép nhân vectơ ma trận dày đặc
mã hiển thị trong Hình 2.4. i và j là các trình lặp trong 2 chiều của tập hợp
và 0 <= i < N và 0 <= j < M là các bất đẳng thức afin hoặc ràng buộc ràng buộc
không gian lặp lại.

$$Tôi = \{[i, j] \mid 0 <= i < N \quad 0 <= j < M\} \tag{2.4}$$

2.4.3 API tính toán

Phần này mô tả API tính toán từ một bài báo hội thảo đã được
đã được chấp nhận [21] để xuất bản. Nghiên cứu trường hợp này đã đóng góp vào việc thiết kế và thử nghiệm
API tính toán. API tính toán là một API hướng đối tượng cung cấp
một thông số kỹ thuật chính xác về cách kết hợp các thành phần riêng lẻ của SPF để
tạo ra một biểu diễn trung gian (IR). IR là một cấu trúc dữ liệu được sử dụng nội bộ
bởi trình biên dịch để biểu diễn mã nguồn. IR này có thể trực tiếp tạo ra đa diện
đồ thị luồng dữ liệu (PDFG) [6] và dịch các hoạt động đồ thị được xác định cho PDFG
vào các mối quan hệ được sử dụng bởi Thư viện Trình tạo Thanh tra/Thực thi (IEGenLib) [26] để
thực hiện các phép biến đổi.

IEGenLib là một thư viện C++ với các cấu trúc dữ liệu và các chương trình con có thể biểu diễn, phân tích cú pháp và truy cập các tập hợp số nguyên và các mối quan hệ với các ràng buộc affine và uninterpreted ràng buộc về sự bình đẳng của biểu tượng hàm được diễn giải [26]. API tính toán được triển khai được coi là lớp C++ trong IEGenLib [26] và chứa tất cả các thành phần cần thiết để thể hiện một phép tính hoặc một loạt các phép tính. Ma trận vector dày đặc và thưa thớt phép nhân, được thể hiện lần lượt trong Hình 2.4 và Hình 2.5, được sử dụng làm ví dụ để biểu diễn các phép tính trong SPF. Các thành phần của một phép tính là: không gian dữ liệu, các câu lệnh, sự phụ thuộc dữ liệu và lịch trình thực hiện. Tiến trình các phần phụ mô tả thiết kế và giao diện cho từng thành phần này.

Nhân vectơ dày đặc

```
cho (i = 0; i < N; i++) {
    đối với (j=0; j<M; j++) {
        y[i] += A[i][j] * x[j];
    }
}
```

Hình 2.4: Phép nhân vectơ ma trận dày đặc.

CSR Ma trận thưa thớt vector nhân

```
cho (i = 0; i < N; i++) {
    đối với (k=rowptr[i]; k<rowptr[i+1]; k++) {
        j = cột[k];
        y[i] += A[k] * x[j];
    }
}
```

Hình 2.5: Phép nhân vectơ ma trận thưa.

Không gian dữ liệu

Không gian dữ liệu biểu thị một tập hợp các địa chỉ bộ nhớ duy nhất về mặt khái niệm. Mỗi sự kết hợp của tên không gian dữ liệu và bộ dữ liệu đầu vào được đảm bảo ánh xạ tới một không gian duy nhất trong bộ nhớ trong suốt thời gian tồn tại của không gian dữ liệu. Các không gian dữ liệu được biểu diễn trong các ví dụ nhân vectơ ma trận được thể hiện trong Hình 2.4 và 2.5 là y, A và x. Trong phiên bản thưa thớt, các mảng chỉ mục rowptr và col cũng được coi là dữ liệu khoảng trống.

```
1 // Ma trận vector dày đặc nhân
2 Tính toán* denseComp = Tính toán mới (); 3 denseComp-
>addDataSpace("y"); 4 denseComp-
>addDataSpace("A"); 5 denseComp-
>addDataSpace("x"); 6

7 // Phép nhân vector ma trận thưa
8 Tính toán* sparseComp = tính toán mới (); 9
sparseComp->addDataSpace("y");
10 sparseComp->addDataSpace("A");
11 sparseComp->addDataSpace("x");
```

Hình 2.6: Thiết lập không gian dữ liệu tính toán

Các tuyên bố

Các câu lệnh thực hiện các hoạt động đọc và ghi trên các không gian dữ liệu. Mỗi câu lệnh có một không gian lặp lại liên kết với nó. Không gian lặp lại này là một tập hợp chứa mọi trường hợp của câu lệnh và không có thứ tự cụ thể. Một câu lệnh được diễn đạt như tập hợp các trình lặp mà nó chạy qua, tuân theo các ràng buộc của vòng lặp của nó (vòng lặp giới hạn).

Một câu lệnh được viết dưới dạng chuỗi và tên của các không gian dữ liệu được phân định với ký hiệu $ (xem dòng 2 và 5 trong Hình 2.7). Không gian lặp được chỉ định là

thiết lập bằng cú pháp IEGenLib, ngoại trừ việc phân định tất cả các không gian dữ liệu bằng
biểu tượng $ (xem dòng 3 và 6 trong Hình 2.7).

```
1 Stmt* thickS0 = new Stmt( 2
"$y$(i) += $A$(i,j) * $x$(j);", 3
"{[i,j]: 0 <= i < N && 0 <= j < M}", ...
4
5 Stmt* sparseS0 = new Stmt( 6
"$y$(i) += $A$(k) * $x$(j)", 7
"{[i,k,j]: 0 <= i < N && rowptr(i) <= k < rowptr(i+1) && j = $col(k)}", ...
```

Hình 2.7: Đặc tả một phần của câu lệnh bằng cách sử dụng API tính toán.

Phụ thuộc dữ liệu

Sự phụ thuộc dữ liệu tồn tại giữa các câu lệnh. Chúng được mã hóa bằng cách sử dụng các mối quan hệ
giữa các vectơ lặp lại và các vectơ không gian dữ liệu. Trong vectơ ma trận nhân
Ví dụ, dữ liệu đọc và ghi được chỉ định như thể hiện trong Hình 2.8.

Lịch trình thực hiện

Lịch trình thực hiện được xác định bằng cách sử dụng các chức năng lập lịch được yêu cầu
để tôn trọng mối quan hệ phụ thuộc dữ liệu. Các hàm lập lịch lấy các trình lặp làm đầu vào
áp dụng cho câu lệnh hiện tại, nếu có, và đưa ra lịch trình dưới dạng một bộ số nguyên
có thể được sắp xếp theo thứ tự từ điển với những thứ khác để xác định việc thực hiện đúng
thứ tự của một nhóm các câu lệnh. Các trình lặp, nếu có, được sử dụng như một phần của đầu ra
tuple, chỉ ra rằng giá trị của các trình lặp ảnh hưởng đến thứ tự của câu lệnh. Đối với
ví dụ, trong hàm lập lịch {[i, j] [0, i, 0, j, 0]}, vị trí của i trước j
biểu thị rằng câu lệnh tương ứng nằm trong một vòng lặp trên j, mà lần lượt là
trong một vòng lặp trên i.

```
/* Tham số thứ 4 và thứ 5 cho hàm tạo Stmt */ // Nhân vectơ ma trận dày đặc
...
{ // đọc
{"y", "{[i,j]->[i]}"},
{"A", "{[i,j]->[i,j]}"}, {"x",
"{[i,j]->[j]}"} }, { // ghi


{"y", "{[i,j]->[i]}"} }
//
Phép nhân vectơ ma trận thưa ... { //
đọc
{"y", "{[i,k,j]-
>[i]}"}, {"A", "{[i,k,j]->[k]}"},
{"x", "{[i,k,j]->[j]}"} }, 
{ // ghi {"y", "{[i,k,j]->[i]}"}
}
```

Hình 2.8: Sự phụ thuộc dữ liệu giữa các câu lệnh.

Hình 2.10 cho thấy thông số kỹ thuật đầy đủ của phép nhân vectơ ma trận dày đặc theo sau là phép nhân vectơ ma trận thưa.

Tạo mã

Quét đa diện được sử dụng để tạo mã được tối ưu hóa. Lớp tính toán giao diện với CodeGen+ [4] để tạo mã. CodeGen+ sử dụng các bộ Omega và

```
/* Tham số thứ 3 cho hàm tạo Stmt */
// Ma trận vector dày đặc nhân "{[i,j]
->[0,i,0,j,0]}"

// Ma trận thưa vector nhân "{[i,k,j]-
>[0,i,0,k,0,j,0]}"
```

Hình 2.9: Lịch trình thực thi của một câu lệnh.

mối quan hệ để quét đa diện. Omega [12] là một tập hợp các lớp C++ để thao tác
các mối quan hệ và tập hợp số nguyên. Các tập hợp và quan hệ Omega có những hạn chế trong
sự hiện diện của các hàm không được diễn giải. Các hàm không được diễn giải bị giới hạn bởi tiền tố
quy tắc theo đó chúng phải là tiền tố của khai báo tuple. Các hàm không được diễn giải
không thể có biểu thức làm tham số. Việc tạo mã khắc phục hạn chế này
bằng cách sửa đổi các hàm chưa được biên dịch trong IEGenLib để tuân thủ Omega.

Hình 2.11 cho thấy kết quả của việc tạo mã cho vectơ ma trận thưa thớt
Phép tính nhân được định nghĩa trong Hình 2.10. Dòng 2 của Hình 2.11 định nghĩa một
macro cho câu lệnh s0, dòng 7 - 9 ánh xạ lại Omega tuân thủ không được giải thích
chức năng trở lại ban đầu của nó, và các dòng 11 - 16 là kết quả trực tiếp của quá trình quét đa diện
từ CodeGen+. Việc triển khai tính toán cung cấp tất cả các hỗ trợ
định nghĩa cho mã chức năng đầy đủ.

Thông số tính toán đầy đủ

```
// Ma trận vector dày đặc nhân
Tính toán* denseComp = Tính toán mới ();
denseComp->addDataSpace("y");
denseComp->addDataSpace("A");
denseComp->addDataSpace("x");
Stmt* denseS0 = new Stmt( // mã nguồn "$y$(i) += $A$(i,j) * $x$(j);", // miền lặp "{[i,j]: 0 <= i < $N$ && 0 <= j < $M$}", // hàm lập lịch "{[i,j] ->[0,i,0,j,0]}", { // dữ liệu đọc
    {"y", "{[i,j]->[i]}"},
    {"A", "{[i,j]->[i,j]}"},
    {"x", "{[i,j]->[j]}"}
},
  { // dữ liệu ghi
    {"y", "{[i,j]->[i]}"} }

); denseComp->addStmt(denseS0);

// Phép nhân vectơ ma trận thưa
Tính toán* sparseComp = Tính toán mới ();
sparseComp->addDataSpace("y");
sparseComp->addDataSpace("A");
sparseComp->addDataSpace("x");
Stmt* sparseS0 = new Stmt( "$y$(i) += $A$(k) * $x$(j)", "{[i,k,j]: 0 <= i < N && rowptr(i) <= k < rowptr(i+1) && j = col(k)}", "{[i,k,j]->[0,i,0,k,0,j,0]}", {
    {"y", "{[i,k,j]->[i]}"},
    {"A", "{[i,k,j]->[k]}"},
    {"x", "{[i,k,j]->[j]}"}
},
{
    {"y", "{[i,k,j]->[i]}"} }

); sparseComp->addStmt(sparseS0);
```

Hình 2.10: Đặc tả API tính toán cho phép nhân vectơ ma trận dày đặc và thưa thớt

Mã SPMV

```
#undef s0
#define s0(__x0, i, __x2, k, __x4, j, __x6) y(i) += A(k) * x(j)

#undef col(t0)
#undef col_0(__tv0, __tv1, __tv2, __tv3)
#undef rowptr(t0)
#undef rowptr_1(__tv0, __tv1)
#undef rowptr_2(__tv0, __tv1)
#define col(t0) col[t0]
#define col_0(__tv0, __tv1, __tv2, __tv3) col(__tv3)
#define rowptr(t0) rowptr[t0]
#define rowptr_1(__tv0, __tv1) rowptr(__tv1)
#define rowptr_2(__tv0, __tv1) rowptr(__tv1 + 1)

đối với (t2 = 0; t2 <= N-1; t2++) { đối với
(t4 = rowptr_1(t1,t2); t4 <= rowptr_2(t1,t2)-1; t4++) {
      t6=col_0(t1,t2,t3,t4);
      s0(0,t2,0,t4,0,t6,0);
    }
}

#undef s0
#undef cột(t0)
#undef cột_0(__tv0, __tv1, __tv2, __tv3)
#undef hàngptr(t0)
#undef hàngptr_1(__tv0, __tv1)
#undef hàngptr_2(__tv0, __tv1)
```

Hình 2.11: Ma trận thưa thớt vector nhân mã C được tạo ra bằng cách gọi codegen trên Computation

CHƯƠNG 3

CẬP NHẬT KỸ THUẬT CỦA GEOAC

Chương này cung cấp thông tin chi tiết về các bản cập nhật kỹ thuật phần mềm được thực hiện trên GeoAc. Các đóng góp được mô tả trong chương này bao gồm phần 3.1 trong đó ghi chép các bước khác nhau dẫn đến việc song song hóa bộ nhớ chia sẻ GeoAc, kết quả hiệu suất là kết quả của việc song song hóa trong phần 3.1.5 và phần 3.2, trong đó nêu chi tiết việc nâng cấp định dạng dữ liệu từ tệp dữ liệu chung lên chuẩn công nghiệp Định dạng NetCDF.

3.1 Song song hóa GeoAc

Các ứng dụng khoa học thường chạy trên các cụm/siêu máy tính lớn. Sức mạnh là một hạn chế thiết kế quan trọng của những máy này. Ứng dụng chạy càng lâu càng tiêu thụ nhiều điện năng. Một kết quả trực tiếp là nhu cầu làm mát tốn kém các giải pháp để giữ cho phần cứng chạy tối ưu. Các ứng dụng khoa học được thiết kế để tận dụng tính song song chạy nhanh hơn và do đó tiết kiệm thời gian, năng lượng và tiền bạc.

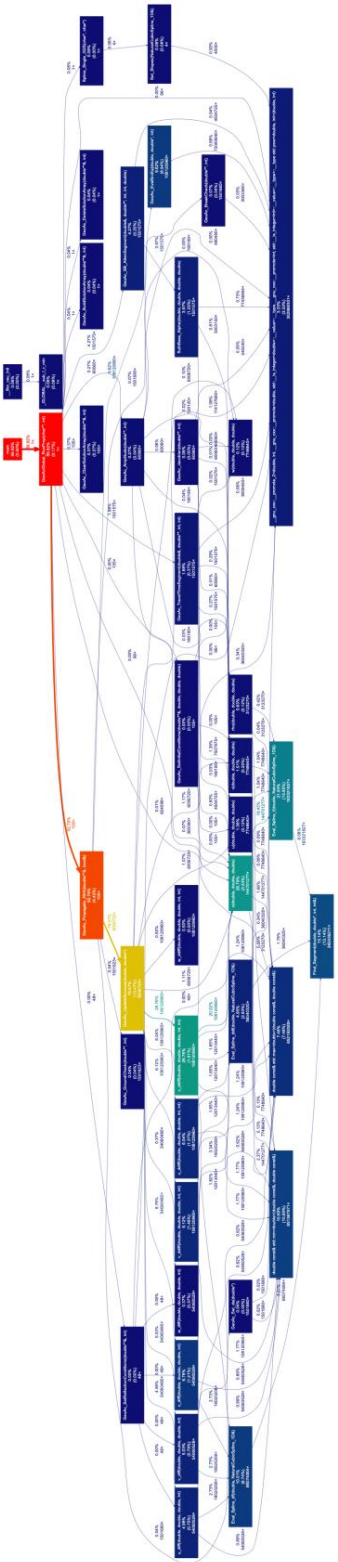
Mặc dù GeoAc đã cải thiện tính vật lý của các mô hình được tạo ra, nhưng nó vẫn chưa nhanh đủ để được sử dụng như một công cụ giám sát thời gian thực. Một lần chạy nối tiếp trên GeoAc với các thông số đầu vào như được chỉ định trong Hình 3.5 mất khoảng 19 giờ trên Boise

Cụm máy tính R2 của tiểu bang. Các phần sau đây mô tả nỗ lực thực hiện để
song song hóa GeoAc bằng OpenMP [5].

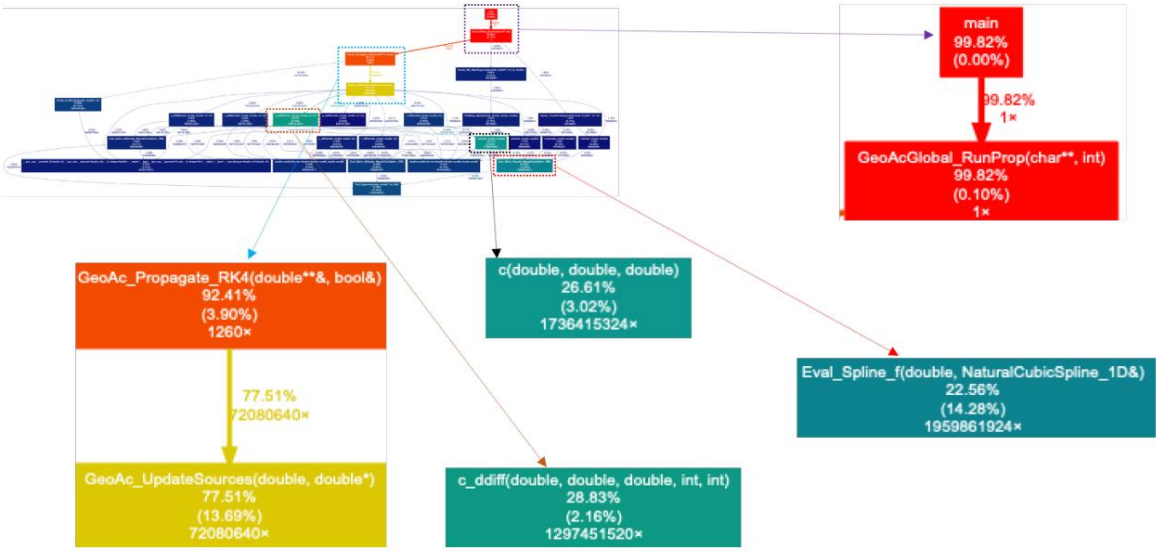
3.1.1 Hồ sơ hiệu suất

Việc lập hồ sơ GeoAc giúp xác định những phần nào của ứng dụng tốn kém,
và có thể là ứng cử viên tốt cho việc viết lại mã để có khả năng tạo ra ứng dụng
chạy song song. Các công cụ lập hồ sơ ghi lại phần thời gian chạy của chương trình được sử dụng
trong nhiều phần khác nhau của mã. Tiện ích GNU gprof [8] là một trong những công cụ như vậy
báo cáo dữ liệu này ở mức độ chi tiết cấp độ chức năng. GeoAc đã được lập hồ sơ bằng gprof để
tạo biểu đồ cuộc gọi được thấy trong Hình 3.1. Mỗi nút trong biểu đồ biểu diễn một
chức năng. Số đầu tiên là phần trăm thời gian tích lũy dành cho chức năng
hoặc các chương trình con của nó so với tổng thời gian chạy của ứng dụng. Số thứ hai,
được liệt kê trong dấu ngoặc đơn, là phần trăm thời gian dành trực tiếp cho chức năng.
số thứ ba là tổng số lệnh gọi đến hàm.

Một tập lệnh python nguồn mở gprof2dot [7] đã được sử dụng để chuyển đổi biểu đồ cuộc gọi thành
một biểu đồ chấm. Trình kết xuất chấm của Graphviz sau đó được sử dụng để tạo ra hình ảnh của chấm
đồ thị. Việc kiểm tra đồ thị chấm giúp chúng tôi có được cái nhìn tổng quan về các mối quan hệ
giữa các chức năng khác nhau trong GeoAc.



Từ biểu đồ chấm, chúng tôi có thể xác định rằng phần lớn thời gian được dành bởi ứng dụng nằm trong hàm GeoAcGlobal RunProp. Hàm này lần lượt được gọi là chức năng GeoAc Propagate RK4.



Hình 3.2: Các phần được phóng to của biểu đồ chấm của GeoAc xác định các điểm nghẽn hiệu suất tiềm ẩn

3.1.2 Xác định vòng lặp ứng viên

Việc lập hồ sơ GeoAc cũng giúp chúng tôi xác định các vòng lặp quan trọng có thể là ứng cử viên để tối ưu hóa. Một phân tích thủ công của mã đã tiết lộ tính hợp pháp của song song lelizing các phần tốn kém của mã. Hình 3.3 cho thấy một lồng vòng lặp bên trong Hàm GeoAcGlobal RunProp. Đây có vẻ là mục tiêu lý tưởng để tối ưu hóa.

3.1.3 Tái cấu trúc để chuẩn bị cho OpenMP

Một phân tích luồng dữ liệu thủ công đã được tiến hành để xác định các lần đọc và ghi trong các phần song song. Bất kỳ biến nào được ghi vào trong phần song song phải là luồng cục bộ. Tất cả các biến toàn cục không phải là hằng số đã được tái cấu trúc và làm cục bộ để

```
...
for(double phi = phi_min; phi <= phi_max; phi+=phi_step){
cho ( theta kép = theta_min; theta <= theta_max; theta+=theta_step){
       ...
       ...
       đối với (int bnc_cnt = 0; bnc_cnt <= số lần nảy; bnc_cnt++){
         k = GeoAc_Propagate_RK4(giải pháp, BreakCheck);
         nếu(WriteRays || WriteCaustics){
           nếu(WriteCaustics){
              ...
           }
           // ghi các cấu hình vào các tập tin dữ liệu và mảng vector
           đối với (int m = 1; m < k; m++){
              ...
           }
            ...
         }
          ...
       }
       ...
}
...
}
```

Hình 3.3: Vòng lặp ứng viên.

rằng mỗi luồng đều có một bản sao cục bộ của biến. Sau khi phân tích luồng dữ liệu, tất cả các biến toàn cục không hằng số được khai báo lại cục bộ và được thêm vào làm tham số các hàm mà các biến toàn cục đang được sử dụng. Trong một số hàm, các cấu trúc toàn cục đã được đang được sử dụng. Vì mỗi luồng yêu cầu một bản sao cục bộ của cấu trúc, nên nó đã được xác định lại và chuyển đến một tệp tiêu đề để các phiên bản cục bộ có thể được trình điều khiển gọi. Trong một hàm cụ thể, một cấu trúc mới có tên là SplineStruct đã được tạo ra như thể hiện trong Hình 3.4, chứa tất cả các biến toàn cục mà hàm sử dụng để thuận tiện truyền biến cần thiết xung quanh.

```
1 // Cấu trúc chứa các biến trước đó là toàn cục
2 // Mỗi luồng có bản sao riêng của nó 3 struct

SplineStruct{ 4 //---------------------------------------//
5 //------Các tham số cho Nội suy------//
6 //---------------------------------------// 7 int r_cnt; // Số điểm
thẳng đứng 8 int accel = 0; // Chỉ số gia tốc 9 double*
r_vals; // r_k phần tử (độ dài r_cnt) 10 double* T_vals; //
Nhiệt độ tại r_k 11 double* u_vals; // Gió EW tại r_k 12 double*
v_vals; // Gió NS tại r_k 13 double* rho_vals; // Mật độ tại
r_k 14 //-----------------------------------------------------//




15 ////-------------Hàm kết hợp để nhập-------------//
16 ////------Tệp G2S và tạo nội suy------//
17 ////-----------------------------------------------------// 18 struct NaturalCubicSpline_1D
Temp_Spline; 19 struct NaturalCubicSpline_1D Windu_Spline; 20
struct NaturalCubicSpline_1D Windv_Spline; 21 struct
NaturalCubicSpline_1D Density_Spline; 22 };
```

Hình 3.4: Kết hợp các biến toàn cục thành một cấu trúc.

3.1.4 Triển khai và thử nghiệm

Sau khi tái cấu trúc mã, OpenMP đã được sử dụng để đạt được bộ nhớ chia sẻ song song lelization. OpenMP bao gồm một tập hợp các chỉ thị biên dịch (#pragmas) và thư viện các thói quen cung cấp hỗ trợ cho lập trình song song trong môi trường bộ nhớ chia sẻ Các chỉ thị này được thiết kế sao cho ngay cả khi trình biên dịch không hỗ trợ chúng, chương trình vẫn sẽ thực thi, nhưng không có bất kỳ sự song song nào.

Chỉ thị '#pragma omp for' được áp dụng cho vòng lặp lồng nhau được xác định trong phần 3.1.2. Cấu trúc vòng lặp 'for' chỉ định rằng các lần lặp của vòng lặp sẽ là được phân phối giữa và được thực hiện bởi nhóm luồng gặp phải. Theo mặc định khi OpenMP gặp chỉ thị '#pragma omp for', nó sẽ tạo ra nhiều luồng vì có nhiều lõi trong hệ thống.

3.1.5 Hiệu suất

Tính chính xác được đảm bảo bằng cách chạy ứng dụng thành công bằng cách sử dụng các bước sau: Kiểm tra các thông số trong Hình 3.5 mà không có bất kỳ lỗi hoặc cảnh báo nào.

```
0 <= theta <= 90 ; theta_step=0,1
0 <= phi <= 90 ; phi_step=1
lần này = 0
rng_max=5000
```

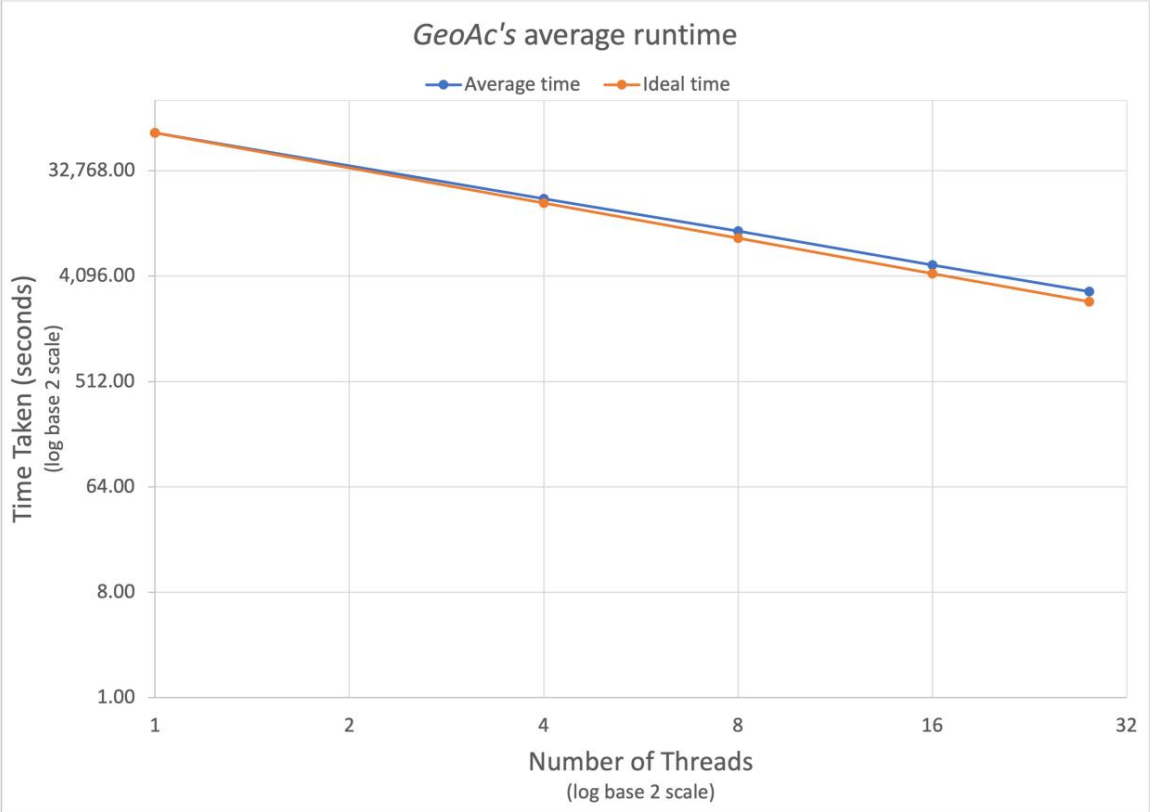
Hình 3.5: Các thông số thử nghiệm cho nghiên cứu hiệu suất của GeoAc.

Nhiều lần chạy ứng dụng được thực hiện với số lượng luồng khác nhau và một phân tích hiệu suất sơ bộ đã được tiến hành.

Bảng 3.1: Kết quả hiệu suất của GeoAc song song

Kết quả hiệu suất của GeoAc						
Luồng Chạy	1 (giây) Chạy	2 (giây) Chạy	3 (giây) Chạy	4 (giây) Trung bình (giây)		Tăng tốc
1	68552 69401	18642		69424	69105	
4		18924		19060	18835	3
8	9917	9909	9951	9961	9934	6
16	5078	5076	5087	5081	5080	13
28	3032	3016	3026	3023	3024	22

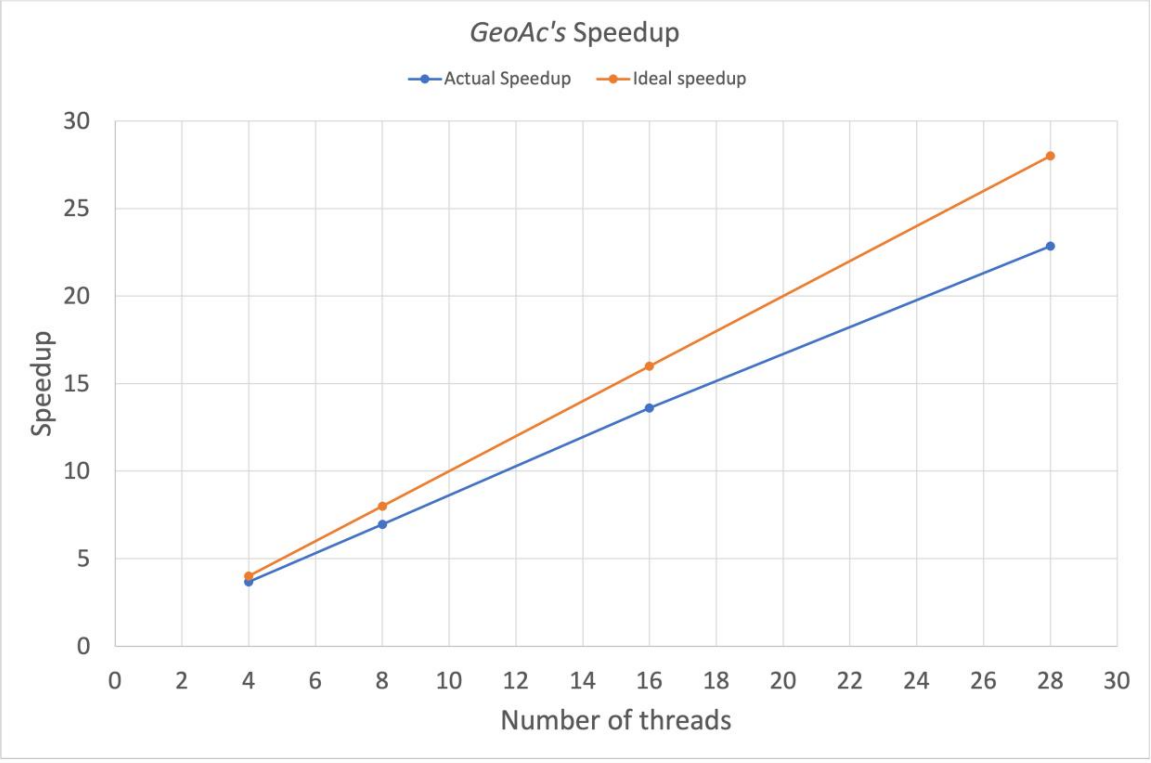
Hình 3.6 cho thấy thời gian chạy trung bình được đo qua bốn lần chạy với 1, 4, 8, 16 và 28 luồng. Đối với tập hợp các tham số đầu vào cụ thể này, 28 luồng là rõ ràng người chiến thắng với thời lượng chạy trung bình khoảng 50 phút so với 19 giờ với phiên bản tuần tự. Chạy 28 luồng dẫn đến tốc độ tăng 22 lần so với phiên bản tuần tự của mã như thể hiện trong Hình 3.7.



Hình 3.6: Thời gian chạy trung bình của GeoAc được đo qua bốn lần chạy với số luồng khác nhau. Thấp hơn là tốt hơn.

3.2 Nâng cấp định dạng dữ liệu theo tiêu chuẩn công nghiệp

GeoAc xuất dữ liệu trong các tệp DAT chung. Một trong những vấn đề với dữ liệu này loại là khi GeoAc được chạy với các bộ tham số lớn hơn, các tệp đầu ra có thể tăng lên lớn đáng kể. Một vấn đề khác đã được thảo luận trước đó trong phần 2.1 là Định dạng NetCDF được sử dụng để lưu trữ thông tin lưới giữa mỗi bước của IonoSeis. Theo yêu cầu của nhà khoa học chuyên ngành và vì lợi ích tuân thủ IonoSeis và các tiêu chuẩn công nghiệp, mã đã được sửa đổi để xuất ra các tệp NetCDF thay vì tập tin DAT chung.



Hình 3.7: Tốc độ tăng lên của GeoAc được đo qua bốn lần chạy với số lượng luồng khác nhau. Cao hơn thì tốt hơn.

3.2.1 Định dạng dữ liệu chuẩn mạng (NetCDF)

NetCDF là một dạng trừu tượng dữ liệu độc lập với máy thường được sử dụng để lưu trữ và truy xuất dữ liệu khoa học đa chiều [24]. Tệp NetCDF chứa dữ liệu kích thước sion, biến và thuộc tính. Các thành phần này được sử dụng cùng nhau để mô tả dữ liệu các trường trong một tập dữ liệu đa chiều. Một chiều có tên và kích thước và có thể được sử dụng để biểu diễn một chiều vật lý như số lượng tia và cực đại các điểm dọc theo một tia. Các biến có kiểu dữ liệu, tên và kích thước và lưu trữ phần lớn dữ liệu. Thuộc tính được sử dụng để cung cấp thêm thông tin về một biến hoặc tệp NetCDF nói chung. Hình 3.8 cho thấy cấu trúc của NetCDF tệp được tạo bằng GeoAc sử dụng ký hiệu CDL (Ngôn ngữ dữ liệu chung). CDL

là ký hiệu dễ đọc đối với dữ liệu NetCDF.

```
netcdf GeoAcResultsAndRaypaths {
chiều:
        nTia = 756 ;
        maxPointsAlongRay = 600 ;
      biến:
        theta đôi (nRays);
        góc cất cánh gấp đôi (n Tia);
        int n_b(nTia);
        lat_0_deg kép (nRays);
        đôi lon_0_deg(nRays);
        gấp đôi TravelTime_s(nRays);
        gấp đôi Celerity_kmps(nRays);
        đôi TurningHeight_km(nRays) ;
        độ nghiêng kép(nRays);
        đôi BackAzimuth_deg(nRays);
        đôi GeoAtten_dB(nRays);
        đôi AtmoAtten_dB(nRays);
        đôi wasp_altitude_km(nRays, maxPointsAlongRay) ;
        gấp đôi wasp_colatitude_deg(nRays, maxPointsAlongRay) ;
        đôi wasp_longitude_deg(nRays, maxPointsAlongRay) ;
        đôi wasp_kr_deg(nRays, maxPointsAlongRay) ;
        đôi wasp_kt_deg(nRays, maxPointsAlongRay) ;
        đôi wasp_kf_deg(nRays, maxPointsAlongRay) ;
        đôi wasp_amplitude_dB(nRays, maxPointsAlongRay) ;
        đôi wasp_Veff_dB(nRays, maxPointsAlongRay) ;
        gấp đôi wasp_arrival_time_s(nRays, maxPointsAlongRay) ;
}
```

Hình 3.8: Cấu trúc của tệp NetCDF kết quả từ GeoAc được biểu diễn bằng

NetCDF không an toàn cho luồng. Do đó, không thể song song hóa tệp i/o không sử dụng khóa hoặc một số loại loại trừ lẫn nhau có thể có tác dụng đáng kể tăng độ phức tạp của việc thực hiện. Việc thực hiện song song đã viết các tệp dữ liệu trên mỗi luồng và kết hợp chúng ở cuối phần song song. Thay vào đó trong việc kết hợp từng tệp dữ liệu của luồng, một lựa chọn tốt hơn là thu thập dữ liệu từ mỗi luồng và ghi trực tiếp vào tệp NetCDF.

CHƯƠNG 4

ĐẠI DIỆN CHO GEOAC SỬ DỤNG SPF

Dựa trên hồ sơ và phân tích luồng dữ liệu thủ công của mã, các phần của GeoAc có khả năng được thể hiện bằng cách sử dụng khuôn khổ đa diện thưa thớt đã được xác định. Chức năng GeoAc Propagate RK4 2 được coi là phù hợp nhất. Không gian dữ liệu và lặp lại được biểu diễn bằng các tập hợp trong khi truy cập các hàm, dữ liệu sự phụ thuộc và chuyển đổi được thể hiện bằng cách sử dụng các mối quan hệ. Chương này thảo luận về sự phát triển của khuôn khổ tối ưu hóa (API tính toán) bằng thể hiện mã GeoAc một cách gia tăng bằng cách sử dụng SPF và giải quyết các hạn chế khi chúng nảy sinh.

Từ biểu đồ chấm của GeoAc trong Hình 3.2, rõ ràng là GeoAc RunProp, Các hàm GeoAc Propogate RK4 và GeoAc UpdateSources có hiệu suất cao nhất phần trăm thời gian tích lũy dành cho chúng. Cốt lõi của việc dò tia âm thanh của GeoAc được thực hiện bởi thuật toán RK4, do đó GeoAc Propogate RK4 được coi là thích hợp nhất để đại diện bằng cách sử dụng SPF. Chiến lược đại diện cho op- mã hóa với SPF liên quan đến việc xác định hàm lá được gọi từ GeoAc Propogate chức năng RK4 và biểu diễn nó bằng cách sử dụng API tính toán. Chúng tôi tiến dần lên từ đó.

Để xác định chức năng lá, một phân tích luồng dữ liệu thủ công đã được tiến hành để xác định xác định tất cả các lệnh gọi hàm. Mỗi hàm được duyệt lặp đi lặp lại cho đến khi hàm lá

tion đã đạt được. Cuộc gọi đầu tiên trong GeoAc Propogate_RK4 là đến GeoAc UpdateSources
chức năng. Chức năng GeoAc UpdateSources gọi hàm c, sau đó hàm c lại gọi
hàm Eval Spline f. Sau đó, hàm Eval Spline f_gọi hàm Find Segment _
chức năng được xác định là chức năng lá từ lệnh gọi đến GeoAc Propogate RK4. _ _
Cần lưu ý rằng chức năng GeoAc Propogate RK4, GeoAc_UpdateSources _
và các chức năng còn lại đều nằm trong các đơn vị biên dịch khác nhau.

Sau khi phân tích luồng dữ liệu, chức năng Tìm phân đoạn được thể hiện bằng cách sử dụng khung đa diện thưa thớt. Điều này được thực hiện bằng cách sử dụng hàm nội tuyến với Compu-API phân tích được mô tả trong Phần 2.4.3.

4.1 Phát triển lặp đi lặp lại

Chuyển đổi lặp đi lặp lại là mục tiêu chính của API tính toán. Đóng góp
của luận án này đánh giá mức độ hiệu quả của các phép biến đổi lặp lại. Chúng tôi phân tích
quá trình biểu diễn SPF thành các phần lặp nhỏ được thử nghiệm với
mã GeoAc hoạt động. Chúng tôi bắt đầu bằng cách biểu diễn hàm lá Tìm đoạn như _
một hàm có thể tái sử dụng trả về một phép tính được nêu trong Phần 4.2. Mã
thế hệ được gọi trên Computation với trình điều khiển được phát triển trong IEGenLib. Điều này
kết quả trong một tệp C với các macro câu lệnh do trình biên dịch tạo ra. Tệp này được thả vào
mã GeoAc hoạt động thay cho chức năng Tìm phân đoạn như một bộ xử lý trước
chỉ thị (# include). Sau đó, GeoAc được chạy trên một trường hợp thử nghiệm với kết quả đã biết và
kết quả mới được so sánh để xác định tính chính xác.

4.2 Nội tuyến hàm

Các hàm hữu ích trong việc chia nhỏ các giải pháp lập trình thành các giải pháp có thể tái sử dụng nhỏ hơn các mô-đun và trợ giúp về cấu trúc mã. Tuy nhiên, chúng đặt ra những thách thức với phân tích thủ tục. Nội tuyến chức năng là một tối ưu hóa sao chép mã từ định nghĩa hàm trực tiếp vào mã của hàm gọi. Điều này loại bỏ nhu cầu để có một tập lệnh riêng biệt trong bộ nhớ và mở ra các cơ hội tối ưu hóa. Một trong những mục tiêu của API tính toán là loại bỏ phân tích liên thủ tục thử thách.

Các hàm c, v và u trong GeoAc UpdateSources mỗi hàm gọi hai hàm khác chức năng - Đánh giá Spline f và Tìm đoạn. Hơn nữa, vì hàm cha GeoAc UpdateSources nằm trong vòng lặp for, mỗi hàm này cuối cùng sẽ được gọi nhiều lần. Để cho phép tái sử dụng mã, các hàm trả về Computation* đã được tạo được lưu trong trình điều khiển tối ưu hóa trong đó mỗi chức năng trong trình điều khiển tương ứng với Biểu diễn SPF của hàm mà chúng tôi đã tối ưu hóa trong GeoAc. Tính toán* các hàm trả về chỉ được gọi một lần. Các lệnh gọi tiếp theo đến một hàm liên quan thêm cùng một phép tính nhiều lần. Hình 4.8 cho thấy hàm lá Tìm tính toán đoạn.

Các cuộc gọi hàm lồng nhau dẫn đến sự phát triển của appendComputation chức năng được hiển thị trong Hình 4.1. Chức năng này lấy tham số là Computation được thêm vào (appendee), miền lặp lại của ngữ cảnh appendee, lịch trình thực hiện của ngữ cảnh phụ lục và một vectơ các chuỗi chứa các đối số cho Computation được thêm vào. Giá trị trả về của hàm này là Cấu trúc AppendComputationResult bao gồm một vectơ các chuỗi chứa giá trị trả về (tên của một không gian dữ liệu hoặc một giá trị theo nghĩa đen) của hàm đang được thêm vào

và một số nguyên là độ sâu của câu lệnh cuối cùng trong phần phụ lục.

```
1 AppendComputationResult appendComputation( 2 const
Computation* khác, std::string bao
3           quanhIterDomainStr, std::string bao
4           quanhExecScheduleStr, const std::vector<std::string>&
5           arguments = {})
```

Hình 4.1: Hàm appendComputation.

Một trong những vấn đề với các hàm nội tuyến trên nhiều đơn vị biên dịch là tên va chạm giữa các hàm gọi và hàm được gọi. Hàm addParameter lấy tham số là một cặp chuỗi - tên của đối số và kiểu dữ liệu của đối số. Việc đổi tên không gian dữ liệu ngăn ngừa xung đột tên giữa người gọi và người được gọi cũng như nhiều lời gọi của người được gọi trong cùng một bối cảnh. Điều này được xử lý bởi appendComputation hàm ẩn bên dưới. Đối số được kiểm tra xem đó là một không gian dữ liệu trong trình gọi và một ngoại lệ sẽ được đưa ra nếu không phải vậy. Các giá trị theo nghĩa đen là ban đầu không được hỗ trợ. Người gọi được yêu cầu gán các giá trị theo nghĩa đen cho các biến tạm thời và sau đó truyền các biến đó vào như là các đối số. Nếu kiểm tra đối số cho biết rằng nó không phải là một không gian dữ liệu, nó được coi là một giá trị theo nghĩa đen và được đánh giá. Biểu thức trong các lệnh gọi hàm không được hỗ trợ. Người gọi sẽ cần gán biểu thức cho một không gian dữ liệu tạm thời.

Vì nội tuyến sao chép mã từ định nghĩa hàm trực tiếp vào lệnh gọi mã hàm, các câu lệnh return không hợp lệ. Hàm appendComputation hỗ trợ các giá trị trả về bằng cách trả về một cấu trúc bao gồm độ sâu của giá trị cuối cùng câu lệnh được thêm vào và một vectơ các chuỗi chứa giá trị trả về (tên của không gian dữ liệu hoặc theo nghĩa đen). Người gọi có trách nhiệm theo dõi xem kiểu trả về là một giá trị thực hoặc không gian dữ liệu.

Trường hợp sử dụng của chúng tôi đòi hỏi các lệnh gọi hàm lồng nhau trong các vòng lặp. Các hàm đang được gọi trong các vòng lặp của hàm gọi là một mô hình khá phổ biến trong khoa học ứng dụng. Trong tình huống này, quy trình nội tuyến phải điều chỉnh không gian lặp lại và lập lịch các chức năng của bên được gọi để xuất hiện trong các cấu trúc vòng lặp. Bằng cách sao chép dữ liệu từ các phép tính trước đó cho đến độ sâu nhất định của quá trình thực thi xung quanh lịch trình, quy trình nội tuyến có thể hỗ trợ các lệnh gọi hàm trong vòng lặp. Hình-ure 4.2 hiển thị một lệnh gọi hàm lồng nhau. Tham số thứ ba của appendComputation hàm trên dòng 13 chỉ ra rằng lệnh gọi đến aComputation là câu lệnh đầu tiên trong i loop (gọi hàm lồng nhau).

```
1 //Phép tính hiện tại (appender)
2 Tính toán* temp = tính toán mới (); 3

4 //Phép tính sẽ được thêm vào (phụ lục)
5 Tính toán* comp = aComputation(); 6

7 //Kiểu dữ liệu trả về của hàm appendComputation là một struct
8 AppendComputationResult aCompRes; 9
aCompRes = temp->appendComputation(comp, "{[0]}", "{[0]->[1, i, 1]}", args);
```

Hình 4.2: Gọi hàm lồng nhau bằng appendComputation.

4.3 Thoát vòng lặp sớm phụ thuộc vào dữ liệu

Chức năng Tìm đoạn được hiển thị trong Hình 4.3 được sử dụng để tìm chỉ mục đoạn cũng như đánh giá spline dọc 1D. Từ mã, và bằng cách kiểm tra kết quả của các câu lệnh in gỡ lỗi, người ta quan sát thấy rằng hàm ban đầu được viết bằng một tối ưu hóa vòng lặp. Bắt đầu, giữa và cuối của mảng x vals đã được kiểm tra tại bắt đầu của hàm để nhanh chóng tính toán chỉ số phân khúc mà không cần phải lặp lại toàn bộ mảng. Có nhiều lần thoát vòng lặp sớm phụ thuộc vào dữ liệu trong mã.

Chức năng Tìm đoạn ban đầu

```
//--------------------------------------------- //
//------Các hàm để tìm chỉ số đoạn------//
//-----và Đánh giá các Spline dọc 1D------//
//--------------------------------------------- //
int Find_Segment(double x, double* x_vals, int chiều dài, int & prev){
     nếu(x > x_vals[chiều dài-1] || x < x_vals[0]){
          cout << "Không thể nội suy ra ngoài giới hạn đã cho. x = không hợp lệ." <<       " << x <<     "
               '\n';
     } khác {

          // Kiểm tra chỉ mục trước đó và các đoạn giới hạn
          nếu(x >= x_vals[trước] && x <= x_vals[trước+1]){
               xong = đúng;
          }

          nếu(! xong && trước+2 <= chiều dài-1){
               nếu(x >= x_vals[trước+1] && x <= x_vals[trước+2]){
                    xong = đúng;
                    trước = trước + 1;
               }
          }
          nếu(! xong && trước-1 >= 0){
               nếu(x >= x_vals[prev-1] && x <= x_vals[prev]){
                    xong = đúng;
                    trước = trước - 1;
               }
          }

          nếu(!thực hiện){
               đối với (int i = 0; i < chiều dài; i++){
                    nếu(x >= x_vals[i] && x <= x_vals[i+1]){
                         chỉ số = i;
                         phá vỡ;
                    }
                    nếu (x >= x_vals[chiều dài - 2 - i] && x < x_vals[chiều dài - 1 - i])
                         {
                         chỉ số = (chiều dài - 2) - i;
                         phá vỡ;
                    }
               }
               trước = chỉ số;
          }
          trở về trước;
     }
}
```

Hình 4.3: Chức năng Tìm đoạn ban đầu trong G2S GlobalSpline1D.cpp đơn vị biên soạn

Các vòng lặp phụ thuộc dữ liệu có sự phụ thuộc vào dữ liệu, khiến việc song song hóa trở nên khó khăn
không cần tái cấu trúc mở rộng. Với chức năng hiện được nhúng, phụ thuộc vào dữ liệu sớm
lệnh exit sẽ khiến mã nhúng thoát sớm, dẫn đến lỗi không lường trước được.
Kết quả của phát hiện này là nghiên cứu đang được tiến hành để bổ sung các lối thoát sớm phụ thuộc vào dữ liệu
và luồng điều khiển phụ thuộc vào dữ liệu nói chung đến SPF. API tính toán thực hiện
hiện tại không hỗ trợ thoát sớm phụ thuộc vào dữ liệu và do đó, chức năng này là
được tái cấu trúc như thể hiện trong Hình 4.4.

Chức năng Tìm đoạn đã_sửa đổi

```
int Find_Segment(double x, double* x_vals, int chiều dài, int & prev){
    đối với (int i = 0; i < chiều dài; i++)
        { nếu (x >= x_vals[i] && x <= x_vals[i+1]){ trước
            = i;
        }
    }
}
```

Hình 4.4: Chức năng Tìm đoạn đã sửa đổi để xử lý các lần thoát sớm và
trả về các câu lệnh

Sau khi chức năng_Tìm đoạn được thể hiện trong SPF, chúng ta chuyển lên
tiếp theo là hàm Eval Spline f_. Chúng ta lặp lại quá trình này cho đến khi mã
trong hàm mà ban đầu chúng ta cố gắng biểu diễn trong SPF được thay thế bằng mã
được tạo ra bằng cách sử dụng API tính toán.

4.4 Viết lại mã bằng SPF

Phần này trình bày mã GeoAc hoạt động được biểu diễn trong SPF bằng cách sử dụng
API tính toán. API tính toán được sử dụng để biểu diễn Phân đoạn tìm kiếm đã sửa đổi _
chức năng trong Hình 4.5. Tham số đầu tiên là mã nguồn trong vòng lặp i và

được viết dưới dạng chuỗi trên dòng 1 và 2 của Hình 4.5. Tên của các không gian dữ liệu trong mã nguồn được phân cách bằng ký hiệu $.

```
1 Stmt* s0 = new Stmt("nếu($x$ >= $x_vals$[i] && 2 3 4
                                $x$ <= $x_vals$[i+1]) $prev$ = i;", //Câu lệnh
        "{[i]: i>=0 && i<length}", //Lịch trình lặp lại
      ...
```

Hình 4.5: Một câu lệnh trong Tính toán tìm đoạn.

Các câu lệnh có một không gian lặp lại liên kết với chúng. Miền lặp lại này là một tập hợp chứa mọi trường hợp của câu lệnh và thường được thể hiện dưới dạng tập hợp các trình lặp mà câu lệnh chạy qua, tuân theo các ràng buộc của vòng lặp của chúng giới hạn. Không gian lặp lại được biểu diễn trên dòng 3 của Hình 4.5 bằng cách sử dụng IEGenLib cú pháp ngoại trừ việc bao bọc các không gian dữ liệu bằng dấu $. Phương trình 4.1 biểu diễn không gian lặp của hàm Tìm đoạn. _

$$Tôi = \{[i] \mid i >= 0 \quad i < \text{chiều dài}\} \tag{4.1}$$

Như đã thảo luận trong Phần 2.4.2, các ràng buộc đối với không gian lặp lại nên được cung cấp như các hằng số tượng trưng. Một trường hợp thú vị mà chúng tôi gặp phải là khi các ràng buộc được truyền vào như là đối số cho hàm. Theo mặc định, thêm tham số vào Tính toán cũng thêm chúng như không gian dữ liệu vào Tính toán. Điều đó là cần thiết để xác định các biến không bao giờ được ghi vào và đảm bảo rằng chúng không bị đổi tên.

Sự phụ thuộc dữ liệu được mã hóa bằng cách sử dụng mối quan hệ giữa các vectơ lặp và dữ liệu vectơ không gian. Dữ liệu đọc và ghi có thể được chỉ định như thể hiện trong Hình 4.6:

Lịch trình thực hiện được xác định bằng cách sử dụng các hàm lập lịch. Chúng thực hiện lặp lại các tors áp dụng cho câu lệnh hiện tại làm đầu vào, nếu có, và đầu ra là lịch trình như một bộ số nguyên có thể được sắp xếp theo thứ tự từ điển với các bộ khác để xác định

```
{{"$x$","{[0]->[0]}"}, {"$x_vals$","{[i]->[i]}"}, {"$x_vals$","{[i]->[ip1]:
    ip1 = i+1}"}}, //Đọc 2
{{"$prev$","{[0]->[0]}"}} //ghi
```

Hình 4.6: Dữ liệu đọc và ghi của Phân đoạn Tìm kiếm.

thứ tự thực hiện đúng của một nhóm các câu lệnh. Hình 4.7 cho thấy thứ tự thực hiện lịch trình của chức năng Tìm đoạn.

```
//Tham số thứ 3 cho hàm tạo Stmt 2 ... 3 "{[i]->[0, i, 0]"},
...
```

Hình 4.7: Lịch trình thực hiện của tính toán tìm đoạn.

Khi chúng ta có tất cả các thành phần khác nhau của một phép tính, chúng ta có thể chỉ định Tính toán đầy đủ như Hình 4.8. Lớp Tính toán giao diện với CodeGen+ để tạo mã. Gọi codegen trên Computation sẽ tạo mã C dưới dạng của các lệnh macro và các lệnh gọi đến chúng. Macro là một đoạn mã có được đặt tên. Bất cứ khi nào tên macro được sử dụng, nó được thay thế bằng nội dung của macro. Hình 4.9 và 4.10 hiển thị kết quả của việc tạo mã.

Tính toán tìm đoạn hoàn chỉnh

```
Tính toán* Find_Segment_Computation(){

 Tính toán* FindSegmentComputation = new Computation();
 FindSegmentComputation->addParameter("$x$", "double");
 FindSegmentComputation->addParameter("$x_vals$", "double*");
 FindSegmentComputation->addParameter("$length$", "int");
 FindSegmentComputation->addParameter("$prev$", "int&");

 //Tạo s0
 //nếu(x >= x_vals[i] && x <= x_vals[i+1]) trước = i;
 Stmt* s0 = new Stmt("nếu($x$ >= $x_vals$[i] &&
        $x$ <= $x_vals$[i+1]) $prev$ = i;",
        "{[i]: i>=0 && i<length}", //Lịch trình lặp lại
        "{[i]->[0, i, 0]}", //Lịch trình thực hiện
        { //Đọc dữ liệu
          {"$x$","{[0]->[0]}"},
          {"$x_vals$","{[i]->[i]}"},
          {"$x_vals$","{[i]->[ip1]: ip1 = i+1}"}
        },
        { //Ghi dữ liệu
          {"$trước$","{[0]->[0]}"}}
        );

 //Thêm s0 vào phép tính
 FindSegmentComputation->addStmt(s0);

 //Thêm giá trị trả về
 FindSegmentComputation->addReturnValue("$prev$", true);

 trả về tính toán FindSegment;
 }
```

Hình 4.8: Chức năng tính toán tìm đoạn hoàn chỉnh

Macro do trình biên dịch tạo ra

```
1 ...
2 #định nghĩa s0(__x0) double _iegen_2r = r; 3 #định
nghĩa s1(__x0) double _iegen_2theta = theta; 4 #định nghĩa
s2(__x0) double _iegen_2phi = phi; 5 #định nghĩa s3(__x0)
NaturalCubicSpline_1D & _iegen_2Temp_Spline = spl.
        Temp_Spline; 6
#định nghĩa s4(__x0) double _iegen_2r_eval = min(_iegen_2r, r_max); 7 #định nghĩa
s5(__x0) _iegen_2r_eval = max(_iegen_2r_eval, r_min); 8 #định nghĩa s6(__x0) double
_iegen_2_iegen_1x = _iegen_2r_eval; 9 #định nghĩa s7(__x0) struct
NaturalCubicSpline_1D & _iegen_2_iegen_1Spline =
        _iegen_2Temp_Spline;
10 #định nghĩa s8(__x0) double _iegen_0_iegen_2_iegen_1x = _iegen_2_iegen_1x; 11 #định nghĩa
s9(__x0) double* _iegen_2_iegen_1_iegen_0x_vals =
        _iegen_2_iegen_1Spline.x_vals; 12 #xác
định s10(__x0) int _iegen_2_iegen_1_iegen_0length =
        _iegen_2_iegen_1Spline.length;
13 #define s11(__x0) int& _iegen_2_iegen_1_iegen_0prev = _iegen_2_iegen_1Spline
        .tăng tốc;
14 #định nghĩa s12( x0, i, x2) nếu( iegen 0 iegen 2 iegen 1x >=
        _iegen 2 iegen 1 iegen 0x vals[i] && iegen 0 iegen 2 iegen 1x <= iegen 2 iegen 1
        iegen 0x vals[i+1]) iegen 2 iegen 1 iegen 0prev = i; 15 #định nghĩa s13 ( __x0) int
_iegen_2_iegen_1k = _iegen_2_iegen_1_iegen_0prev; 16 #định nghĩa s14(__x0) double
_iegen_2_iegen_1result = 0.0; 17 #định nghĩa s15(__x0) if(_iegen_2_iegen_1k
< _iegen_2_iegen_1Spline.length) {
        double _iegen_2_iegen_1X = (_iegen_2_iegen_1x - _iegen_2_iegen_1Spline. x_vals[_iegen_2_iegen_1k])/
        (_iegen_2_iegen_1Spline.x_vals[ _iegen_2_iegen_1k+1] -
        _iegen_2_iegen_1Spline.x_vals[_iegen_2_iegen_1k]); double _iegen_2_iegen_1A =
         _iegen_2_iegen_1Spline.slopes[ _iegen_2_iegen_1k] *
        (_iegen_2_iegen_1Spline.x_vals[_iegen_2_iegen_1k+1] - _iegen_2_iegen_1Spline.x_vals[_iegen_2_iegen_1k])
        - ( _iegen_2_iegen_1Spline. f_vals[_iegen_2_iegen_1k+1] -
        _iegen_2_iegen_1Spline.f_vals[_iegen_2_iegen_1k]); nhân đôi
        _iegen_2_iegen_1B = -_iegen_2_iegen_1Spline.slopes[_iegen_2_iegen_1k+1] *
        (_iegen_2_iegen_1Spline.x_vals[_iegen_2_iegen_1k+1] - _iegen_2_iegen_1Spline.x_vals[_iegen_2_iegen_1k])
          + ( _iegen_2_iegen_1 Spline.f_vals[_iegen_2_iegen_1k+1] -
        _iegen_2_iegen_1Spline.f_vals[_iegen_2_iegen_1k]); _iegen_2_iegen_1result
        = (1.0 - _iegen_2_iegen_1X) *
        _iegen_2_iegen_1Spline.f_vals[ _iegen_2_iegen_1k] + _iegen_2_iegen_1X *
         _iegen_2_iegen_1Spline.f_vals[ _iegen_2_iegen_1k+1] + _iegen_2_iegen_1X * (1.0 -
        _iegen_2_iegen_1X) * ( _iegen_2_iegen_1A * (1.0 - _iegen_2_iegen_1X ) + _iegen_2_iegen_1B *
        _iegen_2_iegen_1X);}


18 #định nghĩa s16(__x0) double _iegen_2c_result = sqrt(gamR *
        _iegen_2_iegen_1result); 19
#định nghĩa s17(__x0, __x1, __x2) sources.c = _iegen_2c_result;
20
21 số nguyên không dấu t2 = 0;
22 ...
```

Hình 4.9: Mã được tạo cho một lệnh gọi hàm c từ hàm GeoAc UpdateSources dẫn đến phép tính được thể hiện dưới dạng macro. Macro s12 trên dòng 14 biểu diễn hàm Find Segment.

Macro đang sử dụng

```
 1 s0(0); 2
 s1(1); 3
 s2(2); 4
 s3(3); 5
 s4(4); 6
 s5(5); 7
 s6(6); 8
 s7(7); 9
 s8(8); 10
s9(9); 11
s10(10); 12
s11(11); 13 đối
với(t2 = 0; t2 <= iegen 2 iegen 1 iegen 0chiều dài-1;t2++) { s12(12,t2,0); } 14 s13(13); 15
        s14(14); 16 s15(15);


17 s16(16); 18
s17(17,0,0);




19 ...
```

Hình 4.10: Gọi đến các macro được định nghĩa trong Hình 4.9. Dòng 13 biểu diễn lệnh gọi đến hàm Tìm phân đoạn.

CHƯƠNG 5

CÔNG VIỆC LIÊN QUAN

5.1 Áp dụng tối ưu hóa đa diện vào ứng dụng khoa học

các hành động

Đồ thị luồng dữ liệu đa diện (PDFG) là một biểu diễn nội bộ của trình biên dịch phơi bày các cơ hội tối ưu hóa như chuyển đổi vòng lặp và lưu trữ tạm thời giảm tuổi. Để triển khai PDFG, khuôn khổ giới thiệu một ngôn ngữ đặc tả được gọi là Ngôn ngữ luồng dữ liệu đa diện (PDFL). Ngôn ngữ đặc tả này có thể được viết trực tiếp, bắt nguồn từ các mã hiện có hoặc được lấy từ một mã trung gian khác đại diện.

Các nghiên cứu hiện tại chứng minh lợi ích của việc tối ưu hóa luồng dữ liệu đa diện. Olschanowsky và cộng sự đã chứng minh lợi ích này trên động lực học chất lỏng tính toán (CFD) chuẩn mực [20]. Davis et al. đã tự động hóa các thí nghiệm từ trước làm việc bằng cách sử dụng đồ thị luồng dữ liệu vĩ mô đã sửa đổi [6]. Strout et al. mở rộng đa diện mô hình cho các phép tính thưa thớt cho phép truy cập mảng gián tiếp [26]. Nghiên cứu này được phân biệt bằng cách áp dụng cho một ứng dụng đầy đủ trong một lĩnh vực khác.

Các công cụ như Polly [10], Pluto [3], Loopy [19], PolyMage [18] và Halide [22, 23, 17] sử dụng mô hình đa diện để chuyển đổi các mã thông thường. PolyMage và Halide là hai ngôn ngữ và trình biên dịch dành riêng cho miền để tối ưu hóa tính song song, tính cục bộ và

tính toán lại trong các đường ống xử lý hình ảnh. Halide tách biệt định nghĩa của thuật toán từ những mối quan tâm về tối ưu hóa làm cho các thuật toán đơn giản hơn, nhiều hơn mô-đun và di động hơn. Điều này cho phép chúng ta chơi xung quanh với các tối ưu hóa khác nhau với sự đảm bảo không thay đổi kết quả. Chiến lược tối ưu hóa của PolyMage dựa vào chủ yếu là về khả năng chuyển đổi và tạo mã của đa diện khung biên dịch và thực hiện hợp nhất phức tạp, sắp xếp và tối ưu hóa lưu trữ tự động.

isl (thư viện tập hợp số nguyên) là một thư viện C an toàn luồng để thao tác các tập hợp và mối quan hệ của các bộ số nguyên bị giới hạn bởi các ràng buộc affine [28]. Thư viện này được sử dụng trong mô hình đa diện cho phân tích chương trình và chuyển đổi và hình thành cơ sở đối với các phép biến đổi afin được sử dụng trong tất cả các công cụ đã thảo luận trước đó trong phần này.

CHƯƠNG 6

PHẦN KẾT LUẬN

Luận án này trình bày một nghiên cứu điển hình để thể hiện việc dò tia âm thanh của GeoAc công cụ sử dụng khuôn khổ đa diện thưa thớt. GeoAc mô hình sóng chính xác hơn hiện tượng lan truyền so với công cụ dò tia hiện tại được sử dụng trong Gói mô hình hóa IonoSeis. GeoAc được tùy chỉnh để đáp ứng nhu cầu của IonoSeis quy trình làm việc. OpenMP đã được sử dụng để song song hóa GeoAc dẫn đến tốc độ tăng gấp 22 lần qua phiên bản tuần tự của mã. Thời gian chạy giảm từ khoảng 19 giờ xuống chỉ dưới 51 phút cho một tập hợp lớn các tham số đầu vào.

Mã GeoAc song song thúc đẩy sự phát triển của một khung tối ưu hóa SPF công việc được gọi là API tính toán. Các hàm lá của cơ sở mã GeoAc là được thể hiện gia tăng trong SPF bằng cách sử dụng API tính toán và mã được tạo ra được sử dụng như các macro câu lệnh. Các macro này được sử dụng để thay thế các hàm lá và ứng dụng đã được chạy trên một tập hợp các tham số thử nghiệm để xác định tính chính xác của kết quả. Mỗi biểu thức lặp lại của mã trong SPF đều đưa ra những thách thức mới giúp thúc đẩy sự phát triển của khuôn khổ tối ưu hóa.

TÀI LIỆU THAM KHẢO

[1] Martha L. Abell và James P. Braselton. 2 - phương trình vi phân thường bậc nhất. Trong Martha L. Abell và James P. Braselton, biên tập viên, Phương trình vi phân với Mathematica (Phiên bản thứ tư), trang 45–131. Nhà xuất bản Học thuật, Oxford, phiên bản thứ tư, 2016.

[2] P Blom. Geoac: Các công cụ số để mô hình hóa sự lan truyền âm thanh trong giới hạn hình học. https://github.com/LANL-Seismoacoustics/GeoAc, 2014.

[3] Uday Bondhugula, J. Ramanujam, và P. Sadayappan. PLuTo: Hệ thống tối ưu hóa chương trình đa diện thực tế và hoàn toàn tự động. PLDI 2008 - Hội nghị ACM SIGPLAN lần thứ 29 về Thiết kế và triển khai ngôn ngữ lập trình, trang 1–15, 2008.

[4] Chun Chen. Xem xét lại quét đa diện. Biên bản báo cáo Hội nghị ACM SIGPLAN về Thiết kế và Triển khai Ngôn ngữ Lập trình (PLDI), trang 499–508, 2012.

[5] Leonardo Dagum và Ramesh Menon. Openmp: một api tiêu chuẩn công nghiệp cho lập trình bộ nhớ chia sẻ. Khoa học và kỹ thuật tính toán IEEE, 5(1):46–55, 1998.

[6] Eddie C Davis, Michelle Mills Strout và Catherine Olschanowsky. Biến đổi chuỗi vòng lặp thông qua biểu đồ luồng dữ liệu vĩ mô. Trong Biên bản báo cáo Hội nghị chuyên đề quốc tế năm 2018 về Tạo và Tối ưu hóa mã, trang 265–277.
ACM, 2018.

[7] José Fonseca. Gprof2dot: Chuyển đổi đầu ra lập hồ sơ thành biểu đồ chấm.

[8] Susan L. Graham, Peter B. Kessler và Marshall K. McKusick. Gprof: Một trình lập hồ sơ thực hiện biểu đồ cuộc gọi. 39(4):49–57, tháng 4 năm 2004.

[9] Tobias Grosser, Sven Verdoolaege và Albert Cohen. Tạo ast đa diện không chỉ là quét đa diện. Giao dịch ACM về Ngôn ngữ lập trình và Hệ thống (TOPLAS), 37(4):12, 2015.

[10] Tobias Grosser, Hongbin Zheng, Raghesh Aloor, Andreas Simb¨urger, Armin Gr¨oßlinger và Louis-No¨el Pouchet. Polly - Tối ưu hóa đa diện trong LLVM. Biên bản Hội thảo quốc tế đầu tiên về Kỹ thuật biên soạn đa diện (IMPACT '11), trang Không có, 2011.

[11] John L Hennessy và David A Patterson. Kiến trúc máy tính: một định lượng cách tiếp cận. Elsevier, 2011.

[12] Wayne Kelly, Vadim Maslov, William Pugh, Evan Rosser, Tatiana Shpeisman và David Wonnacott. Hướng dẫn giao diện thư viện omega. 1995.

[13] Alan LaMielle và Michelle Mills Strout. Cho phép tạo mã trong khuôn khổ đa diện thưa thớt. Báo cáo kỹ thuật, Báo cáo kỹ thuật CS-10-102, 2010.

[14] John D McCalpin. Khảo sát về băng thông bộ nhớ và cân bằng máy trong các máy tính hiệu suất cao hiện nay. Bản tin IEEE TCCA, 19:25, 1995.

[15] Kathryn S. McKinley, Steve Carr và Chau-Wen Tseng. Cải thiện vị trí dữ liệu bằng các phép biến đổi vòng lặp. ACM Trans. Program. Lang. Syst., 18(4):424–453, tháng 7 năm 1996.

[16] Thomas Mikesell, Lucie Rolland, Rebekah Lee, Florian Zedek, Pierdavide Co¨ısson và Jean-Xavier Dessa. Ionoseis: Một gói để mô hình hóa các nhiễu loạn tầng điện ly đồng địa chấn. Atmosphere, 10(8):443, tháng 8 năm 2019.

[17] Ravi Teja Mullapudi, Andrew Adams, Dillon Sharlet, Jonathan Ragan-Kelley và Kayvon Fatahalian. Lên lịch tự động các đường ống xử lý hình ảnh halide. ACM Transactions on Graphics (TOG), 35(4):83, 2016.

[18] Ravi Teja Mullapudi, Vinay Vasista và Uday Bondhugula. Polymage: Tối ưu hóa tự động cho các đường ống xử lý hình ảnh. Trong ACM SIGARCH Computer Architecture News, tập 43, trang 429–443. ACM, 2015.

[19] Kedar S. Namjoshi và Nimit Singhania. Loopy: Các phép biến đổi vòng lặp có thể lập trình và được xác minh chính thức. Ghi chú bài giảng về Khoa học máy tính (bao gồm các tiểu mục Ghi chú bài giảng về Trí tuệ nhân tạo và Ghi chú bài giảng về Tin sinh học), 9837 LNCS (tháng 7): 383–402, 2016.

[20] Catherine Olschanowsky, Michelle Mills Strout, Stephen Guzik, John Loffeld và Jeffrey Hittinger. Một nghiên cứu về cân bằng song song, vị trí dữ liệu và tính toán lại trong các trình giải pde hiện có. Trong Biên bản báo cáo của Hội nghị quốc tế về tính toán hiệu suất cao, mạng, lưu trữ và phân tích, trang 793–804, 3 Park Ave, New York, NY, Hoa Kỳ, 2014. IEEE Press, IEEE Press.

[21] Tobi Popoola, Ravi Shankar, Anna Rift, Shivani Singh, Eddie Davis, Michelle Strout và Catherine Olschanowsky. Giao diện hướng đối tượng cho thư viện đa diện thưa thớt. Năm 2021, Hội thảo thứ chín về Mô hình thực thi luồng dữ liệu cho máy tính quy mô cực lớn, Đã chấp nhận.

[22] Jonathan Ragan-Kelley, Andrew Adams, Sylvain Paris, Marc Levoy, Saman Amarasinghe và Fr ´edo Durand. Tách các thuật toán khỏi lịch trình để tối ưu hóa dễ dàng các đường ống xử lý hình ảnh. 2012.

[23] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Fr´edo Durand và Saman Amarasinghe. Halide: ngôn ngữ và trình biên dịch để tối ưu hóa tính song song, tính cục bộ và tính toán lại trong các đường ống xử lý hình ảnh. Thông báo ACM SIGPLAN, 48(6):519–530, 2013.

[24] R. Rew và G. Davis. Netcdf: giao diện truy cập dữ liệu khoa học. IEEE Đồ họa máy tính và ứng dụng, 10(4):76–82, 1990.

[25] Michelle Mills Strout, Larry Carter và Jeanne Ferrante. Thành phần thời gian biên dịch của dữ liệu thời gian chạy và sắp xếp lại vòng lặp. SIGPLAN Not., 38(5):91–102, tháng 5 năm 2003.

[26] Michelle Mills Strout, Geri Georg và Catherine Olschanowsky. Thao tác tập hợp và quan hệ cho khuôn khổ đa diện thưa thớt. Trong Hội thảo quốc tế về ngôn ngữ và trình biên dịch cho tính toán song song, trang 61–75. Springer, 2012.

[27] Michelle Mills Strout, Alan LaMielle, Larry Carter, Jeanne Ferrante, Barbara Kreaseck và Catherine Olschanowsky. Một cách tiếp cận để tạo mã trong khuôn khổ đa diện thưa thớt. Máy tính song song, 53:32–57, 2016.

[28] Sven Verdoolaege. isl: Thư viện tập hợp số nguyên cho mô hình đa diện. Trong Đại hội quốc tế về phần mềm toán học, trang 299–302. Springer, 2010.

[29] Wm. A. Wulf và Sally A. McKee. Đụng vào bức tường ký ức: Ý nghĩa của điều hiển nhiên. SIGARCH Comput. Archit. News, 23(1):20–24, tháng 3 năm 1995.

[30] L. Zheng và X. Zhang. Chương 8 - phương pháp số. Trong Liancun Zheng và Xinxin Zhang, biên tập viên, Mô hình hóa và Phân tích các Vấn đề Chất lỏng Hiện đại, Toán học trong Khoa học và Kỹ thuật, trang 361–455. Nhà xuất bản Học thuật, 2017.

PHỤ LỤC A

THIẾT LẬP THỬ NGHIỆM

Công trình trong luận án này đã được phát triển và thử nghiệm trên R2 hiệu suất cao cụm tại Đại học Boise State. Mỗi nút trên cụm là một ổ cắm kép, Intel Xeon CPU E5-2680 v4 ở tần số xung nhịp 2,40 GHz. Mỗi nút bao gồm hai Non-Uniform Miền truy cập bộ nhớ (NUMA) mỗi miền chứa 14 lõi. Mỗi lõi có bộ nhớ cục bộ và mỗi nút được kết nối với nhau bằng phương tiện tốc độ cao các liên kết kết nối. Các lõi bao gồm bộ nhớ đệm L1 32KB, L2 256KB. Ngoài ra, mỗi lõi có quyền truy cập vào bộ nhớ đệm dữ liệu L3 35MB được chia sẻ trong mỗi miền NUMA. Hình A.1 hiển thị thông tin về kiến trúc CPU do lệnh lscpu cung cấp.

```
1 Kiến trúc: x86_64 2 Chế độ hoạt
động của CPU: 32 bit, 64 bit 3 Thứ tự Byte: Little
Endian 4 CPU: 28 5 Danh sách CPU trực tuyến:
0-27 6 Luồng trên
mỗi lõi: 1 7 Lõi trên mỗi socket: 14 8 Socket:
2 9 Nút NUMA: 2 10 ID nhà cung cấp:
GenuineIntel 11 Họ CPU: 6 12 Kiểu máy:
79 13 Tên kiểu máy: CPU
Intel(R) Xeon(R) E5-2680 v4
@ 2.40GHz 14 Bước: 1 15 MHz CPU: 1546.875
16 MHz tối đa của CPU:
3300.0000 17 MHz
tối thiểu của CPU: 1200.0000 18 BogoMIPS: 4800.30 19 Ảo hóa: Bộ nhớ đệm VT-x 20 L1d: 32K
Bộ nhớ đệm 21 L1i: 32K
Bộ nhớ đệm L2 22 : 256K
Bộ nhớ đệm L3 23 : 35840K
24 NUMA node0 CPU(s): 0,2,4,6,8,10,12,14,16,18,20,22,24,26
25 NUMA node1 CPU(s): 1,3,5,7,9,11,13,15,17,19,21,23,25,27
```

Hình A.1: Lệnh lscpu hiển thị thông tin kiến trúc CPU của R2.