CHUYỂN ĐỔI ĐỊNH DẠNG THẨM THẢO VÀ MÃ TỔNG HỢP

qua

Tobi Goodness Popoola

Một luận án

nộp một phần để hoàn thành

của các yêu cầu về mức độ

Tiến sĩ Triết học về Máy tính

Đại học Boise State

Tháng 5 năm 2023

TRƯỜNG CAO ĐẲNG ĐẠI HỌC BANG BOISE

ỦY BAN QUỐC PHÒNG VÀ PHÊ DUYỆT ĐỌC CUỐI CÙNG

của luận án được nộp bởi

Tobi Goodness Popoola

Tiêu đề luận án: Chuyển đổi định dạng thưa thớt và tổng hợp mã

Ngày thi vấn đáp cuối kỳ: 24 tháng 2 năm 2023

Những cá nhân sau đây đã đọc và thảo luận luận văn do sinh viên nộp
Tobi Goodness Popoola, và họ đã đánh giá bài thuyết trình và phản hồi các câu hỏi
trong kỳ thi vấn đáp cuối cùng. Họ thấy rằng học sinh đã vượt qua kỳ thi cuối cùng
kiểm tra miệng.

Catherine RM. Olschanowsky, Tiến sĩ	Chủ tịch, Ủy ban giám sát
Elena Sherman, Tiến sĩ	Thành viên, Ủy ban giám sát
Tiến sĩ Tim Andersen	Thành viên, Ủy ban giám sát

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

Dành tặng cho Bố và Mẹ

LỜI CẢM ƠN

Tác giả muốn bày tỏ lòng biết ơn tới cố vấn của mình, Catherine Olschanowsky, vì sự kiên nhẫn, động viên và hỗ trợ không ngừng của cô ấy.

Công trình này đã được hỗ trợ một phần bởi Quỹ Khoa học Quốc gia theo Số tài trợ 1422725, 1563818 và của Bộ Năng lượng Hoa Kỳ, Văn phòng Khoa học, Chương trình nghiên cứu máy tính khoa học tiên tiến theo Giải thưởng số DE-SC-04030 và số hợp đồng DE-AC02-05-CH11231. Bất kỳ ý kiến, phát hiện, và các kết luận hoặc khuyến nghị được nêu trong tài liệu này là của tác giả và không nhất thiết phản ánh quan điểm của Bộ Năng lượng.

TÓM TẮT

Tính toán thưa thớt rất quan trọng trong tính toán khoa học. Nhiều khoa học các ứng dụng tính toán trên dữ liệu thưa thớt. Dữ liệu được cho là thưa thớt nếu nó có tương đối số lượng nhỏ các số khác không. Các định dạng thưa thớt sử dụng các mảng phụ trợ để lưu trữ các số khác không, kết quả là, nội dung của các mảng phụ trợ không được biết cho đến thời gian chạy.

Mô hình Thanh tra/Thực thi (I/E) sử dụng thông tin thời gian chạy để tối ưu hóa trình biên dịch Thanh tra tính toán thông tin tại thời điểm chạy để điều khiển các chuyển đổi.

Trình thực thi—một chuyển đổi thời gian biên dịch của mã gốc—sử dụng thông tin được tính toán bởi thanh tra. Khung đa diện thưa thớt (SPF) bao gồm một loạt công cụ hỗ trợ chuyển đổi thời gian chạy I/E. Công việc này giới thiệu một khuôn khổ bao bọc các công cụ SPF trong khi cung cấp cái nhìn toàn diện về tính toán như một biểu diễn trung gian (IR). Công trình này cũng giới thiệu một phương pháp để tự động hóa tổng hợp các thanh tra viên để chuyển đổi giữa các định dạng thưa thớt và cải tiến SPF để khám phá hiệu suất của các ứng dụng không đều đặn.

MỤC LỤC

DANH SÁCH CÁC BẢNG

DANH SÁCH CÁC HÌNH ẢNH

DANH SÁCH CÁC TỪ VIẾT TẮT

ECP – Dự án điện toán Exascale

DOE – Bộ Năng lượng

RAW – Đọc Sau Khi Ghi

WAR – Viết Sau Khi Đọc

WAW – Viết Sau Khi Viết

RAR – Đọc Sau Khi Đọc

AST – Cây cú pháp trừu tượng

IR – Biểu diễn trung gian

COO – Định dạng tọa độ

MCOO – Định dạng tọa độ Morton

CSR – Hàng thưa nén

CSC – Cột thưa nén

CSF – Sợi thưa nén

ELL – Định dạng ELLPack

BSR – Hàng thưa nén khối

CSB – Khối thưa nén

HiCOO – Định dạng tọa độ phân cấp

SPF – Khung đa diện thưa thớt

PDFG – Đồ thị đa diện + luồng dữ liệu

SPMV – Phép nhân vectơ ma trận thưa thớt

API – Giao diện lập trình ứng dụng

DAG – Đồ thị có hướng phi chu trình

CFG – Biểu đồ luồng điều khiển

DFG – Biểu đồ luồng dữ liệu

SSA – Phân công đơn tĩnh

DSL – Ngôn ngữ dành riêng cho miền

PDFG – Đồ thị đa diện + luồng dữ liệu

PEP – Sự lan truyền biểu hiện đa diện

SCoP – Phần điều khiển tĩnh

CHƯƠNG 1

GIỚI THIỆU

Các ứng dụng tính toán chuyên sâu như nhiệt động lực học phân tử, khí hậu
Mô hình hóa bạn đời và máy học đòi hỏi nhiều tài nguyên tính toán.
Năng lượng tiêu thụ bởi các trung tâm dữ liệu lớn chạy các ứng dụng này có thể dễ dàng cung cấp năng lượng
lên các thành phố nhỏ. Cải thiện hiệu suất và hiệu quả của các ứng dụng này là
quan trọng. Hiệu suất được cải thiện có nghĩa là ít thời gian sử dụng tài nguyên máy tính hơn,
do đó dẫn đến việc tiêu thụ năng lượng tốt hơn. Cơ hội lớn nhất để cải thiện
chứng minh hiệu suất trong các ứng dụng khoa học là làm giảm sự di chuyển của bộ nhớ. Số học
Tính toán số liệu thường diễn ra nhanh và băng thông đến bộ nhớ là điểm nghẽn.
Bộ xử lý phải đợi để lấy dữ liệu từ bộ nhớ, điều này làm giảm hiệu suất.
bộ nhớ càng xa CPU thì càng cần nhiều chu kỳ tính toán hơn
để di chuyển dữ liệu trong quá trình tính toán. Công việc này nhằm mục đích giảm sự di chuyển bộ nhớ trong
các ứng dụng, đặc biệt tập trung vào các ứng dụng hoạt động trên dữ liệu thưa thớt. Chúng tôi đạt được
điều này thông qua việc giảm lưu trữ tạm thời, mở rộng bộ nhớ và ánh xạ lại, và
thuật toán đơn giản hóa đồ thị/phương pháp tìm kiếm để tự động chuyển đổi các phép tính
để sử dụng trí nhớ tốt hơn.

Việc tối ưu hóa các phép tính chạy trên dữ liệu thưa thớt rất phức tạp do gián tiếp
truy cập bộ nhớ. Những loại ứng dụng này được gọi là ứng dụng không thường xuyên.
Một tập dữ liệu được gọi là thưa thớt nếu nó có tỷ lệ phần trăm giá trị dữ liệu tương đối nhỏ

không phải là số không. Để tiết kiệm bộ nhớ, các định dạng thưa thớt chỉ lưu trữ các giá trị khác không và sử dụng mảng phụ trợ để xác định tọa độ gốc của các số khác không. Mảng phụ trợ mảng và thực tế là mẫu thưa thớt không được biết cho đến khi giới hạn thời gian chạy chuyển đổi tĩnh. Mô hình Thanh tra/Thực thi là một kỹ thuật phổ biến cho tối ưu hóa các ứng dụng không đều. Điều này đòi hỏi phải viết mã biên dịch sẽ tạo ra quyết định tại thời điểm chạy, Thanh tra và chuyển đổi phép tính ban đầu để sử dụng thanh tra, Người thực thi. Tuy nhiên, việc viết mã này bằng tay rất tẻ nhạt và dễ mắc lỗi.

Tối ưu hóa tự động các ứng dụng thông thường sử dụng các công cụ không áp dụng có thể áp dụng cho các ứng dụng không thường xuyên. Các công cụ này không hỗ trợ Inspector/Executor mô hình và các chuyển đổi sắp xếp lại thời gian chạy có liên quan khác. Nhiều công cụ như Omega/Codegen và IEGenLib cung cấp một khuôn khổ để chuyển đổi bất thường tuy nhiên, các ứng dụng cần phải cung cấp một điểm vào thống nhất cho việc này toolchain. Công việc này giới thiệu một khuôn khổ thống nhất bao gồm các công cụ thưa thớt có liên quan chuỗi trong khi cung cấp một cái nhìn toàn diện về tính toán. Khung này hỗ trợ chuyển đổi có thể cấu hình, biểu diễn và phân tích luồng dữ liệu và giao diện người dùng để tự động dịch các ứng dụng cũ sang một biểu diễn phù hợp để chuyển đổi sự hình thành.

Có một số định dạng được sử dụng để lưu trữ dữ liệu thưa thớt. Lựa chọn tối ưu của dữ liệu thưa thớt định dạng để tính toán phụ thuộc vào dữ liệu và phép tính. Quyết định định dạng thưa thớt tốt nhất cho tính toán không phải là trọng tâm của công trình này, mà nó tập trung vào cho phép chuyển đổi sau quyết định. Một công trình gần đây [29] cho thấy cách lựa chọn định dạng ma trận ảnh hưởng đến hiệu suất của Mạng nơ-ron đồ thị (GNN). Họ đã phát triển một mô hình dự đoán có thể chọn động định dạng thưa thớt tối ưu để sử dụng bởi một lớp GNN phụ thuộc vào các ma trận đầu vào. Điều này cho thấy không có một kích thước-

phù hợp với tất cả các lựa chọn định dạng thưa thớt, do đó cần phải chuyển đổi từ một định dạng sang một cái khác (Tổng hợp). Tổng hợp mã chuyển đổi định dạng có hiệu suất cao là tốt hơn là viết tay và tối ưu hóa mọi sự kết hợp có thể.

Lựa chọn tốt nhất của định dạng tenxơ thưa thớt thay đổi theo các mẫu tính toán và mẫu thưa thớt của dữ liệu. Sau khi lựa chọn định dạng đã được thực hiện, tối ưu hóa cần có các chương trình chuyển đổi mã thưa thớt từ định dạng này sang định dạng khác. Chuyển đổi định dạng thưa thớt có thể được thực hiện từ bất kỳ định dạng thưa thớt nào sang bất kỳ định dạng thưa thớt nào khác, tạo ra một không gian rộng lớn của sự biến đổi. Hơn nữa, sự lựa chọn đó có thể thay đổi thời gian sống của việc thực hiện ứng dụng. Như một ví dụ đơn giản, hãy xem xét một tenxơ thưa thớt được sử dụng trong nhiều giai đoạn tính toán và đôi khi sẽ được đọc trong giai đoạn đầu tiên chế độ và sau đó ở cuối. Thay đổi định dạng giữa các giai đoạn có thể có lợi tùy thuộc vào số lần các hoạt động được thực hiện. Chúng tôi trình bày tính biểu đạt của đặc tả với một tập hợp các định dạng tenxơ thưa thớt phổ biến và chúng tôi đánh giá tính đúng đắn của thuật toán tổng hợp. Công trình này trình bày một phương pháp mô tả các định dạng tenxơ thưa thớt và tổng hợp mã dịch trong số họ.

1.1 Đóng góp

Một số câu hỏi nghiên cứu vẫn cần được trả lời trước khi đạt được hiệu quả hoạt động thời gian hóa các ứng dụng khoa học sử dụng dữ liệu thưa thớt. Câu hỏi đầu tiên lý do về những gì cấu thành nên một biểu diễn nội bộ hoàn chỉnh cho các phép tính thưa thớt và cách thức biểu diễn nội bộ đó có thể được chuyển đổi bằng các phương pháp đã biết. thiếu thông tin cần thiết tại thời điểm biên dịch, câu hỏi nghiên cứu thứ hai là làm thế nào để chuyển đổi dữ liệu có thể được tự động hóa mà không phá vỡ sự trừu tượng trong

đại diện nội bộ. Công trình này giới thiệu hai đóng góp nghiên cứu để trả lời
những câu hỏi này.

1. Trình bày các phép biến đổi đa diện thưa thớt có thể cấu thành thông qua một đối tượng hướng
   API. Kết quả là một biểu diễn trung gian được chỉ định rõ ràng để mô tả và
   biến đổi các phép tính thưa thớt.

2. Tổng hợp các thanh tra để chuyển đổi định dạng thưa thớt và tối ưu hóa lặp lại đồng thời
   tions. Điều này giới thiệu một kỹ thuật tự động hóa để chuyển đổi bố cục dữ liệu
   của một phép tính thưa thớt.

1.2 Cấu trúc luận văn

Luận văn này được cấu trúc như sau. Chương 2 giới thiệu những điều cần thiết
nền tảng cần thiết để hiểu tài liệu này, đường ống biên dịch, đa diện
mô hình, mô hình đa diện thưa thớt, định dạng thưa thớt và mô hình thanh tra/thực thi.
Chương 3 giới thiệu biểu diễn trung gian SPF và các phương thức chuyển đổi được hỗ trợ của nó
hình thành. Chương 4 giới thiệu tổng hợp mã, một đóng góp chính của công trình này
nơi chúng tôi mô tả các bố cục dữ liệu thưa thớt và sử dụng các mối quan hệ ràng buộc trong việc chuyển đổi
từ một bố cục này sang một bố cục khác. Công việc liên quan đến tài liệu này được cung cấp trong Chương
Chương 5 và Chương 6 bao gồm phần kết luận và hướng nghiên cứu tiếp theo.

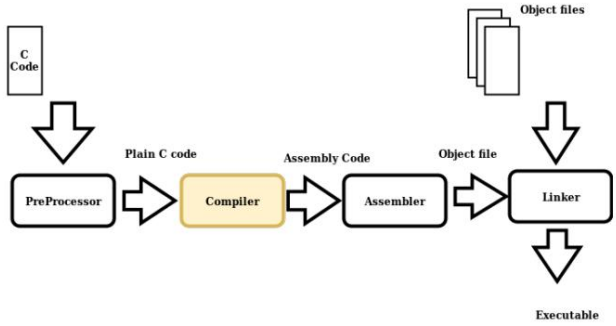
CHƯƠNG 2

LÝ LỊCH

Trong phần này, chúng tôi thảo luận về các khái niệm tạo thành cơ sở cho công việc của chúng tôi. Trình biên dịch đường ống, mô hình đa diện, mô hình đa diện thưa thớt và các định dạng thưa thớt là chìa khóa các khái niệm được sử dụng trong nghiên cứu của chúng tôi.

2.1 Đường ống biên dịch

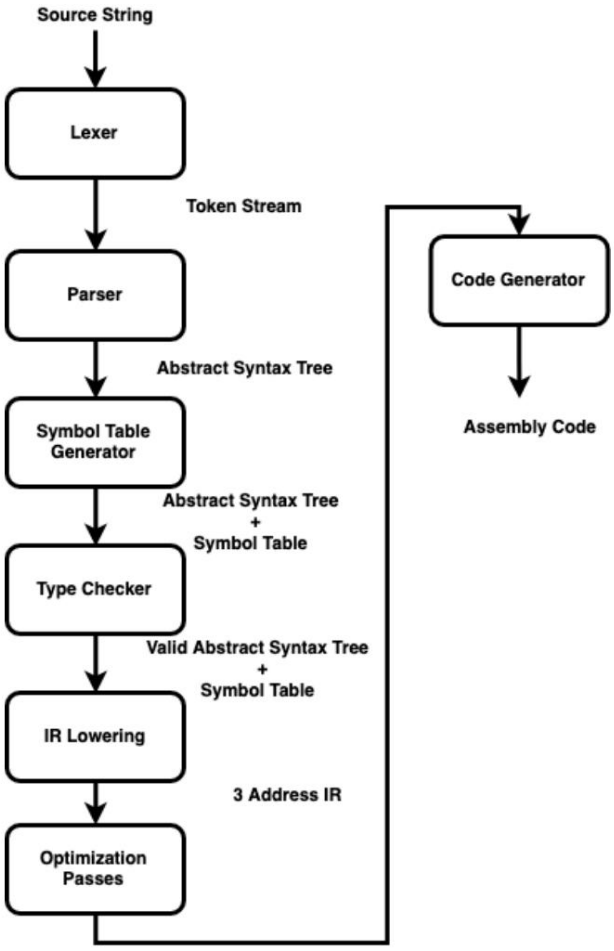
Một đường ống biên dịch điển hình bao gồm bốn giai đoạn: Bộ tiền xử lý, Bộ biên dịch, Assembler và Linker như thể hiện trong Hình 2.1



Hình 2.1: Đường ống biên dịch

Bộ tiền xử lý đọc một ngôn ngữ cấp cao chứa các chỉ thị có thể có, xử lý các chỉ thị và tạo ra mã nguồn. Trình biên dịch tạo ra các mã lắp ráp từ mã lập trình cấp cao. Trình lắp ráp tạo ra một tệp đối tượng từ mã lắp ráp cấp thấp. Cuối cùng, trình liên kết kết hợp đối tượng

tập tin với các tập tin đối tượng khác và các thư viện thời gian chạy để tạo ra một tệp thực thi nhị phân. Chúng tôi tập trung vào các chuyển đổi được áp dụng trong giai đoạn biên dịch



Hình 2.2: Các giai đoạn biên dịch

Việc biên dịch được chia thành nhiều giai đoạn khác nhau, giai đoạn đầu tiên là mã hóa nguồn văn bản thành từ điển, giai đoạn này rất quan trọng vì nó trích xuất các phần quan trọng của nguồn mã được gửi đến trình phân tích cú pháp. Trình phân tích cú pháp là giai đoạn thứ hai của đường ống biên dịch, trong giai đoạn này, Cây cú pháp trừu tượng (AST) của chương trình được xây dựng từ luồng của các mã thông báo sử dụng ngữ pháp không ngữ cảnh. Ngữ pháp xác định cú pháp của cấp độ cao

ngôn ngữ lập trình. Cây cú pháp trừu tượng cho thấy cấu trúc cú pháp của
văn bản ngôn ngữ lập trình.

Giai đoạn tiếp theo bao gồm việc xây dựng bảng ký hiệu, đây là một cấu trúc dữ liệu
giữ thông tin về các định danh và hàm (chữ ký tham số chính thức, trả về
loại và các thông tin khác về chức năng). Kiểm tra loại liên quan đến trình biên dịch
kiểm tra xem chương trình do người dùng viết có tuân thủ các quy tắc gõ do
ngôn ngữ cấp cao, bảng ký hiệu đóng vai trò quan trọng trong việc kiểm tra kiểu vì
trình biên dịch có thể truy cập thông tin quan trọng trong quá trình này.

Trình biên dịch tối ưu hóa thường có một biểu diễn trung gian được tạo ra
được tính từ AST của chương trình. Các biểu diễn trung gian hoặc IR có thể
phơi bày nhiều cơ hội tối ưu hóa khác nhau. Clang /LLVM sử dụng LLVM-IR như một
biểu diễn trung gian và GCC sử dụng Gimple. LLVM-IR [33] sử dụng RISC-like
định dạng hướng dẫn (còn được gọi là mã ba địa chỉ) của ngôn ngữ để cho phép chung
tối ưu hóa cấp thấp và thông tin gõ cho phép tối ưu hóa cấp cao.
LLVM-IR [33] và Gimple [1] sử dụng chỉ định tĩnh đơn (SSA), nghĩa là
mỗi biến chỉ được ghi một lần và có thể được đọc nhiều lần trong chương trình.
AST và các IR cấp thấp khác đã thảo luận trước đó cung cấp khả năng tối ưu hóa khác nhau
những cơ hội.

Việc tạo mã lắp ráp cho các máy mục tiêu cho biết kết thúc quá trình biên dịch
đường ống.

2.2 Mô hình đa diện

Mô hình đa diện là một công cụ tối ưu hóa hiệu quả cho các ứng dụng có affine
giới hạn vòng lặp. Mô hình này biểu diễn các phép tính dưới dạng tập hợp và quan hệ. Lặp lại

không gian được biểu diễn bằng các tập hợp và sự phụ thuộc dữ liệu được biểu diễn bằng các mối quan hệ. Sự kết hợp của không gian lặp lại và sự phụ thuộc dữ liệu này cung cấp một thứ tự một phần cho mục tiêu tính toán. Trong thứ tự một phần, thứ tự thực hiện có thể được thay đổi bằng cách thực thi các mối quan hệ với không gian lặp lại. Các mối quan hệ này được gọi là thành những chuyển đổi.

```
1 cho(int i=0; i<M; i++){ 2
cho(int j=0; j<N ;j++){
3     printf("tôi: %d, j:%d\n",tôi,j)
4   }
5 }
```

Hình 2.3: Một vòng lặp lồng nhau đơn giản

2.2.1 Tập hợp afin và miền lặp

Miền lặp lại biểu diễn các trường hợp của một câu lệnh trong phép tính. tập presburger biểu thị miền lặp. Một tập presburger là một tập hợp có ký hiệu bao gồm một mẫu tuple số nguyên có cấu trúc và một ràng buộc ký hiệu xây dựng tập hợp (công thức presburger). Các biến bị hạn chế trong công thức presburger có thể bao gồm của các biến trong mẫu và hằng số tượng trưng.

$$\{[i, j] \quad Z^{2} \ : 0 \le i < n \quad 0 \le j < n : n \quad Z\} \tag{2.1}$$

Trong Phương trình 2.3, [i, j] là mẫu tuple số nguyên có cấu trúc được chính thức hóa như thể hiện trong Phương trình 2.2 và n là hằng số tượng trưng được đưa vào trong các ràng buộc.

$$[i0, i1, \ldots, id\ 1], d \ge 0 : d, i \quad Z \tag{2.2}$$

Công thức Presburger trong ví dụ Phương trình 2.3 được đưa ra là 0 ≤ i < n 0 ≤ j < n.
Tập hợp presburger có thể được mở rộng hoàn toàn dưới dạng ký hiệu danh sách cho n = 3 như được hiển thị
dưới

{[0, 0], [0, 1], [0, 2],.....[2, 2]}

Mô hình đa diện sử dụng lý luận này để biểu diễn các trường hợp lặp lại như là các điều kiện tiên quyết
bộ burger. Lấy ví dụ vòng lặp được hiển thị trong Hình 2.3 có thể được biểu diễn như
presburger đặt bên dưới

{[i, j] : 0 ≤ i < M   0 ≤ j < N}

2.2.2 Quan hệ truy cập Affine, Bản đồ và Lịch trình

Một bản đồ là một quan hệ nhị phân của một bộ số nguyên có cấu trúc. Bản đồ được biểu thị chính thức
như mối quan hệ presburger-phần tử cặp mô tả bao gồm một cặp mẫu
và công thức Presburger C về các biến trong mẫu.

{[i0, .., id] Z^nply [j0, ..., jn] Z^N : C} (2.3)

Các quan hệ truy cập ánh xạ miền lặp của một câu lệnh thành các truy cập mảng của nó.
Bản đồ quan hệ truy cập rất quan trọng đối với việc phân tích và xác thực sự phụ thuộc của dữ liệu
chuyển đổi. Phụ thuộc đọc sau khi ghi (RAW) và ghi sau khi đọc (WAR)
phải được duy trì sau khi chuyển đổi. Mặt khác, ghi-sau-ghi
Các phụ thuộc (WAW) và đọc sau khi đọc (RAR) không cần phải được bảo toàn.

Hình 2.4 cho thấy S1 và S2 đọc từ các mảng a và t tương ứng được thể hiện

```
đối với (int i = 0; i < N; i++)
S1: t[i] = f(a[i]) }

cho(int i=0; i<N ;i++){
S2: b[i] = g(t[Ni-1]) }
```

Hình 2.4: Danh sách mã hiển thị các lệnh đọc và ghi khi truy cập mảng.

như một mối quan hệ truy cập được hiển thị bên dưới

{S1[i] a[i]}

{S2[i] t[N i 1]}

Các câu lệnh S1 và S2 lần lượt ghi vào t và b như được hiển thị bên dưới

{S1[i] t[i]}

{S2[i] b[i]}

Một Lịch trình là một thứ tự một phần nghiêm ngặt của các phần tử trên tập hợp thể hiện chỉ định thứ tự mà các câu lệnh phải được thực hiện. Câu lệnh S1 được thực hiện trước câu lệnh S2. Thông tin này có thể được biểu diễn bằng lịch trình.

{S1[i] [0, i]}

{S2[i] [1, i]}

Bản đồ có thể được sử dụng để chuyển đổi miền lặp của một câu lệnh để khám phá cơ hội tối ưu hóa. Người ta có thể áp dụng mối quan hệ sau đây cho phép lặp lại

không gian của danh sách mã trong Hình 2.3 để thực hiện trao đổi vòng lặp.

$$IC = \{[i, j] \quad [j, i]\}$$

$$^{\text{tôi}}{}' = IC(I)$$

Thực hiện tạo mã trên I $'$ tạo ra danh sách mã trong Hình 2.1.

```
1 cho(int j=0; j<N; j++){
2 cho(int i=0; i<M ;i++){
3     printf("i: %d, j:%d\n",i,j);
4   }
5 }
```

Danh sách 2.1: Vòng lặp được chuyển đổi sau khi áp dụng Interchange (xem Hình 2.3).

2.2.3 Các phép toán afin

Mô hình đa diện hỗ trợ một số hoạt động được sử dụng trong công việc này.
các hoạt động bao gồm số lượng, vỏ afin, phạm vi của bản đồ, hợp, giao, áp dụng
ánh xạ tới tập hợp, phép bằng nhau, tập con và tập siêu để kể đến một vài ví dụ.

2.3 Mô hình đa diện thưa thớt

Khung đa diện thưa thớt (SPF) mở rộng mô hình đa diện bằng cách bổ sung
chuyển các không gian lặp lại không có afin và các phép biến đổi bằng cách sử dụng các hàm chưa được giải thích.
SPF cung cấp nhiều chức năng tương tự như các công cụ đa diện truyền thống: mã
thế hệ với CodeGen+ [15] được xây dựng trên Omega [57] và tập hợp chính xác và mối quan hệ
các hoạt động khi có các hàm chưa được giải thích với IEGenLib [51].

2.3.1 Các hàm không được diễn giải

Trong các phép tính thưa thớt, việc truy cập mảng trở thành một phần của giới hạn vòng lặp trong com-
sự sắp đặt, điều này không thể được mô hình hóa trong khuôn khổ đa diện. Điều này được gọi là không
affine và được biểu diễn dưới dạng các hàm chưa được diễn giải trong mô hình đa diện thưa thớt.
Các hàm không được diễn giải (UF) là một trường hợp đặc biệt của hằng số tượng trưng. Không được diễn giải
các hàm biểu diễn các cấu trúc dữ liệu như mảng chỉ mục ở định dạng dữ liệu thưa thớt.
Một lợi thế độc đáo của việc giới thiệu các hàm chưa được giải thích là toán học
các thuộc tính của một hàm có thể được thêm vào mô hình do đó mở rộng khả năng áp dụng
của khuôn khổ đa diện. Ví dụ, mọi hàm đều thỏa mãn ràng buộc rằng
nếu cùng một đầu vào được đưa ra, cùng một đầu ra sẽ được tạo ra. Điều này được gọi là chức năng
sự nhất quán, được định nghĩa dưới đây

$$z = x = \quad f(z) = f(x)$$

SPF cung cấp các cơ chế để mô tả thêm các chức năng chưa được giải thích - miền,
phạm vi và các thuộc tính mảng chỉ mục. Thông tin này được sử dụng cho sự phụ thuộc dữ liệu
tối ưu hóa [36] và trong công trình này, để tổng hợp mã.

2.3.2 Không gian lặp lại không afin

Khi xác định không gian cho phép nhân ma trận-vectơ thưa thớt trong mã
liệt kê trong Hình 3.2, các truy cập mảng không affine được sử dụng trong các giới hạn vòng lặp được mô hình hóa như
các chức năng chưa được giải thích như được hiển thị bên dưới

$$I = \{[i, k] : 0 \leq i < N \quad rowptr(i) \leq k < rowptr(i + 1)\}$$

2.3.3 Truy cập dữ liệu và bản đồ không affine

Trong SPF, các hàm không được diễn giải được phép trong các mối quan hệ truy cập dữ liệu. Sparse Phép nhân vectơ ma trận (SPMV) như thể hiện trong Hình 3.2 cho thấy một phép đọc gián tiếp truy cập vào vectơ x được mô hình hóa như một mối quan hệ được hiển thị bên dưới

{[i, k] [t]|t = cột(k)}

2.3.4 Biến đổi phi afin

Các chuyển đổi được thực hiện trong khuôn khổ đa diện thưa thớt được xây dựng trên cơ sở kiểm tra mô hình thực thi tor. Các hàm không được diễn giải có thể được đưa vào một biến đổi và áp dụng cho một hạt nhân tính toán (trình thực thi) với giả định rằng các hàm chưa được giải thích sẽ được tạo ra tại thời điểm biên dịch và được điền vào tại thời điểm chạy (kiểm tra).

2.3.5 Định lý hợp thành

Công trình của Strout et al. [52] đã giới thiệu các định lý thành phần được sử dụng trong công trình này. Công trình của họ đã giới thiệu ứng dụng chính xác và các hoạt động thành phần trong phi affine tập hợp và bản đồ. Bằng chứng cho các định lý được liệt kê có thể được tìm thấy trong công trình được tham chiếu [52]. Trong công trình này, chúng tôi sử dụng định lý Trường hợp 1 1

Định lý 1 (Trường hợp 1: Cả hai quan hệ đều là hàm) Giả sử x, y, v và z là các hàm số các bộ ger trong đó |y| = |v|, F1() và F2() có thể là các hàm affine hoặc không được diễn giải, và C1 và C2 là các tập hợp các ràng buộc liên quan đến đẳng thức, bất đẳng thức, số học tuyến tính- metric và các lệnh gọi hàm chưa được diễn giải trong

{v z|z = F1(v) C1} {x y | y = F2(x) C2}

Kết quả của phép hợp thành là {x z | y, v | y = z z = F1(v) C1 y = F2(x) C2} tương đương với

{x z | z = F1(F2(x)) C1[v/F2(x)] C2[y/F2(x)]}

trong đó C1[v/F2(x)] chỉ ra rằng v nên được thay thế bằng F2(x) trong tập hợp các con-
chủng C1.

2.4 Mô hình thanh tra/thực thi

Một thanh tra tính toán thông tin tại thời điểm chạy để thúc đẩy các chuyển đổi.
trình thực thi—- một trình biến đổi thời gian biên dịch của mã gốc—- sử dụng thông tin com-
được đưa ra bởi thanh tra viên. Thanh tra viên có thể được lý giải là dân số không được giải thích
các chức năng và trình thực thi có thể được lý giải để sử dụng các chức năng chưa được giải thích này để hướng dẫn
sự biến đổi.

2.5 Định dạng thưa thớt

Định dạng thưa thớt mô tả cách lưu trữ tọa độ thưa thớt và dữ liệu tương ứng
và thường dựa trên các định dạng ma trận thưa thớt. Dữ liệu được cho là thưa thớt nếu nó
có một lượng tương đối lớn các số khác không. Hình 2.6 cho thấy một số
các định dạng ma trận thưa thớt phổ biến bao gồm định dạng tọa độ. Tọa độ
Định dạng (COO) lưu trữ từng số khác không và lưu trữ các chỉ số tọa độ riêng biệt
các mảng được sắp xếp theo chiều. Compressed Sparse Row (CSR) nén

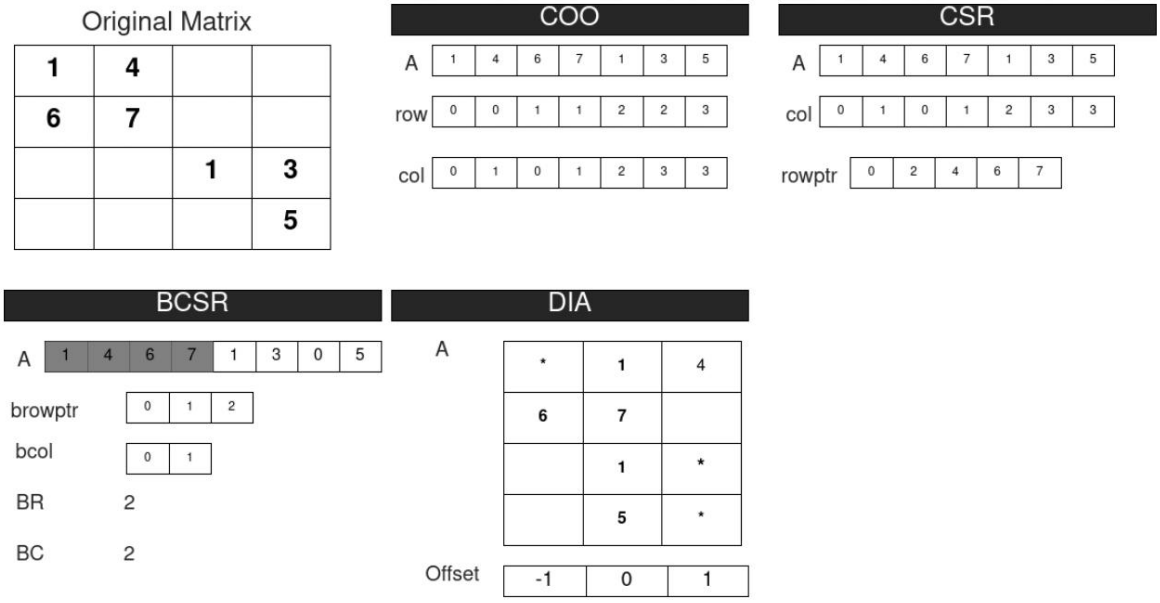
các hàng và mỗi số khác không được sắp xếp và có một cột không nén tương ứng tọa độ. Blocked Compressed Sparse Row chia ma trận dày đặc thành các khối và nén các hàng bị chặn. Đường chéo (DIA) nén từng đường chéo của ma trận.

Các định dạng tenxơ thưa thớt được thiết kế riêng cho dữ liệu có chiều cao hơn bao gồm HICOO [34] và Alto [28]. Các định dạng này phức tạp hơn và liên quan đến việc sắp xếp và các cấu trúc dữ liệu khác. Định dạng Morton Coordinate (MCOO) bảo toàn vị trí của điểm dữ liệu trong tọa độ đa chiều của nó. Hệ nhị phân của tọa độ mỗi số không được xen kẽ và giá trị kết quả sắp xếp vị trí của dữ liệu giá trị trong định dạng.

Khái niệm chính cho mỗi định dạng tenxơ thưa thớt là trợ từ (hoặc chỉ số) mảng cung cấp tọa độ dày đặc của dữ liệu tương ứng. Khi kết hợp lại với nhau, chúng cung cấp ánh xạ từ không gian lặp sang không gian dữ liệu.

```
đối với (int i = 0; i < N; i++) { đối
  với (int k = rowptr [i]; k < rowptr [i + 1]; k++) { y [i] + = a [k] *
     x [col [k]];
  }
}
```

Hình 2.5: Phép nhân vectơ ma trận thưa thớt



Hình 2.6: Định dạng ma trận thưa thớt.

CHƯƠNG 3

TÍNH TOÁN BIỂU DIỄN TRUNG GIAN

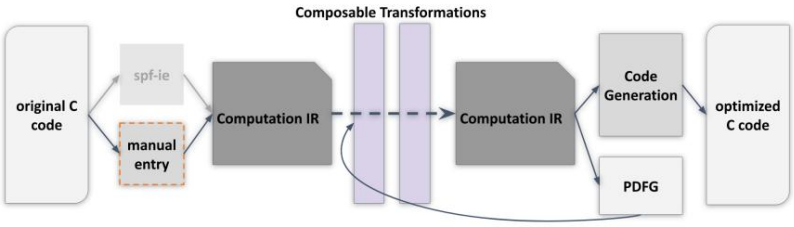
Khung đa diện thưa thớt mở rộng mô hình đa diện bằng cách hỗ trợ
không gian lặp lại không afin và các phép biến đổi sử dụng các hàm không được giải thích. Không
các hàm được diễn giải là các hằng số tượng trưng biểu diễn các cấu trúc dữ liệu như
mảng chỉ mục trong các định dạng dữ liệu thưa thớt. SPF cung cấp nhiều chức năng tương tự
tính chất như các công cụ đa diện truyền thống: tạo mã với CodeGen+ [15] được xây dựng trên
Omega [57] và các phép toán thiết lập và quan hệ chính xác khi có sự hiện diện của các phép toán chưa được giải thích
chức năng với IEGenLib [51]. Tuy nhiên, các công cụ này không được tích hợp chặt chẽ và
mỗi thành phần đều có API cấp thấp riêng để tương tác với từng thành phần.
các thành phần bao gồm các cách để biểu diễn các tập hợp và mối quan hệ bên trong, các câu lệnh trong một
tính toán, tạo mã và phân tích. Trong một số trường hợp, một số công cụ hỗ trợ mã
thế hệ [16, 17, 45, 55] trong khi một số khác thì không [2].

Chương này thảo luận về biểu diễn trung gian SPF (SPF-IR) và
API tính toán. API tính toán cung cấp thông số kỹ thuật chính xác về cách
kết hợp các thành phần riêng lẻ của SPF để tạo ra một đại diện trung gian
tation. IR này có thể trực tiếp tạo ra Đồ thị luồng dữ liệu đa diện (PDFG) [21] và
biên dịch các hoạt động đồ thị được xác định cho PDFG thành các mối quan hệ được IEGenLib sử dụng để
thực hiện các phép biến đổi. Trong công trình này, chúng tôi mở rộng biểu diễn PDFG để xử lý
vòng lặp không affine với lồng nhau không hoàn hảo và sự phụ thuộc mang vòng lặp.

Hình 3.1 cho thấy tổng quan về khuôn khổ tối ưu hóa của chúng tôi và nơi hợp nhất
API putation phù hợp với quy trình. Spf-i/e được tô sáng màu xám trong hình là một công cụ
tự động dịch từ mã nguồn sang IR của chúng tôi và được công khai [5]. Trong
công việc chúng tôi tập trung vào việc dịch thủ công mã nguồn gốc sang SPF IR. SPF
IR tự nó hỗ trợ các chuyển đổi có thể cấu hình nhờ vào đa diện thưa thớt
khung. Các hoạt động đồ thị trên PDFG tạo ra các phép biến đổi có thể ghép lại và
được áp dụng cho các tuyên bố trong IR.

Chương này trả lời câu hỏi nghiên cứu đầu tiên lý giải về những gì cấu thành
hướng dẫn một biểu diễn nội bộ hoàn chỉnh cho các phép tính thưa thớt và cách thức biểu diễn nội bộ đó
biểu diễn được chuyển đổi bằng các phương pháp đã biết. Để trả lời câu hỏi này, chúng tôi
giới thiệu một biểu diễn trung gian chính thức sử dụng API SPF cấp thấp như
CodeGen+, Omega và IEGenlib cung cấp một điểm vào duy nhất để mô tả thưa thớt
tính toán theo cách phù hợp cho các phép biến đổi. Chúng tôi cũng cung cấp các tiêu chuẩn tốt
các hoạt động đã biết như hợp nhất, loại bỏ biến chết, nhúng, lên lịch lại và
các hoạt động chuyển đổi không liên quan trên các câu lệnh trong biểu diễn. CHiLL [27],
một công cụ SPF, cung cấp giao diện dựa trên tập lệnh cho SPF và hỗ trợ non-affine
chuyển đổi bằng cách sử dụng Omega. Omega có những hạn chế về độ chính xác của hoạt động
các hành vi liên quan đến các chức năng chưa được giải thích, cũng như các hạn chế về cách chúng phải được
được thể hiện. Công trình này khắc phục những hạn chế của các phương pháp tiếp cận trước đây bằng cách sử dụng
công cụ thao tác tập hợp và quan hệ chính xác sử dụng IEGenlib.

Việc hình dung một phép tính dưới dạng đồ thị là một phương pháp nổi tiếng trong việc hướng dẫn tối ưu hóa
chuyên gia. Chúng tôi tiếp tục trả lời các câu hỏi nghiên cứu này bằng cách cho phép tự động
tạo ra Biểu đồ luồng dữ liệu đa diện (PDFG) để cung cấp cho các chuyên gia nghiên cứu
lý do tốt hơn về việc tối ưu hóa.



Hình 3.1: Tổng quan về đường ống tối ưu hóa

3.1 Tính toán: Một lớp C++

Giao diện với SPF được triển khai như một lớp C++ trong IEGenLib.
lớp tính toán chứa tất cả thông tin cần thiết để thể hiện một phép tính hoặc
một loạt các phép tính. Điều này bao gồm: không gian dữ liệu, các câu lệnh, sự phụ thuộc dữ liệu,
và lịch trình thực hiện. Phần này mô tả thiết kế và giao diện cho từng
các yếu tố này. Phép nhân vectơ ma trận được sử dụng như một ví dụ chạy trong suốt
phần còn lại của bài báo này. Hình 3.1 và 3.2 cho thấy phiên bản dày đặc và thưa thớt.

```
1 cho (i = 0; i < N; i++) {
2       đối với (j=0; j<M; j++) {
3           y[i] += A[i][j] * x[j];
4       }
5 }
```

Liệt kê 3.1: Ma trận vectơ dày đặc nhân

```
1 cho (i = 0; i < N; i++) {
2       đối với (k=rowptr[i]; k<rowptr[i+1]; k++) {
3           j = cột[k];
4           y[i] += A[k] * x[j];
5       }
6 }
```

Liệt kê 3.2: Phép nhân vectơ ma trận thưa thớt

3.2 Không gian dữ liệu

Không gian dữ liệu biểu thị một tập hợp các địa chỉ bộ nhớ duy nhất về mặt khái niệm. Không gian dữ liệu có 0 chiều tương đương với một số vô hướng. Mỗi tổ hợp dữ liệu tên không gian và bộ dữ liệu đầu vào được đảm bảo ánh xạ tới một không gian duy nhất trong bộ nhớ tuổi thọ của không gian dữ liệu. Ví dụ, D(s), đảm bảo rằng đối với mỗi tuple s sẽ có một vị trí bộ nhớ duy nhất và trong suốt thời gian tồn tại của D vị trí bộ nhớ sẽ không chồng lấn với bất kỳ vị trí bộ nhớ nào khác trong không gian dữ liệu trực tiếp.

Theo mặc định, tất cả các không gian dữ liệu đều là chung. Chúng được định nghĩa bằng cú pháp D(s). Ví dụ, đối với một bộ dữ liệu đầu vào 3 tham số i, j, k thì không gian dữ liệu có thể được biểu diễn như D(i, j, k). Không gian dữ liệu này chỉ có thể được ghi một lần nhưng được đọc từ bất kỳ số nào của thời gian. Ngoại lệ của quy tắc này là đối với các hoạt động tích lũy khi một vị trí dữ liệu trong không gian dữ liệu có thể được ghi nhiều lần (+ =, =, = , tối đa, tối thiểu...).

Các không gian dữ liệu được biểu diễn trong ví dụ nhân vectơ ma trận bao gồm: y, A và x. Trong phiên bản thưa thớt, các mảng chỉ mục rowptr và col cũng được xem xét không gian dữ liệu. Tuy nhiên, vì chúng được sử dụng trong giới hạn vòng lặp và để truy cập vào một không gian dữ liệu khác, chúng phải không đổi trong suốt thời gian tính toán này. Do đó, chúng không phải là một phần bắt buộc của định nghĩa Tính toán.

```
// dày đặc
Tính toán* dsComp = tính toán mới ();
dsComp->addDataSpace("y");
dsComp->addDataSpace("A");
dsComp->addDataSpace("x");

// thưa thớt
Tính toán* spsComp = tính toán mới ();
```

```
spsComp->addDataSpace("y");
spsComp->addDataSpace("A");
spsComp->addDataSpace("x");
```

3.3 Các tuyên bố

Các câu lệnh thực hiện các hoạt động đọc và ghi trên không gian dữ liệu. Chúng tôi hạn chế định nghĩa của các câu lệnh là các khối cơ bản. Có một mục nhập và một lối thoát duy nhất từ mỗi khối mã được biểu diễn.

Tất cả các câu lệnh đều có một miền lặp liên kết với chúng. Lặp này miền giá trị là tập hợp chứa mọi trường hợp của câu lệnh và không có thứ tự cụ thể. Nó thường được thể hiện như là tập hợp các trình lặp mà câu lệnh chạy qua, tùy thuộc vào đến các ràng buộc của vòng lặp của chúng (giới hạn vòng lặp). Khối mã sau đây cho thấy cách tạo một câu lệnh bằng cách sử dụng API tính toán. Một câu lệnh được viết dưới dạng chuỗi như được thấy ở dòng 2 và 5 bên dưới cho các trường hợp dày đặc và thưa thớt tương ứng. miền lặp được chỉ định là một tập hợp sử dụng cú pháp IEGenLib, ngoại trừ của việc phân định tất cả các không gian dữ liệu bằng $, điều này có thể được thấy ở dòng 3 và 6 bên dưới.

```
Stmt* ds0 = Stmt mới (
"y(i) += A(i,j) * x(j);",
"{[i,j]: 0 <= i < N && 0 <= j < M}", ...

Stmt* sps0 = Stmt mới (
"y(i) += A(k) * x(j)",
"{[i,k,j]: 0 <= i < N && rowptr(i) <= k < rowptr(i+1) && j = col(k
    )}", ...
```

3.4 Mối quan hệ 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. Tính toán một phép đóng cung cấp mối quan hệ phụ thuộc giữa các câu lệnh và ràng buộc sắp xếp một phần trên tính toán. Trong các ví dụ đang chạy của chúng tôi, dữ liệu đọc và ghi có thể được chỉ định là được viết bên dưới.

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

// thưa thớt
 ...
{ // đọc
{"y", "{[i,k,j]->[i]}"},
{"A", "{[i,k,j]->[k]}"},
{"x", "{[i,k,j]->[j]}"}
},
{ // viết
{"y", "{[i,k,j]->[i]}"}
}
```

3.5 Lịch trình thực hiện

Lịch trình thực hiện được xác định bằng cách sử dụng các hàm phân tán cần thiết 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 xuất lịch trình dưới dạng số nguyên bộ có thể được sắp xếp theo thứ tự từ điển với các bộ khác để xác định việc thực hiện chính xác thứ tự của một nhóm các câu lệnh. Các trình lặp thường được sử dụng như một phần của đầu ra bộ, biểu thị rằng giá trị của các trình lặp ảnh hưởng đến thứ tự của câu lệnh.
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. Ngoài ra, trong thứ tự từ điển, tất cả các trường hợp của câu lệnh với i = 1 sẽ đi trước tất cả các trường hợp với i = 2, bất kể giá trị của j.

```
31 /* Tham số thứ 3 cho hàm tạo Stmt */
32 // dày đặc
33 "{[i,j] ->[0,i,0,j,0]}"
34
35 // thưa thớt
36 "{[i,k,j]->[0,i,0,k,0,j,0]}"
```

Hình 3.2 cho thấy thông số kỹ thuật đầy đủ của hai phép tính, ma trận dày đặc đầu tiên phép nhân vectơ theo sau là phép nhân vectơ ma trận thưa.

3.6 Tạo mã

Lớp Computation giao diện với CodeGen+ [15] để tạo mã. Mã-Gen+ sử dụng các tập hợp và mối quan hệ Omega để quét đa diện. Các tập hợp và mối quan hệ Omega

có những hạn chế khi có các hàm chưa được giải thích. Các hàm chưa được giải thích
bị giới hạn bởi quy tắc tiền tố. Quy tắc này nêu rằng một hàm không được diễn giải phải là
tiền tố của khai báo tuple. Các hàm không được diễn giải không thể có biểu thức như
tham số. Việc tạo mã khắc phục hạn chế này bằng cách sửa đổi các tham số chưa được giải thích
các chức năng trong IEGenLib tuân thủ Omega, trong khi lưu trữ bản đồ của bản gốc
chức năng không được diễn giải thành chức năng không được diễn giải đã sửa đổi của nó. Sự tách biệt của rep-
các yêu cầu chuyển đổi và tạo mã cho phép các hoạt động chính xác trong
chuyển đổi trong khi vẫn tận dụng chức năng của CodeGen+ cho đa diện
quét.

Hình 3.3 cho thấy kết quả của việc tạo mã cho vectơ ma trận thưa thớt
phép tính nhân được định nghĩa trong Hình 3.2. Dòng 2 của Hình 3.3 định nghĩa một
macro cho câu lệnh s0, dòng 9 - 13 ánh xạ lại Omega tuân thủ không được giải thích
chức năng trở lại ban đầu của nó. Dòng 15 - 20 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.

3.7 Hình dung phép tính trên đồ thị

Việc trực quan hóa các phép tính dưới dạng biểu đồ luồng dữ liệu cung cấp cho các chuyên gia hiệu suất một cách phù hợp
xem xét lý do về các chuyển đổi cho các cơ hội tối ưu hóa. Để đạt được mục đích này,
PDFG thể hiện các phép tính thông thường bằng cách kết hợp mô hình đa diện
và đồ thị luồng dữ liệu [21]. Đồ thị gốc có một số hạn chế nhất định: vòng lặp được thực hiện
sự phụ thuộc không được thể hiện và các vòng lặp không hoàn hảo không được hỗ trợ. Điều này
phần này mô tả cách khắc phục những hạn chế này. Ngoài ra, việc tạo ra
PDFG bằng cách duyệt SPF IR được tích hợp với API tính toán. Sau một

Tính toán được tạo ra như trong Hình 3.2, một hàm có thể được gọi là
xuất PDFG dưới dạng tệp chấm.

Trong PDFG gốc, các hộp chữ nhật được tô bóng biểu diễn không gian dữ liệu và đảo ngược
hình tam giác biểu diễn các câu lệnh. Trong PDFG mở rộng, các hộp chữ nhật được tô bóng
đại diện cho các miền, các hộp chữ nhật trong suốt đại diện cho các không gian dữ liệu và các hình tròn
hộp chữ nhật biểu diễn các câu lệnh. Các cạnh biểu diễn các lần đọc và ghi trong cả hai
PDFG gốc và mở rộng. PDFG mở rộng hiện không thể hiện
loại và kích thước của không gian dữ liệu trong phép tính.

Các phụ thuộc vòng lặp được mang theo và các lồng vòng lặp không hoàn hảo là các mẫu quan trọng để
cân nhắc khi quyết định áp dụng tối ưu hóa nào. Các phụ thuộc vòng lặp được thực hiện tham chiếu
để mã hóa các mẫu trong đó một lần lặp của vòng lặp đọc hoặc ghi dữ liệu được tạo ra bởi
một lần lặp lại khác của cùng một vòng lặp. Một lồng vòng lặp không hoàn hảo là một lồng có các câu lệnh
ở nhiều cấp độ như thể hiện trong Hình 3.4.

3.8 Phụ thuộc vòng lặp mang

Biểu diễn hiện tại không có khả năng hình dung sự hiện diện của một vòng lặp
phụ thuộc mang theo. Hình 3.6 cho thấy PDFG cho ví dụ giải quyết về phía trước trong
Hình 3.4. Trong ví dụ, mã chứa một vòng lặp mang sự phụ thuộc cho dữ liệu
khoảng trống u trong các câu lệnh S1 và S2. Tuy nhiên, đồ thị PDFG ban đầu trong Hình 3.6
không hình dung được sự phụ thuộc của vòng lặp mang.

Hình 3.5 cho thấy PDFG mở rộng. Cạnh đi ra từ nút dữ liệu u đến
nút lệnh S1 tại chỉ mục j hiển thị quyền truy cập đọc trong vòng lặp j.
cạnh từ nút lệnh S2 đến nút dữ liệu u tại chỉ mục i hiển thị quyền truy cập ghi
trong vòng lặp i. Điều này cho thấy một vòng lặp phụ thuộc vào i.

3.9 Lồng vòng lặp không hoàn hảo

Ví dụ giải quyết về phía trước cũng thể hiện một mô hình lồng vòng lặp không hoàn hảo: trạng thái-
Các câu lệnh S0 và S2 nằm trong vòng lặp i bên ngoài trong khi câu lệnh S1 nằm trong vòng lặp j bên trong.
PDFG gốc trong Hình 3.6 không thể hiện mẫu này. Thiết kế được cập nhật
trong PDFG mở rộng hiển thị trong Hình 3.5 trực quan hóa mẫu vòng lặp không hoàn hảo.

3.10 Các phép toán trên tính toán

Các phép tính được tạo thành từ các Bộ và Quan hệ IEGenLib. Chức năng của IEGenLib
Tính chất này được sử dụng trực tiếp để thao tác các thành phần của phép tính. Cốt lõi
các hoạt động được triển khai trong IEGenLib bao gồm đảo ngược, soạn thảo, áp dụng và các hoạt động liên quan
các chức năng hỗ trợ. Sử dụng các hoạt động này, chúng ta có thể thực hiện chức năng nội tuyến
và các phép biến đổi vòng lặp trong lớp Tính toán.

3.11 Nội tuyến

Tính toán nội tuyến xử lý sự phức tạp của việc tạo biểu diễn SPF
của các hàm gọi các hàm khác. Các thể hiện của phép tính cần có thể tái sử dụng
tương tự như các chức năng có thể tái sử dụng.

Mỗi chức năng trong mã nguồn gốc được biểu diễn như một phép tính. Khi
một lệnh gọi hàm đầu tiên được gặp trong nguồn, một phép tính phải được tạo cho
nó. Sau đó, nội dung của phép tính đó được chèn (nội tuyến) vào trình gọi
Tính toán đang được xây dựng. Nếu cùng một hàm được gọi nhiều lần,
Tính toán đã được tạo ra cho nó sẽ được sử dụng lại. Quá trình nội tuyến
có thể tiếp tục tới bất kỳ độ sâu lồng nhau nào, nếu một hàm đang được gọi cũng gọi các hàm khác.

Hàm nội tuyến chịu trách nhiệm:

- tránh xung đột tên giữa các biến trong hàm gọi và hàm được gọi,
- tạo ra các câu lệnh gán cho các tham số hàm,
- cung cấp các giá trị trả về của người được gọi cho người gọi và
- cập nhật miền lặp lại, lịch trình thực hiện và sự phụ thuộc dữ liệu trong câu lệnh nội tuyến.

Trước khi nhúng, tên của không gian dữ liệu và trình lặp trong lệnh gọi được thêm tiền tố với một chuỗi duy nhất để tránh va chạm với không gian dữ liệu của người gọi hoặc với các các trường hợp của cùng một phép tính nội tuyến trong cùng một phạm vi. Sự thay đổi này là được phản ánh trong chuỗi mã nguồn được lưu trữ của câu lệnh, cũng như các phần khác của sự biểu diễn của nó liên quan đến những cái tên này.

Để đơn giản hóa mọi thứ, các giá trị trả về và đối số cho một hàm nội tuyến bị giới hạn ở tên của không gian dữ liệu hoặc các giá trị theo nghĩa đen. Để truyền một biểu thức thành một hàm, như A[0] hoặc x+y, trước tiên nó phải được gán cho một hàm tạm thời biến sau đó có thể được truyền vào. Để bảo toàn ngữ nghĩa gọi ban đầu của chương trình, khi các đối số được truyền cho một hàm, các câu lệnh được tạo ra để tuyên bố mỗi tham số của nó bằng với các giá trị được truyền vào. Không có quy trình tương đương xảy ra với các giá trị trả về, vì chúng có khả năng được sử dụng trong nhiều loại khác nhau của ngữ cảnh (gán cho các biến, được sử dụng ngay lập tức trong một biểu thức hoặc bị bỏ qua hoàn toàn). Do đó, quá trình nội tuyến trả về các giá trị được trả về bởi hàm nội tuyến, dưới dạng chuỗi, có thể được sử dụng theo bất kỳ cách nào mà người gọi thấy phù hợp.

Các miền lặp lại, lịch trình thực hiện và quan hệ truy cập dữ liệu được cập nhật thành phản ánh bối cảnh xung quanh mà các câu lệnh đã được chèn vào. Ví dụ,

nếu một hàm được gọi trong một vòng lặp và chính hàm được gọi cũng chứa một vòng lặp, thì
việc thể hiện các tuyên bố sâu sắc nhất được điều chỉnh để phản ánh rằng chúng hiện đang
lồng vào bên dưới cả hai vòng lặp.

3.12 Biến đổi vòng lặp như các phép toán đồ thị

Trong công trình này, chúng tôi cung cấp các hoạt động hợp nhất và sắp xếp lại trên đồ thị.
các hoạt động trên đồ thị được chuyển thành các phép biến đổi được thực hiện bởi giao diện của chúng tôi
API để điều khiển các thành phần SPF.

3.12.1 Sự hợp nhất

Đây là một phép biến đổi nối hai câu lệnh từ các vòng lặp riêng biệt thành một vòng lặp duy nhất
vòng lặp. Có nhiều loại hợp nhất vòng lặp khác nhau, bao gồm hợp nhất giảm đọc và
sự kết hợp giữa nhà sản xuất và người tiêu dùng. Giảm đọc liên quan đến việc kết hợp các vòng lặp đọc từ
cùng một vị trí bộ nhớ trong khi sự kết hợp giữa nhà sản xuất và người tiêu dùng liên quan đến các vòng lặp hợp nhất trong đó
một vòng lặp ghi một biến sau đó được vòng lặp thứ hai đọc.

Hoạt động đồ thị để hợp nhất hai câu lệnh với nhau ở một cấp độ cụ thể của
lịch trình thực hiện được biểu thị là fuse(S1, S2, level) trong IR của chúng tôi. S1 và S2 là
các câu lệnh cần được hợp nhất và mức độ cho biết độ sâu cần hợp nhất. Chỉ định
độ sâu để hợp nhất cho phép linh hoạt hơn trong hoạt động hợp nhất. Sau khi hợp nhất, S2 sẽ
được đặt hàng ngay sau S1.

3.12.2 Lên lịch lại

Hoạt động lên lịch lại bao gồm việc di chuyển một câu lệnh đến một vị trí mới trong
đồ thị và do đó thay đổi lịch trình thực hiện của nó. Việc lập lại lịch trình tự nó là

không phải là một sự tối ưu hóa, tuy nhiên, nó phơi bày các cơ hội tối ưu hóa. Việc lên lịch lại hoạt động đồ thị được biểu thị là reschedule(S1, S2) trong IR của chúng tôi. Điều này sẽ gây ra tuyên bố Buổi S1 sẽ được lên lịch lại để diễn ra trước buổi S2.

3.12.3 Loại bỏ biến chết

Việc loại bỏ các tính toán dư thừa là một kỹ thuật tối ưu hóa giúp cải thiện hiệu suất của một ứng dụng. Trong một đồ thị, một nút lệnh sẽ chết nếu nó ghi đến một nút dữ liệu không bao giờ được đọc từ đó. Trong công việc của chúng tôi, chúng tôi cung cấp sự chuyển đổi này như một tùy chọn cho biểu diễn trung gian tính toán. Cần lưu ý rằng hoạt động này loại bỏ các phép gán chết khỏi phép tính trung gian đại diện. Chúng tôi triển khai chức năng này bằng cách thực hiện tìm kiếm theo chiều rộng từ các nút dữ liệu chết trong đồ thị, chúng tôi tiếp tục xóa các nút câu lệnh một cách đệ quy cho đến khi chúng ta đạt đến các nút dữ liệu được đọc từ các câu lệnh khác. Một chiều rộng đầu tiên cách tiếp cận cho phép thuật toán của chúng tôi loại bỏ các nhiệm vụ chết ở mỗi cấp độ khi đi ngược lại đồ thị. Hoạt động này dẫn đến một kết quả nhỏ hơn đáng kể Biểu đồ luồng dữ liệu. Mã có vẻ như đã chết nhưng có tác dụng phụ không bị loại bỏ. Điều này bao gồm mã ghi vào các vị trí dữ liệu được đánh dấu là tham số hàm và các vị trí bộ nhớ được đặt tên bí danh.

```
// mvm dày đặc
Tính toán* dsComp = tính toán mới ();

// thêm không gian dữ liệu
dsComp->addDataSpace("y");
dsComp->addDataSpace("A");
dsComp->addDataSpace("x");

Stmt* ds0 = Stmt mới (
    // 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]}"}
    },
    { // ghi dữ liệu
       {"y", "{[i,j]->[i]}"}
    }
);
dsComp->addStmt(ds0);

// mvm thưa thớt
Tính toán* spsComp = tính toán mới ();

// thêm không gian dữ liệu
spsComp->addDataSpace("y");
spsComp->addDataSpace("A");
spsComp->addDataSpace("x");

Stmt* sps0 = Stmt mới (
"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]}"}
    }
);
spsComp->addStmt(sps0);
```

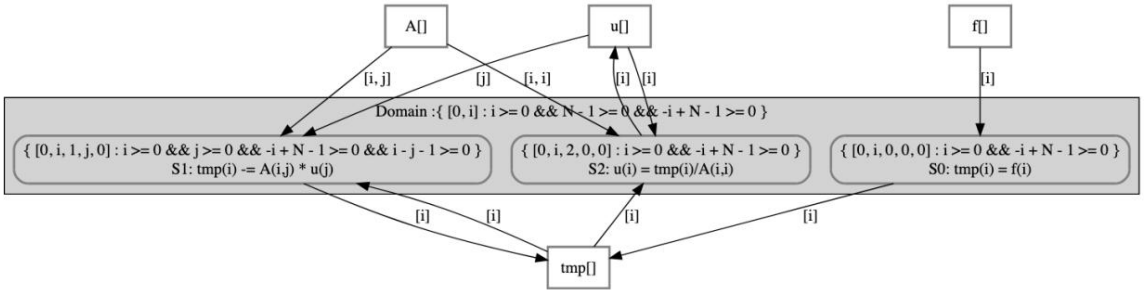
Hình 3.2: Đặc tả API tính toán cho vectơ ma trận dày đặc và thưa thớt nhân lên

```
#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]

#xác định col_0(__tv0, __tv1, __tv2, __tv3) col(__tv3) # xác định rowptr(t0)
rowptr[t0] #xác định rowptr_1(__tv0, __tv1)
rowptr(__tv1) #xác định 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 col(t0) #undef
col_0(__tv0, __tv1, __tv2, __tv3) #undef rowptr(t0) #undef
rowptr_1(__tv0, __tv1)
#undef rowptr_2(__tv0, __tv1)
```

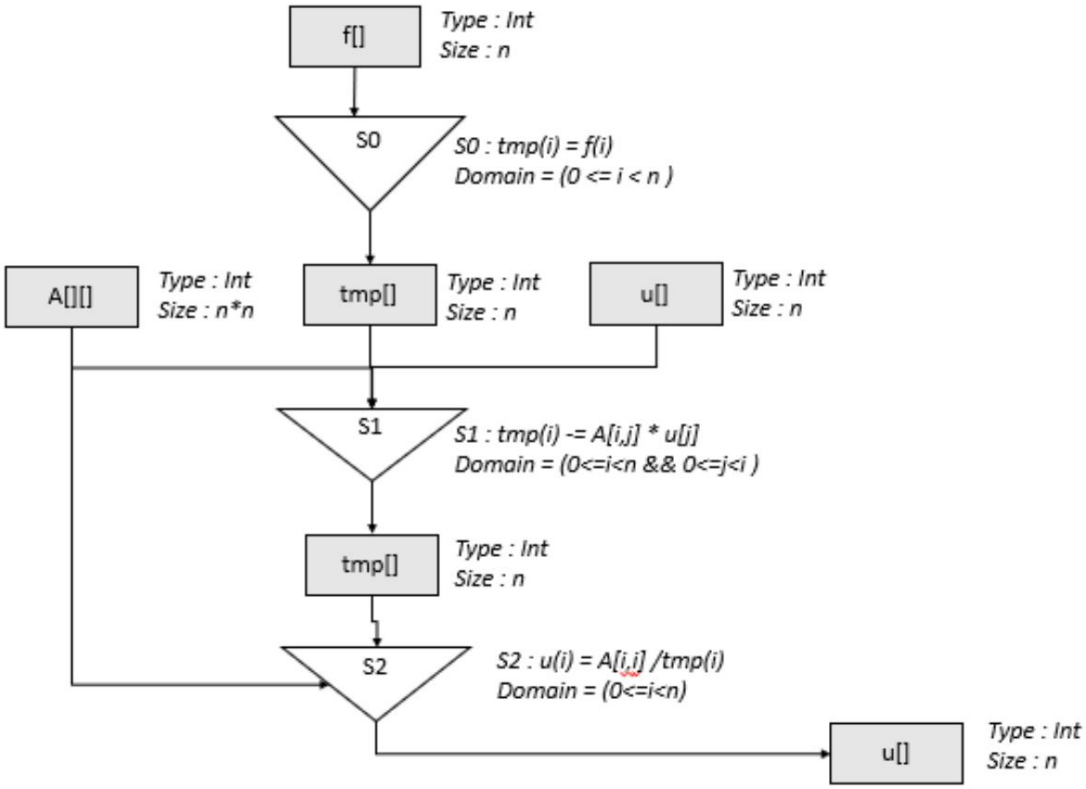
Hình 3.3: Mã SPMV

```
cho (i=0; i<N; i++){
      S0: tmp(i) = f(i); đối
      với(j=0; j<i; j++){
          S1: tmp(i) -= A(i, j)*u(j);
      }
      S2: u(i) = tmp(i)/A(i, i);
}
```

Hình 3.4: Giải quyết theo hướng tiến



Hình 3.5: PDFG mở rộng. Đồ thị này được tạo tự động từ lớp Tính toán và bao gồm các phụ thuộc vòng lặp và lồng vòng lặp bất thường. Thứ tự từ điển của các bộ trong tập hợp được đính kèm vào mỗi câu lệnh thông báo thứ tự thực hiện của nó.



Hình 3.6: PDFG gốc cho ví dụ Giải theo hướng tiến.

CHƯƠNG 4

TỔNG HỢP MÃ

Nhiều ứng dụng khoa học xử lý dữ liệu thưa thớt. Dữ liệu thưa thớt nếu nó có tương đối phần trăm lớn các số không. Các ứng dụng này sử dụng các định dạng tenxơ thưa thớt để giảm yêu cầu về bộ nhớ. Với số lượng ngày càng tăng của các định dạng tenxơ thưa thớt và cần có các thói quen được tối ưu hóa cao để dịch chuyển giữa chúng, các phương pháp tự động được yêu cầu tổng hợp mã dịch. Chương này trình bày một cách tiếp cận mô tả các định dạng tenxơ thưa thớt và tổng hợp mã dịch giữa chúng.

Tổng hợp mã chuyển đổi định dạng có hiệu suất cao được ưa chuộng hơn là mã viết tay. ting tất cả các kết hợp có thể. Tối ưu hóa thêm các mã viết tay cho định dạng chuyển đổi là tẻ nhạt và khó duy trì. Lựa chọn tốt nhất của định dạng tenxơ thưa thớt thay đổi theo các mẫu tính toán và mẫu thưa thớt của dữ liệu. Một khi lựa chọn của định dạng đã được thực hiện, các thói quen được tối ưu hóa chuyển đổi mã thưa thớt từ một định dạng sang một định dạng khác là bắt buộc. Chuyển đổi định dạng thưa thớt có thể từ bất kỳ định dạng thưa thớt sang bất kỳ định dạng thưa thớt nào khác, tạo ra một không gian chuyển đổi rộng lớn. Hơn nữa, lựa chọn đó có thể thay đổi trong suốt thời gian thực thi ứng dụng.
Như một ví dụ đơn giản, hãy xem xét một tenxơ thưa thớt được sử dụng trong nhiều giai đoạn của tính toán và đôi khi sẽ được đọc ở chế độ đầu tiên và sau đó ở chế độ cuối cùng.
Việc thay đổi định dạng giữa các giai đoạn có thể có lợi tùy thuộc vào số lượng số lần các hoạt động được thực hiện.

Các kỹ thuật tự động hiện tại bị giới hạn bởi các loại định dạng thưa thớt có thể được thể hiện và chuyển đổi giữa. Giải pháp của chúng tôi hỗ trợ các định dạng yêu cầu sắp xếp như ALTO [28] và HICOO [34]. Các định dạng này (ALTO và HICOO), sử dụng thứ tự Morton trên các chỉ số dữ liệu để cải thiện vị trí khi thực hiện chế độ tính toán bất khả tri. Các cách tiếp cận khác để tổng hợp chuyển đổi định dạng thưa thớt không hỗ trợ các định dạng này [3, 54].

Một cơ chế biểu cảm và chính xác để mô tả hiện tại và phát triển mới thưa thớt định dạng tensor là một thành phần quan trọng trong thuật toán tổng hợp mã. Chúng tôi đề xuất mô tả các định dạng tenxơ thưa thớt như các hàm từ không gian lặp thưa thớt đến không gian dày đặc tọa độ. Các hàm được thể hiện dưới dạng các mối quan hệ và mỗi hàm không được diễn giải được mô tả thêm bằng cách cung cấp miền, phạm vi và các ràng buộc chung cho từng ràng buộc. Các mô tả định dạng thưa thớt được thể hiện bằng cách sử dụng khuôn khổ đa diện thưa thớt.

Khung đa diện thưa thớt (SPF) cung cấp cú pháp và các hoạt động cần thiết để chỉ định các định dạng thưa thớt và tổng hợp mã từ các thông số kỹ thuật đó. SPF hỗ trợ nhiều phép biến đổi vòng lặp bao gồm hợp nhất, xiên, mở cuộn, lát gạch và nhiều phép khác. Bằng cách tổng hợp trực tiếp mã định dạng thưa thớt thành SPF và thể hiện bản gốc tính toán trong SPF, cả hai có thể được tối ưu hóa song song.

SPF Internal Representation (SPF-IR) cung cấp một giao diện hướng đối tượng để truy cập các hoạt động và yêu cầu SPF cho một phép tính được chỉ định đầy đủ [41]. Nó có thể biểu thị một lớp rộng các phép tính bao gồm cả những phép tính có lồng vòng lặp không hoàn hảo và phụ thuộc vòng lặp mang. Công trình này đề xuất một tổng hợp chuyển đổi định dạng thưa thớt kỹ thuật phát ra mã được thể hiện trong SPF bằng cách sử dụng SPF-IR.

Chương này trả lời câu hỏi thứ hai trong luận án này, một nghiên cứu thứ hai về cách chuyển đổi dữ liệu có thể được tự động hóa mà không phá vỡ sự trừu tượng trong sự biểu diễn nội bộ. Những đóng góp của chương này bao gồm:

- Một đặc điểm kỹ thuật của các định dạng tenxơ thưa thớt sử dụng khuôn khổ đa diện thưa thớt, Và

- Một phương pháp tổng hợp các chương trình chuyển đổi định dạng thưa thớt.

Chúng tôi chứng minh tính biểu cảm của đặc tả bằng một tập hợp các thông tin chung định dạng tenxơ thưa thớt và chúng tôi đánh giá tính chính xác của thuật toán tổng hợp. A so sánh hiệu suất giữa công việc của chúng tôi và việc triển khai của TACO như thể hiện trong Phần 4.3 cho thấy cách tiếp cận của chúng tôi có tính cạnh tranh hoặc vượt trội hơn TACO trong các trường hợp nơi có thể so sánh được.

4.1 Chuyển đổi định dạng Tensor thưa thớt

Công trình này giới thiệu một phương pháp tổng hợp chương trình thanh tra bằng cách sử dụng suy diễn lý luận cho việc chuyển đổi định dạng tenxơ thưa thớt. Định dạng thưa thớt được mô tả bằng cách sử dụng Các mô tả định dạng. Các mô tả được thiết kế để hỗ trợ nhiều loại tenxơ thưa thớt định dạng, cụ thể là những định dạng phụ thuộc vào việc sắp xếp do người dùng xác định. Sắp xếp do người dùng xác định cho phép người dùng chỉ định các ràng buộc sắp xếp lại trong mô tả tenxơ thưa thớt. Định dạng các mô tả được kết hợp để tạo ra một ánh xạ từ không gian thưa thớt này sang không gian thưa thớt khác. bản đồ đóng vai trò là nguồn của các mối quan hệ ràng buộc giữa nguồn và đích cấu trúc dữ liệu và là cơ sở của tổng hợp thanh tra.

Thuật toán tổng hợp thanh tra tạo ra một biểu diễn trung gian SPF đảm bảo các chức năng chưa được giải thích trong định dạng đích được tạo ra và đáp ứng tất cả các ràng buộc. Biểu diễn trung gian kết quả là một chuỗi vòng lặp thưa thớt điền vào các hàm đích không được diễn giải. Hoạt động cuối cùng trong vòng lặp thưa thớt chuỗi là hoạt động sao chép. Chuỗi vòng lặp thưa thớt ban đầu, hoàn chỉnh, trong khi chính xác,

```
cho(i = 0; i < NR; i++){
      đối với (j = 0; j < NC; j++) { in
          ("%d,%d,%d", i, j, A (i, j))
      }
}
```

Liệt kê 4.1: Hạt nhân để in nội dung của một ma trận dày đặc

thường sẽ hoạt động kém. Nó có thể được chuyển đổi bằng cách sử dụng các hoạt động SPF tiêu chuẩn để cải thiện hiệu suất.

Phần này trình bày chi tiết từng thành phần bắt buộc: định dạng thưa thớt mô tả, thuật toán tổng hợp và các phép biến đổi thông thường.

4.1.1 Mô tả định dạng thưa thớt

Bộ mô tả định dạng thưa thớt chứa đủ thông tin để tạo và sử dụng định dạng thưa thớt cụ thể. Mỗi phần của mô tả định dạng được thể hiện bằng SPF ký hiệu. Các thành phần bao gồm một bản đồ từ không gian lặp thưa thớt đến dày đặc (một quan hệ), một bản đồ từ không gian lặp thưa thớt đến dữ liệu (một quan hệ), miền và phạm vi của mỗi hàm chưa được giải thích, và danh sách các lượng từ phổ quát mô tả thêm các chức năng chưa được giải thích được sử dụng trong bản đồ đầu tiên.

Bản đồ thưa thớt đến dày đặc. Một mối quan hệ thể hiện một hàm từ lặp thưa thớt không gian đến không gian lặp dày đặc. Bộ đầu vào của mối quan hệ là thưa thớt không gian lặp lại. Theo trực giác, bản đồ thưa thớt đến dày đặc có thể được suy ra từ một máy tính lặp lại qua các số không theo định dạng thưa thớt. Phép tính dưới đây lặp qua một ma trận dày đặc có kích thước NR × NC.

Không gian lặp của phép tính ma trận dày đặc như được hiển thị trong Liệt kê 4.1 được đưa ra theo bộ:

```
1 đối với (n = 0; n < NNZ; i++){ i = hàng
2     (n) j = cột (n)
3
4 in("%d,%d,%d",i,j,A(n)) 5 }
```

Liệt kê 4.2: Kernel để in tọa độ và giá trị của tất cả các số không theo định dạng COO

$$\{[i, j] : 0 \le i < NR \quad 0 \le j < NC\}$$

Phép tính 4.2 lặp lại qua ma trận COO và không gian lặp lại của nó là
được cho bởi tập hợp:

$$\{[n, i, j] : 0 \le n < NNZ \quad i = hàng(n) \quad j = cột(n)\}$$

Tương tự như vậy, phép tính 4.3 lặp lại qua ma trận CSR và phép lặp của nó
không gian được cho bởi tập hợp:

$$\{[i, k, j] : 0 \le i < NR \quad rowptr(i) \le k < rowptr(i + 1) \quad j = col(k)\}$$

Bản đồ từ thưa thớt đến dày đặc của COO và CSR được thể hiện trong Bảng 4.2 dựa trên cách
không gian lặp thưa thớt ánh xạ đến tọa độ dày đặc. Bản đồ thưa thớt đến dày đặc phải
là một hàm. Điều này được yêu cầu bởi tổng hợp thanh tra và chuyển đổi thực thi.

Mối quan hệ truy cập dữ liệu. Mối quan hệ truy cập dữ liệu, cũng được thể hiện dưới dạng SPF
mối quan hệ, ánh xạ từ không gian lặp thưa thớt đến không gian dữ liệu. Trong ví dụ của chúng tôi
sử dụng COO, mối quan hệ là {[n, i, j] [n]}. Không gian lặp của CSR là {[i, k, j]},
và mối quan hệ truy cập dữ liệu của nó là {[i, k, j] [k]}. Mối quan hệ truy cập dữ liệu tách rời

```
1 cho (i = 0; i < NR; i++){ 2 cho
(k = rowptr(i) ; k < rowptr(i+1); k++){ j = col(k) in("%d,%d,
3          %d",i,j,A(k))
4
5     }
6 }
```

Liệt kê 4.3: Hạt nhân để in nội dung của ma trận CSR

không gian lặp lại và không gian dữ liệu cho phép chúng được chuyển đổi riêng biệt.

Miền xác định và phạm vi. Miền xác định và phạm vi của mỗi hàm chưa được giải thích là bắt buộc. Trong ví dụ COO, phạm vi của mỗi mục là giống nhau. Lưu ý rằng Trong trường hợp này, định nghĩa miền và phạm vi sẽ đưa ra các hằng số ký hiệu bổ sung.

Các lượng từ phổ quát. Các lượng từ phổ quát tinh chỉnh thêm thông số kỹ thuật của định dạng thưa thớt. COO trong Bảng 4.2 không có lượng từ phổ quát trong khi MCOO giới thiệu một bộ lượng hóa phổ quát để đảm bảo rằng định dạng COO này được sắp xếp bằng cách sử dụng một lệnh Morton. Điều này đạt được bằng một hàm so sánh do người dùng xác định. Nó là điều quan trọng cần lưu ý là các hàm chỉ xuất hiện trong các lượng từ phổ quát là phải cung cấp định nghĩa đầy đủ và do người dùng định nghĩa.

4.1.2 Thuật toán tổng hợp

Tổng hợp mã đề cập đến việc tự động viết mã chuyển đổi dữ liệu từ một định dạng thưa thớt, nguồn, đến một định dạng khác, đích đến. Trong suốt phần này, chúng tôi tham khảo các ví dụ sử dụng COO làm nguồn. Quá trình này hoạt động bằng bất kỳ định dạng nào như nguồn. Tuy nhiên, hầu hết các tenxơ thưa thớt được lưu trữ trong COO và nó là dễ nhất định dạng để giải thích.

Đầu vào của thuật toán tổng hợp là hai mô tả định dạng thưa thớt và đầu ra là một biểu diễn SPF của thanh tra. Quá trình bắt đầu bằng cách soạn thảo

sự đảo ngược của bản đồ đích thưa thớt đến dày đặc với bản đồ nguồn thưa thớt đến dày đặc
bản đồ (xem định nghĩa soạn thảo trong [51]).

$$RAsrc \;\; Adest = (RAdest \;\; Adense)^{1} \;\; RAsrc \;\; Adense$$

Sử dụng mối quan hệ và các ràng buộc phổ quát, chúng ta giải quyết cho mỗi ẩn số
chức năng chưa được giải thích và tạo ra một biểu diễn SPF của mã tạo ra
những chức năng không được diễn giải đó. Mối quan hệ có được từ thành phần được sử dụng
để tạo mã sao chép dữ liệu. Dưới đây là tóm tắt về quá trình tổng hợp được thực hiện
bằng cách mô tả chi tiết từng bước.

1. Đảo ngược định dạng đích và chèn hàm hoán vị.

2. Soạn thảo các bản đồ từ thưa đến dày.

3. Đối với mỗi UF chưa biết, hãy tạo biểu diễn SPF để điền vào.

4. Đối với mỗi lượng từ q trong Universal Quantifiers, UQ tạo ra biểu diễn SPF
   để thực thi.

5. Tạo biểu diễn SPF cho thao tác sao chép.

Đảo ngược mối quan hệ định dạng đích và chèn Hoán vị. Định dạng
mô tả chỉ định một bản đồ từ không gian lặp thưa thớt đến không gian lặp dày đặc.
Đảo ngược mối quan hệ chuyển đổi các bộ dữ liệu đầu vào và đầu ra. Tiếp theo, chúng tôi giới thiệu một
chức năng tạm thời chưa được giải thích, được gọi là hoán vị, để đảm bảo nghịch đảo
bản đồ là các hàm: P([bộ đầu vào])_= [bộ đầu ra]. Các bộ đầu vào và đầu ra

là các bộ của định dạng đích ngược. Sau đây minh họa bước này
khi chuyển đổi sang MCOO.

$$RAMCOO\ \ Adense^{1} = \{[i, j] \quad [n2, ii, jj] \mid colm(n2) = jj$$
$$hàngm(n2) = ii \quad P(i, j) = [n2, ii, jj]$$
$$tôi = ii \quad j = jj\}$$

Soạn thảo. Các mối quan hệ được soạn thảo để thực hiện một ánh xạ duy nhất từ nguồn
đến các không gian lặp đích. Thành phần trong sự hiện diện của không được diễn giải
chức năng được hỗ trợ bởi IEGenLib [51]. Quá trình tổng hợp mã tập trung vào
bản đồ này.

$$RACOO\ \ AMCOO = RAMCOO\ \ Dày\ đặc^{1} \quad RACOO\ \ Adense$$
$$RACOO\ \ AMCOO = \{[n1, ii, jj] \quad [n2, ii, jj] \mid$$
$$jj = cột1(n1) \quad cột1(n1) = cột(n2)$$
$$ii = hàng1(n1) \quad hàng1(n1) = hàngm(n2)$$
$$P(hàng1(n1), cột1(n1)) = [n2, ii, jj]\}$$

Các hàm không được diễn giải chưa biết. Mối quan hệ xuất phát từ phép hợp thành
ở bước trước (trong ví dụ của chúng tôi
RACOO AMCOO) chứa một danh sách các ràng buộc. Các hàm không được giải thích (UF) từ
định dạng đích được cho là không xác định (UF không xác định). UF đã biết là
UF từ định dạng nguồn hoặc đã được giải quyết tại một thời điểm nào đó trong quá trình tổng hợp.
Chúng tôi giải quyết cho mỗi hàm chưa được giải thích và tổng hợp mã để
điền chúng (UF không xác định: rowm, colm, NNZ, P).

Một UF chưa biết được giải quyết bằng cách sử dụng mối quan hệ của nó với thông tin/chức năng đã biết (từ định dạng nguồn) trong bản đồ tổng hợp của chúng tôi. Lưu ý rằng NR và NC không xuất hiện trong danh sách này. Không thể suy ra hình dạng của ma trận một cách đáng tin cậy từ định dạng thưa thớt. Điều này là do các hàng hoặc cột ngoài cùng có thể là giá trị bằng không và ma trận sẽ trông nhỏ hơn thực tế. Do đó, chúng tôi yêu cầu các biến phải có sẵn để mô tả hình dạng của tenxơ.

Các ràng buộc liên quan đến mỗi UF chưa biết được xác định. Có có khả năng có nhiều hạn chế hơn trong danh sách này so với dự đoán ban đầu vì chúng ta phải sử dụng thay thế để tìm tất cả các ràng buộc. Danh sách các ràng buộc liên quan đến từng hàm chưa được giải thích trong ví dụ này được hiển thị trong Bảng 4.3.

Tổng hợp mã yêu cầu chúng ta xác định một câu lệnh để thực thi cùng với một không gian lặp lại và một lịch trình thực hiện. Có hai quyết định cần đưa ra tại thời điểm này. Đầu tiên, thứ tự nào để tạo ra các UF chưa biết và thứ hai, cái gì các câu lệnh và không gian lặp để tổng hợp.

Hãy xem xét dạng chung của mối quan hệ từ định dạng thưa thớt nguồn đến đích định dạng thưa thớt.

$$RA_{src} \; A_{dest} = \{ \; x \quad y \mid C\}$$

$$x \quad Z^{l}, \quad y \quad Z^{r}$$

Trong đó x là một bộ số nguyên có độ dài l, y là một bộ số nguyên có độ dài r và C là một danh sách ràng buộc. Trong các trường hợp dưới đây UF biểu thị cái chưa biết chưa được giải thích hàm và, f và f' đại diện cho các chức năng bao gồm các tổ hợp tuyến tính của các giá trị đã biết các hàm không được diễn giải và các hằng số biểu tượng. u và v là các bộ số nguyên các tập con của x và y tương ứng.

Các ràng buộc từ mối quan hệ kết quả RAsrc Adest được nhóm thành 5 trường hợp. Các trường hợp 1-3 bên dưới đề cập đến các ràng buộc chỉ sử dụng các biến tuple từ tuple đầu vào. Trường hợp 4 và 5 xử lý các ràng buộc liên quan đến cả bộ dữ liệu đầu vào và đầu ra biến, nhưng thứ tự bị hoán đổi. Có những tổ hợp toán tử bổ sung và các đặc điểm quan hệ không được xem xét. Có thể là họ sẽ cần được thêm vào nếu chúng được tìm thấy tồn tại trong các định dạng tenxơ thưa thớt. Tuy nhiên, tại thời điểm này, chúng tôi chỉ thêm các trường hợp cho những kết hợp có trong định dạng hiện tại.

Trường hợp 1 Ràng buộc: UF(u) = f(u)

Phát biểu: UF(u) = f(u)

Miền: { u : C}, trong đó C là danh sách ràng buộc từ RAsrc Adest sau khi chiếu.

Trường hợp 1 bao gồm ràng buộc bằng nhau và cả vế trái và vế phải lấy cùng một bộ (u) là một tập hợp con của các biến bộ cho bộ đầu vào của mối quan hệ ban đầu. Câu lệnh tương ứng là câu lệnh gán. Không gian lặp lại cho câu lệnh đó được tạo ra bằng cách chiếu ra tất cả các biến tuple từ mối quan hệ ban đầu không phải là thành viên của u. Không có ràng buộc nào trong Ví dụ đang chạy được phân loại là trường hợp 1.

Trường hợp 2 Ràng buộc: UF(f ′ (u)) ≤ f(u)

Phát biểu: UF(u) = min(UF(u), f(u))

Miền: { u : C}, trong đó C là danh sách ràng buộc từ RAsrc Adest sau khi chiếu.

Trường hợp 3 Ràng buộc: UF(u) ≥ f(u)

Phát biểu: UF(u) = max(UF(u), f(u))

Miền: { u : C}, trong đó C là danh sách ràng buộc từ

RAsrc Adest sau khi chiếu.

Trường hợp 2 và 3 là các ràng buộc bất đẳng thức trong đó cả vế trái và vế phải đều sử dụng u là tập hợp con của các biến tuple đầu vào của mối quan hệ ban đầu. Không xác định UF trong trường hợp 2 có giới hạn trên là f(u), được dịch thành một phép gán cho giá trị nhỏ nhất của f(u). UF chưa biết trong trường hợp 3 có giới hạn dưới của f(u), trong đó dịch thành một phép gán tới giá trị cực đại của f(u). Không gian lặp của cả hai trường hợp 2 và 3 được tạo ra bằng cách chiếu ra tất cả các biến tuple từ mối quan hệ ban đầu không phải là thành viên của u. Cho rằng f(u) là tổ hợp tuyến tính của các phần tử đã biết khác UF, đối với mỗi trường hợp tuple u, cần phải có giá trị min hoặc max để thỏa mãn bất đẳng thức trong trường hợp 2 và 3. Không có ràng buộc nào trong ví dụ này được phân loại như trường hợp 2 hoặc 3.

Trường hợp 4 Ràng buộc: UF(u) = f(v)

Câu lệnh: UF.insert(F(u))

Miền: { u : C}, trong đó C là danh sách ràng buộc từ

RAsrc Adest sau khi chiếu.

Trường hợp 4 là ràng buộc bằng nhau trong đó vectơ u, được sử dụng ở vế trái, là tập hợp con của bộ dữ liệu đầu vào và vectơ v, được sử dụng ở phía bên phải, là tập hợp con của bộ đầu ra. Câu lệnh tổng hợp là lệnh gọi chèn lấy hàm f(v) như một tham số. Vectơ v trong UF phân biệt Trường hợp 4 với Trường hợp 1. Trong Ví dụ, các ràng buộc trên rowm, colm và P là trường hợp 4.

Không phải tất cả các ràng buộc đủ điều kiện là trường hợp 4 trong ví dụ đang chạy đều có thể được thỏa mãn ngay lập tức. Các mối quan hệ cho các ràng buộc trên rowm và colm không

```
1 P = new OrderedList(2,1,MORTON(),"<"); 2 cho(int
c0=0;c0<NNZ;c0++){ 3
P.insert(row1(c0),col1(c0)); 4 }
```

Liệt kê 4.4: Hàm sắp xếp lại hoán vị P

chức năng. Được thực hiện với các ràng buộc chung, các mối quan hệ cho P là một hàm và

nên được xử lý trước.

{[u] v]|Danh sách}

Mối quan hệ kết quả cho P như sau.

{[ii, jj] [n2, ii, jj]|}

Không có ràng buộc đủ điều kiện. Tuy nhiên, khi chúng ta cũng xem xét tính phổ quát

lượng tử có đủ thông tin để tạo ra một ánh xạ chính xác. Mã mà

sẽ được tạo ra từ SPF-IR đại diện cho mã tổng hợp tạo ra một lớp

điều đó sẽ thực thi phép lượng hóa phổ quát.

Danh sách 4.4 hiển thị các tham số của trình xây dựng danh sách là số lượng đầu vào,

số lượng đầu ra, chức năng sử dụng làm bộ so sánh và hoạt động mong muốn (ít hơn

lớn hơn hoặc lớn hơn). Điều quan trọng cần lưu ý là không yêu cầu phải có phép ánh xạ chính xác.

Nếu chuyển đổi sang định dạng chưa được sắp xếp, thứ tự tùy ý sẽ được sử dụng (

thứ tự chèn). Ánh xạ cụ thể nhất được tìm thấy là ánh xạ được chọn cho

tổng hợp.

Trường hợp 5 Ràng buộc: UF(v) = f(u)

Câu lệnh: UF.insert(F(u))

Miền: { u : C}, trong đó C là danh sách ràng buộc từ

RAsrc Adest sau khi chiếu.

Trường hợp 5 bao gồm các ràng buộc về mặt bằng trong đó vectơ v được sử dụng bởi vế trái là một tập hợp con của bộ dữ liệu đầu vào và vectơ bên phải u là một tập hợp con của đầu ra tuple. Sự khác biệt duy nhất giữa trường hợp 4 và 5 là việc sử dụng v và u trên hai mặt đối lập của câu phát biểu.

Một ví dụ về trường hợp này là ràng buộc tắt như được thấy trong DIA (xem Bảng 4.1). Giả sử DIA là tenxơ đích và off cần được giải quyết cho: giải quyết cho off trong ràng buộc sẽ dẫn đến off(d) = j i. Các biến tuple j và i được biết đến nhưng tuple biến d không phải là tổ hợp tuyến tính của các biến tuple đã biết và bị giới hạn bởi ND mà cũng không được biết đến. Trong sự trừu tượng chèn, bất kỳ ràng buộc nào có mặt như một lượng từ phổ quát trên UF trong mô tả được thực thi. Trong ví dụ này, (j i) được chèn vào off và ràng buộc e1, e2 : e1 < e2 off(e1) < off(e2) được thực thi.

Thứ tự xử lý các ràng buộc được xác định bởi tính khả dụng của thông tin và RHS của ràng buộc và các phẩm chất của mối quan hệ. Tất cả UF ở RHS phải được biết đến từ định dạng nguồn hoặc đã được biết đến trước đó đã xử lý. Ví dụ đang chạy của chúng tôi có 4 UF chưa biết: rowm, colm, NNZ và P. P được xử lý đầu tiên, sau P, cả rowm và colm đều có các mối quan hệ là các hàm và có thể được xử lý theo bất kỳ thứ tự nào. NNZ có thể được xử lý sau rowm hoặc colm . việc triển khai ngây thơ sẽ là trường hợp 2. Tuy nhiên, hợp nhất vòng lặp và loại bỏ mã chết làm cho nó thành một bài tập đơn giản.

Thực thi các lượng từ phổ quát. Bất kỳ lượng từ phổ quát nào có trong des- định dạng tination được thực thi. Có hai loại lượng từ phổ quát trên một un- hàm được diễn giải: một lượng từ sắp xếp lại và một lượng từ đơn điệu. Sắp xếp lại

các lượng từ phổ quát dẫn đến một ràng buộc về thứ tự được đặt trên toàn bộ đích đến
tenxơ, trong khi một lượng tử đơn điệu chỉ áp dụng cho hàm chưa được giải thích
đang được mô tả. Ví dụ Morton dưới đây là một ví dụ về lượng từ sắp xếp lại.

```
phạm vi(hàngm) = {0 <= i < NR}
miền(rowm) = {0 <= x < NNZ}
    n1, n2 : n1 < n2     MORT ON(rown(n1), coln(n1))
                <MORT ON(hàng(n2), cột(n2))
```

Ở đây, ràng buộc trên n1, n2 có tác dụng phụ lên thứ tự của định dạng. Một đơn điệu
mặt khác, lượng từ là cục bộ đối với hàm chưa được giải thích và không có
bất kỳ tác động nào đến thứ tự của tenxơ. Một ví dụ sẽ là rowptr trong nén
định dạng hàng thưa (CSR) - xem Hình 2.6.

```
phạm vi(rowptr) = {0 <= x <= NNZ}
miền(rowptr) = {0 <= i <= NR}
    e1, e2 : e1 < e2     rowptr(e1) <= rowptr(e2)
```

Trong cả hai trường hợp, tổng hợp sẽ đảm bảo những ràng buộc này được thỏa mãn. Trong trường hợp
sắp xếp lại các lượng từ, các ràng buộc sẽ được thực thi như các ràng buộc sắp xếp.
tính tổng quát của những ràng buộc này cho phép các hàm do người dùng xác định trong các thông số định dạng
và đây là đóng góp độc đáo của chúng tôi. Trong hầu hết các trường hợp, mã được tổng hợp trong
giai đoạn này không cần thiết để đảm bảo tính chính xác và có thể được loại bỏ trong quá trình tối ưu hóa.

Tạo bản sao dữ liệu. Bước cuối cùng trong quá trình tổng hợp là mã “sao chép”. Tại
điểm này, tất cả các chức năng chưa được giải thích trong định dạng đích đã được thực hiện thành công

được tổng hợp như đặc tả SPF. Mã sao chép sao chép dữ liệu từ nguồn sang đích đến. Miền của mã sao chép là mối quan hệ được tạo thành như một tập hợp. câu lệnh là một câu lệnh sao chép và lệnh đọc và lệnh ghi là nguồn và đích truy cập dữ liệu tương ứng.

4.1.3 Ví dụ phức tạp hơn: COO đến CSR

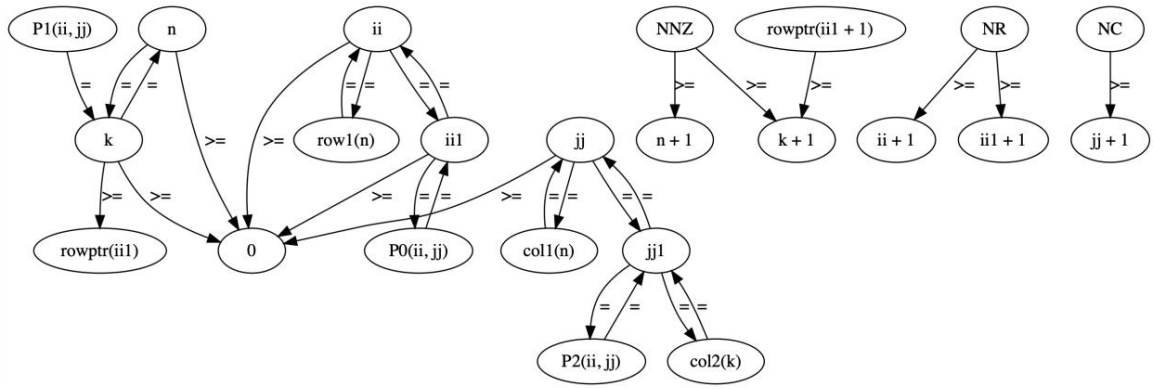
Ví dụ về COO với Morton COO không phát huy hết sức mạnh của sự đồng bộ thuật toán luận án. Phần này cung cấp tổng quan về tổng hợp chuyển đổi mã từ COO đến CSR. Ví dụ phức tạp hơn này chứng minh tầm quan trọng của các lượng từ phổ quát cho các trường hợp vượt ra ngoài thứ tự.

Chèn hoán vị & Soạn. Bước đầu tiên thêm hoán vị con- căng thẳng trên hàm nghịch đảo của

RACSR Adense đảm bảo rằng đó là một hàm và sau đó thực hiện thao tác soạn thảo như được hiển thị dưới.

$$\text{RACSR Adense}^{1} = \{[i, j] \quad [ii, k, jj] \mid ii = i \quad col2(k) = j \quad j = jj \quad 0 \le ii < NR \quad rowptr(ii) <= k \quad k < rowptr(ii + 1) \quad P(i, j) = (ii, k, jj)\}$$

Ràng buộc hoán vị P(i, j) = (ii, k, jj) có thể được đơn giản hóa hơn nữa thành một hoán vị chiều P0(i, j) = ii, P2(i, j) = jj, và P1(i, j) = k.



Hình 4.1: Đồ thị ràng buộc thu được từ mối quan hệ hợp thành RACOO ACSR .

$$RACSR\ Adense^{1} = \{[i, j]\ \ [ii, k, jj] | ii = i\ \ col2(k) = j$$

$$0 \le ii < NR\ \ rowptr(ii) <= k$$

$$k < hàngptr(ii + 1)\ \ P0(i, j) = i$$

$$P1(i, j) = k\}$$

$$RACOO\ ACSR = RACSR\ Adense^{1}\ \ RACOO\ ACSR$$

$$RACSR\ ACSR = \{[n, ii, jj]\ \ [ii1, k, jj1] | ii = hàng1(n)$$

$$jj1 = col2(k)\ \ jj = col1(n)\ \ 0 \le ii < NR$$

$$rowptr(ii1) <= k\ \ k < rowptr(ii1 + 1)$$

$$P0(ii, jj) = ii1\ \ P0(ii, jj) = jj1\ \ jj = jj1$$

$$P1(ii, jj) = k\ \ ii = ii1\}$$

Ràng buộc trên từ thành phần có thể được biểu diễn dưới dạng đồ thị ràng buộc như sau được hiển thị trong Hình 4.1. Biểu đồ ràng buộc được sử dụng nội bộ để biểu diễn các ràng buộc như các đỉnh và các cạnh. Một đỉnh là một thuật ngữ trong ràng buộc và một cạnh biểu diễn một mối quan hệ giữa hai cạnh.

Áp dụng thuật toán đường đi ngắn nhất của Floyd Warshall trên mọi đỉnh trong con-

đồ thị biến dạng giới thiệu một hành vi đóng cửa bắc cầu trên đồ thị. Một sửa đổi nhỏ
sự hóa thuật toán Floyd-Warshal như được thể hiện trong Thuật toán 1 được sử dụng để thực hiện
đóng trên đồ thị ràng buộc.

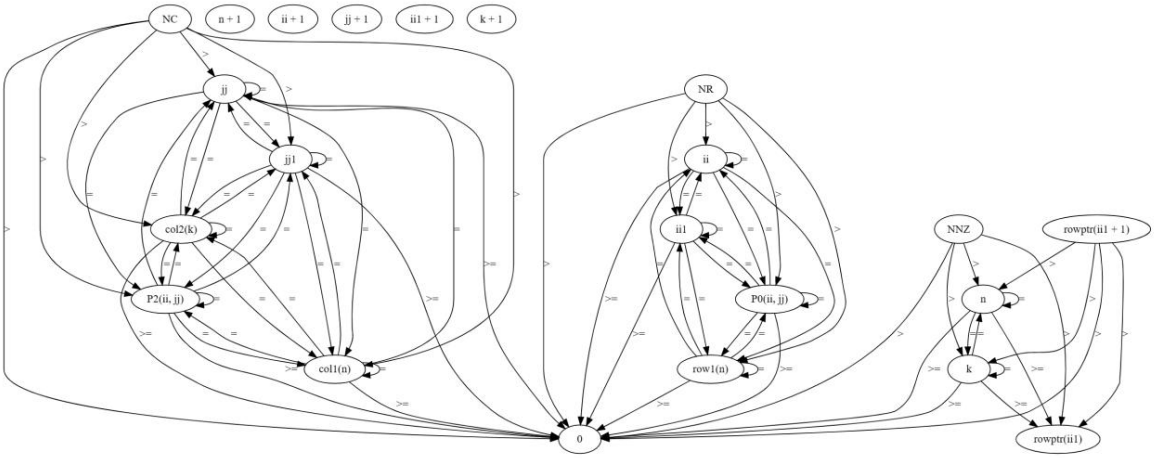
Thuật toán 1 Sửa đổi của Floyd-Warshall cho đóng cửa chuyển tiếp

```
Yêu cầu: E : (c, v1, v2) : c    {>, <, ≤, ≥, =}    vi    V thủ
  tục Closure(G(V, E)) cho k từ 1 đến                                    Biểu đồ ràng buộc
      V thực hiện
          đối với i từ 1 đến V làm
              đối với k từ j đến V thì làm
                  cost[i][j]    GetStrongConstraint(cost[i][k],cost[k,i],cost[i][j]) kết thúc
              cho
          kết thúc cho
      kết thúc cho
```

Trong Thuật toán 1, chi phí là một ma trận kề biểu diễn đồ thị ràng buộc,
GetStrongConstraint là một hàm lấy vào một danh sách các ràng buộc tùy ý và
trả về ràng buộc mạnh nhất. Nó gán một số cho phép bằng (=), bất đẳng thức mạnh
(>, <) và bất đẳng thức yếu (≥, ≤) sao cho đẳng thức > bất đẳng thức mạnh >
bất bình đẳng yếu. Hình 4.2 cho thấy đồ thị ràng buộc mới với một
mối quan hệ hạn chế để hỗ trợ tổng hợp.

Một bước quan trọng trong COO đến CSR là thuật toán nhận ra một khái niệm phổ quát tương tự
bộ lượng hóa bao gồm các cặp n và k từ định dạng nguồn và đích tương ứng.
Một lượng từ phổ quát trên một hàm không được diễn giải sẽ ảnh hưởng đến thứ tự tổng thể của
một tenxơ nếu và chỉ nếu bộ dữ liệu liên quan đến một biến bộ dữ liệu không gian.

Mối quan hệ truy cập dữ liệu cho COO và CSR như thể hiện trong bảng 4.2 và 4.1 là
được đưa ra dưới đây:



Hình 4.2: Đồ thị ràng buộc sau phép đóng trên quan hệ hợp thành RACOO ACSR .

DICO ACOO = {[n, ii, jj] [n]}

DICSR ACSR = {[ii, k, jj] [k]}

COO được sắp xếp theo hàng trước bằng cách sử dụng ràng buộc chung bên dưới.

n1, n2 : n1 < n2 hàng(n1) <= hàng(n2)

CSR cũng được sắp xếp theo mặc định như sau:

k1, k2 : k1 < k2 dim0(k1) <= dim0(k2)

CSR không có chức năng chưa được giải thích cho hàng, nhưng thuật toán tổng hợp biết rằng dim0 là thông tin tọa độ dày đặc trên chiều thứ nhất. Do sự giống nhau trong cả hai thuộc tính mảng mô tả trực tiếp một bộ không gian dữ liệu, sắp xếp lại hàm P1(i, j) = k có thể được loại bỏ khỏi mối quan hệ và n = k được đưa vào trực tiếp.

Các ràng buộc liên quan đến các hàm hoán vị có thể giải quyết được như một hàm của đầu vào tuple cũng bị loại bỏ, điều này áp dụng cho P0(i, j) = ii và P2(i, j) = jj. Biểu đồ ràng buộc 4.2 giới thiệu một số lượng lớn các ứng viên cho tổng hợp có thể có nghĩa tương tự.

Sau đó, thuật toán sẽ chọn từng UF chưa biết và tìm kiếm các ứng viên ràng buộc liên quan đến UF như vậy. Bắt đầu với rowptr các ứng cử viên để xem xét tổng hợp và các trường hợp tương ứng được thể hiện dưới đây:

```
ứng viên 1: rowptr(ii1) <= n, Trường hợp: 2
ứng viên 2: rowptr(ii1) <= k, Trường hợp: 2
ứng viên 3: rowptr(ii1 + 1) >= 1, Trường hợp: 3
ứng viên 4: rowptr(ii1 + 1) >= n + 1, Trường hợp: 3
ứng viên 5: rowptr(ii1 + 1) >= k + 1, Trường hợp: 2
ứng viên 6: rowptr(ii1) < NNZ, Trường hợp: 2
ứng viên 7: -rowptr(ii1) + rowptr(ii1 + 1) - 1 >= 0, Trường hợp: không có
```

Mục cuối cùng trong danh sách ở trên không rơi vào trường hợp nào. Tổng hợp Thuật toán tạo mã cho tất cả các câu lệnh trên. Trong mô tả CSR như được thấy trong Bảng 4.1, rowptr có một lượng từ phổ quát như được hiển thị bên dưới:

```
ii1, ii2 : ii1 < ii2     rowptr(ii1) <= rowptr(ii2)
```

Bộ định lượng này không liên quan đến một bộ được sử dụng trong truy cập dữ liệu nên việc thực thi điều này quantifier là cục bộ đối với rowptr UF. Câu lệnh được tạo cho mã này kiểm tra xem với mọi ii1 < ii2 đối với điều kiện không được đáp ứng, hãy tạo ít nhất điều kiện bằng nhau để đáp ứng được điều này có thể được tạo ra trong mã như sau:

```
nếu ( không (rowptr(ii1) <= rowptr(ii2))){
```

```
        rowptr(ii2) = rowptr(ii1);
    }
```

Tập xác định của phát biểu trên thỏa mãn ii1 < ii2 và cũng hợp lệ như
miền của rowptr (xem Bảng 4.1 được đưa ra trong Phương trình 4.1.

```
{[ii1, ii2] : 0 <= ii1 < NR + 1    0 <= ii2 < NR + 1    ii1 < ii2}                    (4.1)
```

Phương trình 4.2 cũng có nghĩa tương tự và ít tốn kém hơn về mặt tính toán
so với Phương trình 4.1. Trong quá trình thực hiện, chúng tôi thích phương án sau hơn phương án trước.

```
{[ii1, ii2] : 0 <= ii1 < NR + 1    ii2 = ii1 + 1}                    (4.2)
```

Cuối cùng, chúng tôi chọn các ràng buộc từ đồ thị 4.2 liên quan đến col2 của CSR như được hiển thị
trong danh sách dưới đây

ứng viên 1: col2(n) = jj, Trường hợp: 1

ứng viên 2: col2(k) = jj, Trường hợp: 1

ứng viên 3: col2(k) = jj1, Trường hợp: 1

ứng viên 4: col2(n) = col1(n) , Trường hợp: 1

ứng viên 5: col2(k) = col1(n) , Trường hợp: 1

ứng viên 6: col2(n) >= 0, Trường hợp: 3

ứng viên 7: col2(k) >= 0, Trường hợp: 3

ứng viên 8: NC > col2(n), Trường hợp: 2

ứng viên 9: NC > col2(k), Trường hợp: 2

Cả chín ứng viên trong danh sách đều hợp lệ để tổng hợp mã, tuy nhiên, một trường hợp
đủ để tạo mã. Chúng tôi thích ứng viên trường hợp 1 hơn các trường hợp khác vì
nó dẫn đến một ràng buộc bình đẳng bao phủ hoàn toàn hàm chưa được giải thích. Đối với
Ví dụ, chọn ứng viên 1 sẽ dẫn đến không gian lặp được hiển thị trong Phương trình 4.3

{[n, ii, jj] : 0 <= n < NNZ ii = hàng(n) jj = cột(n)} (4.3)

Danh sách các ứng viên được chọn cho mỗi chức năng chưa được giải thích được gán một giá trị duy nhất
lịch trình thực hiện. Thuật toán tổng hợp cũng hoạt động như một giải pháp điểm cố định.
Thuật toán chọn một UF chưa biết và cố gắng áp dụng đã thảo luận trước đó
các hoạt động đối với nó, nếu trong trường hợp không có giải pháp hợp lệ hoặc ứng cử viên nào rơi vào
trong mọi trường hợp, UF được đặt ở cuối hàng đợi và các UF khác sẽ được giải quyết.
Khi thuật toán đã đạt đến điểm mà không có gì được giải quyết và các mục vẫn còn
vẫn còn trong hàng đợi, chương trình sẽ kết thúc sớm với lỗi chỉ ra
chức năng chưa được giải thích không thể giải quyết được. Mô tả đầy đủ các phép tính
trong SPF IR cũng yêu cầu quyền truy cập đọc và ghi, điều này được tính toán trong tất cả các mục đã chọn
ứng viên bằng phân tích tĩnh. Hình 4.6 cho thấy mã được tạo ra từ SPF IR
biểu diễn được tạo ra bởi thuật toán tổng hợp.

4.1.4 Chuyển đổi SPF để tối ưu hóa

Biểu diễn SPF ban đầu có thể chứa mã thừa hoặc không cần thiết và
sử dụng các cấu trúc vòng lặp không lý tưởng cho hiệu suất. Chúng tôi sử dụng một bộ sưu tập
của các chuyển đổi SPF tiêu chuẩn để cải thiện hiệu suất.

Trong quá trình tổng hợp, chúng tôi chọn ra những ràng buộc là ứng cử viên khả thi cho
tổng hợp các câu lệnh cho một UF chưa biết. Tuy nhiên, điều này có thể tạo ra nhiều

các câu lệnh thực hiện cùng một việc. Nếu nhiều câu lệnh bao phủ cùng một không gian dữ liệu chúng tôi loại bỏ tất cả trừ một cái.

Có những tình huống mà hoán vị P không cần thiết để đảm bảo tính chính xác của mã. Trường hợp này được phát hiện bằng cách loại bỏ mã chết tiêu chuẩn. SPF, ở mức cơ bản nhất, là một biểu đồ luồng dữ liệu. Biểu đồ đó được duyệt ngược lại, bắt đầu bằng live-out không gian dữ liệu. Bất kỳ không gian dữ liệu và tính toán tương ứng nào không được truy cập là LOẠI BỎ.

Fusion kết hợp hai vòng thành một vòng. Fusion giảm đọc nhằm mục đích kết hợp các câu lệnh đọc từ cùng một vị trí trong bộ nhớ để giảm dung lượng bộ nhớ. Công trình trước đây [41] cho thấy sự hỗ trợ cho sự hợp nhất trong SPF. Nhiều lần đọc trên cùng một dữ liệu vị trí xảy ra trong quá trình tổng hợp như một loạt các chuỗi vòng.

Sự kết hợp giữa nhà sản xuất và người tiêu dùng kết hợp các chuỗi vòng lặp ghi và sau đó đọc từ cùng một dữ liệu. Điều này thường dẫn đến việc giảm không gian cần thiết cho dữ liệu tạm thời lưu trữ. Tất cả các cơ hội để áp dụng giảm đọc và hợp nhất nhà sản xuất-người tiêu dùng đều được áp dụng.

4.2 Thực hiện

Việc triển khai của chúng tôi sử dụng IEGenLib [51] để thao tác tập hợp thưa thớt và quan hệ. Thuật toán tổng hợp tạo ra SPF IR bằng cách sử dụng SPF API [41] hỗ trợ tạo mã với quét đa diện. Trong phần này chúng tôi mô tả việc triển khai chi tiết, độ phức tạp của thuật toán và cấu trúc dữ liệu được sử dụng trong công việc của chúng tôi.

4.2.1 Thuật toán chi tiết

Trong phần này, chúng tôi sẽ tập trung vào các chi tiết triển khai liên quan đến thuật toán tổng hợp và cơ chế hoạt động của nó. Thuật toán 2 cho thấy chi tiết hơn thực hiện thuật toán tổng hợp.

Phép hoán vị AddP thêm một ràng buộc hoán vị UF P([in]) = [out] vào tham số quan hệ. Thủ tục RemoveP ermuteEqualT oResolvableT uples re- di chuyển tất cả các hàm hoán vị không cần thiết có thể tạo ra mã trùng lặp— các hàm bằng với các bộ đầu ra có thể giải quyết được. Một bộ đầu ra được gọi là có thể giải quyết được nếu nó có thể được biểu thị như một hàm của các biến tuple đầu vào và/hoặc đã biết các hàm không được diễn giải. Hoạt động soạn thảo đã được định nghĩa trước đó trong Phần Định lý 2. Quy trình BuildConstrGraph xây dựng một đồ thị ràng buộc từ những ràng buộc của một mối quan hệ, một ví dụ về đồ thị này được thấy trong Hình 4.1. chức năng Closure áp dụng thuật toán Floyd-Warshall như được mô tả trong Thuật toán 1, điều này khám phá nhiều ứng viên có thể hơn cho tổng hợp. Quy trình GetConstraints lấy một đồ thị ràng buộc và trả về một danh sách chứa tất cả các cạnh trong đồ thị như ràng buộc. Tất cả các hằng số tượng trưng và các hàm chưa được giải thích đều được tính toán bằng cách sử dụng thủ tục GetSymbols trong thuật toán. Điều này quan trọng để điền vào các và các UF/hằng số tượng trưng chưa biết; rất quan trọng đối với thuật toán tổng hợp.

Giải quyết cho một UF trong một phương trình/ràng buộc là quan trọng đối với tổng hợp. Điều này đảm bảo rằng UF chưa biết được giải quyết và nằm trong vế trái của phương trình. solveF hoặcUF giải quyết cho một uf trong ràng buộc được cung cấp. Các trường hợp đã thảo luận trước đó trong Phần 4.1.2 được tính toán bằng thuật toán với hàm GetSynthCase. Hàm này sử dụng tất cả các điều kiện được liệt kê để phân loại một ràng buộc như rơi vào trường hợp 1 đến 5 và một trường hợp đặc biệt của không xác định. Các trường hợp không xác định sẽ

trả về false cho một alid IsV trong khi các trường hợp từ 1 đến 5 sẽ trả về true. Quy trình GetCaseDomain sử dụng projectOut (xem Phần Bối cảnh 2) để loại bỏ các bộ không phải là một phần của ràng buộc c. Đây là một cách hiệu quả để tạo ra một miền/không gian bao phủ ràng buộc c từ mối quan hệ Rsrc dest. Thủ tục GetCaseStatement tạo ra một câu lệnh dựa trên trường hợp tổng hợp được thảo luận trong Mục 4.1.2. Các lệnh đọc và ghi được thực hiện bằng cách sử dụng các thủ tục GetReads và GetWrites tương ứng. Điều này được tính toán bằng cách kiểm tra LHS để truy cập ghi và RHS cho các truy cập đọc trong câu lệnh được tạo ra. ID thực thi giữ nguyên theo dõi lịch trình thực hiện của từng thành phần được tạo thành công của thuật toán. Các hàm không được diễn giải trong định dạng nguồn và đích được mô tả sử dụng các thuộc tính miền, phạm vi và mảng. Thủ tục GetUF Thuộc tính tóm tắt lời kêu gọi truy xuất thông tin về UF. Nếu UF có thuộc tính mảng và miền của nó không trải dài trên một không gian dữ liệu, chúng tôi tạo mã để thực thi mảng đó thuộc tính như được thể hiện trong Thuật toán 3.

Thực thi thủ tục thuộc tính mảng được trình bày chi tiết trong Thuật toán 3. Các thành phần của Thuộc tính ufP bao gồm các thuộc tính miền, phạm vi và mảng. Hình 4.3 cho thấy các thành phần của một thuộc tính mảng được sử dụng trong đặc tả thuật toán 3. chức năng BuildQuantDomain xây dựng một tập hợp bao phủ miền để thực thi một thuộc tính mảng. Quy trình sử dụng vị từ LHS như một ràng buộc trong tập hợp, lượng hóa như khai báo tuple, và tất cả các biến tuple đều bị hạn chế bởi miền của hàm. BuildQuantDomain xây dựng một tập hợp được hiển thị trong Phương trình 4.4 cho một thuộc tính mảng 4.3 và miền giá trị của rowptr trong Bảng 4.1.

$$\{[ii1, ii2] : 0 <= ii1 <= NR \quad 0 <= ii2 <= NR \quad ii1 < ii2\} \tag{4.4}$$

Hình 4.3: Các thành phần của một thuộc tính mảng UF

Quy trình GetM inimalStmt trả về một câu lệnh tối thiểu cần thiết cho
Vị ngữ RHS của thuộc tính mảng là đúng. Vị ngữ RHS như được hiển thị trong Hình 4.3
tổng hợp một câu lệnh rowptr(ii2) = rowptr(ii1). Câu lệnh được tạo ra bởi
bước trong thuật toán cho ví dụ rowptr được đưa ra trong danh sách bên dưới.

```
for(ii1 = 0; ii1 <= NR; ii1++){
      đối với (ii2 = ii1+1; ii2 <= NR; ii2++){
            nếu (không (rowptr(ii1) <= rowptr(ii2))){
                  rowptr(ii2) = rowptr(ii1)
            }
      }
}
```

Tạo thủ tục mã sao chép tạo ra một bản tính toán sao chép cuối cùng từ nguồn đến
đích đến sau khi tất cả các hàm chưa được biên dịch đã được tổng hợp thành công.

4.2.2 Cấu trúc dữ liệu tập hợp có thứ tự và không có thứ tự

Sự trừu tượng trong Hình 4.7 cho thấy cấu trúc dữ liệu được sử dụng trong các chương trình không gian đặc biệt
các hàm được diễn giải. Các hàm không được diễn giải đặc biệt rơi vào trường hợp 5 và 4.
trừu tượng bao gồm chèn, so sánh, sắp xếp và lấy. Chèn thêm các bộ vào
trừu tượng, bộ so sánh là một hàm sắp xếp lại được tạo ra trong thời gian biên dịch, sắp xếp
sắp xếp lại tuple được chèn dựa trên bộ so sánh và get trả về tuple cho

Thuật toán 2 Tổng hợp Thủ tục thuật

```
toán Tổng hợp(srcDesc, destDesc) spf ir
    Rsrc_ dense
      srcDesc.SparseT oDenseM ap Rdest  dense
    destDesc.SparseT oDenseM ap Rdense  dest
    Rdest  dense AddP                   1
    ermutation(Rdense  dest)
    RemoveP ermuteEqualT oResolvableT uples(Rdense  dest)
    Rsrc  đích   Rdense  đích   Rsrc  dense
    G(V, E)   BuildConstrGraph(Rsrc  đích)
    GT(V, E)   Đóng(G(V, E))
    C   GetConstraints(GT(V, E))
    unknownUF s   GetSymbols(Rdense  dest) knownUF
    s   GetSymbols(Rsource  dense) execution id
      0 while unknownUF
    s is not empty do uf   unknownUfs.dequeue()
        solvedF or   f alse for c   C
        do if uf   c then eq
           SolveF
            orUF(c, uf) case
                  GetSynthCase(eq,
               Rsrc  dest) if IsV alid(case) then

                    miền   GetCaseDomain(c, case, Rsrc  dest) câu lệnh
                    GetCaseStatement(c, case, Rsrc  dest) đọc   GetReads(c, case,
                    Rsrc  dest) ghi   GetW rites(c, case,
                    Rsrc  dest) lịch thực thi   GetSchedule(id thực
                    thi) spf ir.add((câu lệnh, miền, đọc, ghi, lịch thực thi))
                    id thực thi   id thực thi + 1 solvedF hoặc   true kết thúc nếu kết thúc nếu kết
                    thúc cho nếu solvedF hoặc là false thì

            unknownUfs.enqueue(uf) khác            Đặt lại vào hàng đợi nếu chưa giải quyết được

            knownUF s.push(uf)
            ufQuantif ier   GetUF Quantif ier(uf) nếu
            ufQuantif ier /=   thì
                EnforceArrayP roperty(spf ir, ufQuantif ier, uf) kết thúc nếu kết
            thúc nếu
        kết thúc
    trong khi
    GenerateCopyCode(spf ir, srcDesc, destDesc, ID thực thi) trả về spf ir
kết thúc thủ tục=0
```

Thuật toán 3 Thực thi thủ tục thuộc tính mảng

```
thủ tục EnforceArrayProperty(spf ir, ufQuantif- ier, uf, execution id) ufdomain-
    ufProperty.domain ufProperty
    ufQuantif ier.arrayP roperty q
    ufProperty.quantification
    lhsPred    GetLHSP redicate(ufP roperty.predicate)
    rhsPred    GetRHSP redicate(ufP roperty.predicate)
    domain    BuildQuantDomain(uf domain, q, lhsP red) ” if
    tuyên bố          (not (”+rhsP red+”)) {”+GetM inimalStmt(rhsP red)+”}” đọc
    GetReads(statement) ghi
    GetWrites(statement) spf
    ir.add((statement, domain, reads, writes, execution sched-)) kết
thúc thủ tục
```

Thuật toán 4 Tạo thủ tục thuật toán mã sao

```
chép GenerateCopyCode(spf ir, srcDesc, destDesc, execution id)          -
    DIsrc  Asrc    sourceDesc.DataAccess
    DIdest  Adest    destDesc.DataAccess
    copyStatement    GetCopyStatement(DIsrc  Asrc , DIdest  Adest )
    copyDomain    T oSet(Rsrc  dest)
    lịch thực hiện    GetSchedule(id thực hiện) spf
    ir.add((statement, domain, reads, writes, execution sched-)) kết
thúc thủ tục
```

```
1      // tắt(d)
2 #define off(d) off->get_inv(d)
3 Hoán vị < forall e1,e2: e1<e2 <=> tắt(e1)< tắt(e2)> tắt = mới
        Permute(); 4 //
tắt chèn (j - i) được sử dụng bên trong một số miền 5 tắt->insert( {j - i});
```

Hình 4.4: Một ví dụ về mã được tạo cho Trường hợp 4 với độ lệch trong DIA

một vị trí nhất định trong trừu tượng. Xem xét ràng buộc off(d) = j i, với ràng buộc e1, e2 : e1 < e2 mã off(e1) < off(e2) được tạo ra được hiển thị trong Hình 4.4 để chèn các câu lệnh và truy cập vào các hàm chưa được biên dịch.

4.2.3 Sắp xếp lại luồng trên hoán vị

Lặp lại qua định dạng nguồn và truy xuất các vị trí mới bằng cách sử dụng get chức năng trong Hình 4.7 là tốn kém. Trong quá trình triển khai, chúng tôi giới thiệu một dữ liệu cấu trúc khám phá luồng. Kỹ thuật này lặp lại thông qua chức năng sắp xếp lại thay vì định dạng nguồn. Theo cách này, hãy khám phá việc giữ quyền truy cập đọc vào sắp xếp lại chức năng trong bộ nhớ đệm và tránh các bước nhảy không thể đoán trước khi lặp qua định dạng nguồn. Chúng tôi sử dụng cấu trúc dữ liệu được sửa đổi một chút như trong Hình 4.5. Các hàm insert, setComparator, sort và getSize tương tự như permute ban đầu cấu trúc dữ liệu trong Hình 4.7. Sự thay đổi chính bao gồm việc giới thiệu getMap, getDim và một tham số dim với hàm tạo. Hàm getMap trả về vị trí được sắp xếp lại mới cho vị trí cũ idx và getDim trả về chiều dày đặc cho một số vị trí cũ idx.

4.2.4 Độ phức tạp

Phần này thảo luận về độ phức tạp của thuật toán tổng hợp và kết quả mã được tạo ra từ tổng hợp. Mã kết quả từ tổng hợp là trình kiểm tra cho

```
Sắp xếp lại luồng{
    ReorderStream(dim) void
    insert(Tuple tuple) void
    setComparator(Comparator comp) void sort()
    getSize() int
    getMap(idx)
    int getDim(kích
    thước, idx)
};
```

Hình 4.5: Cấu trúc dữ liệu hoán vị khám phá luồng sắp xếp lại

chuyển đổi định dạng thưa thớt chuyển đổi từ định dạng thưa thớt này sang định dạng thưa thớt khác. Nó nên xin lưu ý rằng độ phức tạp của thuật toán tổng hợp là chi phí thời gian biên dịch cho thanh tra trong khi độ phức tạp của trình kiểm tra mã được tạo ra là chi phí thời gian chạy. Tất cả các đánh giá được thực hiện trong công việc này trong Phần 4.3 đánh giá mã thanh tra cho định dạng chuyển đổi chứ không phải độ phức tạp của thuật toán tổng hợp.

Độ phức tạp của thuật toán tổng hợp

Thuật toán tổng hợp được tạo thành từ việc tạo ra SPF IR ban đầu và mã thế hệ từ SPF IR. Sự phức tạp tiệm cận của việc tạo ra SPF IR với thuật toán tổng hợp là O(nm) trong đó n là số ẩn số và m là số lượng ràng buộc. SPF IR hỗ trợ tạo mã với đa diện quét có độ phức tạp theo cấp số nhân O(q p) trong đó q là số trạng thái ments trong IR và p là kích thước bộ của lịch trình thực hiện phép tính. Trong thực tế, số lượng câu lệnh và kích thước lịch trình thực hiện thường nhỏ và thường bị ảnh hưởng bởi số lượng UF có trong các định dạng đích. Độ phức tạp tổng thể của thuật toán tổng hợp cho đến khi tạo mã là được đưa ra là O(q p).

Thanh tra phức tạp

Trong mã được tạo ra của chúng tôi, độ phức tạp phụ thuộc vào hàm sắp xếp lại được có trong mã được tạo ra cuối cùng. Trong trường hợp sắp xếp lại hoán vị hàm được tạo ra, trường hợp tệ nhất của mã được tạo ra sẽ là O(n log n) trong đó n là số lượng phần tử trong dữ liệu của định dạng nguồn. Trong các tình huống khi hàm sắp xếp lại không được tạo ra, trường hợp tệ nhất của mã được tạo ra sẽ là O(n). Chi phí bổ sung khi sử dụng chức năng sắp xếp lại là do bước sắp xếp bắt buộc của nó.

4.3 Đánh giá

Chúng tôi đã đánh giá tính chính xác và hiệu suất của mã tổng hợp bằng cách sử dụng một bộ của các ma trận thưa thớt từ Bộ sưu tập Ma trận SuiteSparse [23]. Mã tổng hợp là tuần tự, chúng tôi không khám phá các cơ hội song song hóa. Chúng tôi hiển thị kết quả cho COO cho CSR (COO CSR), CSR cho CSC (CSR CSC), và COO cho DIA (COO DIA). hiệu suất của các chuyển đổi thay đổi tùy thuộc vào định dạng mục tiêu. COO chuyển đổi sang CSR nhanh hơn TACO 2,85 lần, trong khi COO phức tạp hơn DIA chậm hơn TACO 1,4 lần nhưng nhanh hơn SPARSKIT và Intel MKL khi sử dụng trung bình hình học. Chúng tôi đánh giá kết quả cho COO MCOO bằng cách so sánh kết quả của chúng tôi với việc sắp xếp lại bước z-Morton viết tay trong HiCOO. Tất cả tăng tốc hoặc chậm lại so sánh sử dụng trung bình hình học.

4.3.1 Thiết lập thử nghiệm

Tất cả các thí nghiệm đều được chạy trên cụm Linux (CentOS bản phát hành 7) hỗ trợ 27 các nút tính toán, mỗi nút có CPU Intel Xeon E5-2680 14 lõi kép. Chúng tôi biên dịch mã được tạo và mã TACO sử dụng GCC 10.2.0.

```
#define s_0(n, ii) rowptr(ii) = min(rowptr(ii),n)
#xác định s0(__x0, a1, __x2, a3, __x4, __x5, __x6, __x7, __x8, __x9,
      __x10, __x11, __x12) s_0(a1, a3);
#xác định s_1(n, ii) rowptr(ii + 1) = max(rowptr(ii + 1),n + 1)
#xác định s1(__x0, a1, __x2, a3, __x4, __x5, __x6, __x7, __x8, __x9,
      __x10, __x11, __x12) s_1(a1, a3);
#define s_2(e1, e2) nếu ( không phải (rowptr(e1) <= rowptr(e2))){rowptr(e2)
      = rowptr(e1);}
#xác định s2(__x0, a1, __x2, a3, __x4, __x5, __x6, __x7, __x8, __x9,
      __x10, __x11, __x12) s_2(a1, a3);
#xác định s_3(n, ii, jj, ii1, k, jj1) col2(k)=jj1
#define s3(__x0, a1, __x2, a3, __x4, a5, __x6, a7, __x8, a9, __x10,
      a11, __x12) s_3(a1, a3, a5, a7, a9, a11);
#xác định s_4(n, ii, jj, ii1, k, jj1) ACSR(ii,k,jj) = ACOO(n,ii,jj )
#define s4(__x0, a1, __x2, a3, __x4, a5, __x6, a7, __x8, a9, __x10,
      a11, __x12) s_4(a1, a3, a5, a7, a9, a11);
 ......
nếu (NR >= 1 && NC >= 1) {
// Trường hợp 2: rowptr
cho(t2 = 0; t2 <= NNZ-1; t2++) {
        t4=hàng1_0(t1,t2);
        s0(0,t2,0,t4,0,0,0,0,0,0,0,0,0);

    }
// Trường hợp 3: rowptr
cho(t2 = 0; t2 <= NNZ-1; t2++) {
        t4=hàng1_0(t1,t2);
        s1(1,t2,0,t4,0,0,0,0,0,0,0,0,0);
    }
}
// Thực thi thuộc tính mảng trên rowptr
cho(t2 = 0; t2 <= NR-1; t2++) {
s2(2,t2,0,t2+1,0,0,0,0,0,0,0,0,0);
}
nếu (NC >= 1 && NR >= 1) {
// Trường hợp 1: cột
đối với (t2 = 0; t2 <= NNZ-1; t2++) {
      t4=hàng1_0(t1,t2);
      t6=cột1_1(t1,t2);
      s3(3,t2,0,t4,0,t6,0,t4,0,t2,0,t6,0);
    }
// Sao chép mã
đối với (t2 = 0; t2 <= NNZ-1; t2++) {
      t4=hàng1_0(t1,t2);
      t6=cột1_1(t1,t2);
      s4(5,t2,0,t4,0,t6,0,t4,0,t2,0,t6,0);
    }
}
```

Hình 4.6: Một ví dụ về mã được tạo ra từ COO đến CSR. Các phần mã bị bỏ qua để rõ ràng hơn.

```
1 Hoán vị <Bộ so sánh> { 2
vector<Tuple> newPos;
3 Permute(Comparator comparator) 4 void
sort() 5 void
insert(Tuple tuple) 6 int
get(originalPos) 7 }
```

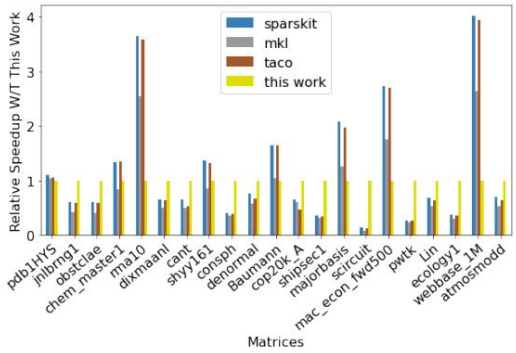
Hình 4.7: Cấu trúc dữ liệu hoán vị

So sánh hiệu suất được thực hiện bằng cách sử dụng cùng một tenxơ ma trận được sử dụng trong Công việc chuyển đổi định dạng của TACO. Bảng 4.4 hiển thị các ma trận được sử dụng trong đánh giá của chúng tôi. Ma trận COO được cho là được sắp xếp theo thứ tự từ điển hàng đầu tiên. Bảng 4.5 cho thấy Tenxơ 3D được sử dụng để đánh giá sự sắp xếp lại COO MCOO.

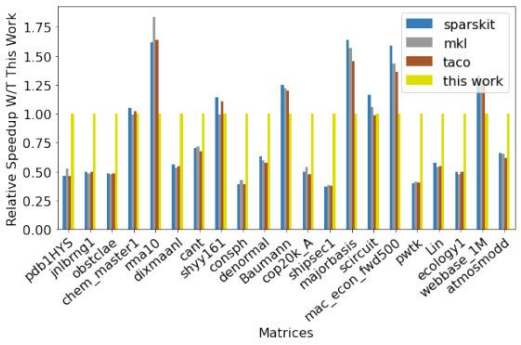
4.3.2 Đánh giá hiệu suất

Chúng tôi đánh giá mã kiểm tra để chuyển đổi định dạng được tạo ra bởi thuật toán của chúng tôi nhịp điệu bằng cách so sánh với Intel MKL, TACO [31], SPARSKIT [47] và viết tay Thứ tự z-morton của HiCOO [34]. Hình 4.8c hiển thị kết quả chuyển đổi từ COO đến CSR, nơi chúng tôi thấy tốc độ tăng đáng kể 2,85 lần so với TACO và các thư viện. Mã được tạo cho COO tới CSC (Hình 4.8a) và CSR tới CSC (Hình 4.8b) cho thấy tốc độ tăng 1,3x và 1,5x theo mức trung bình hình học tương ứng. COO đến CSR cho thấy sự tăng tốc đáng kể so với CSR CSC và COO CSC do _ _ hàng đầu tiên theo thứ tự từ điển của định dạng COO nguồn, không có hàm hoán vị nào được được tạo ra. Chỉ số hiệu suất là tốc độ tăng thời gian thực hiện của trạng thái của nghệ thuật so với công việc của chúng tôi (càng cao càng tốt).

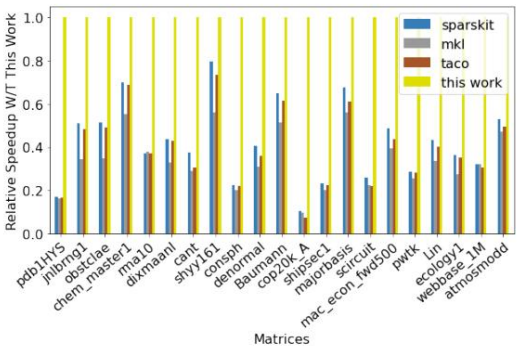
Chúng tôi so sánh COO-3D của chúng tôi với chuyển đổi COO-3D của Morton thành bản viết tay có độ chính xác cao bước đặt hàng z-morton được tối ưu hóa trong Hi-COO và chúng tôi cũng thấy tốc độ chậm lại 1,64 lần một giá trị trung bình hình học như được thấy trong Bảng 4.5. Thứ tự z-Morton viết tay chia tách



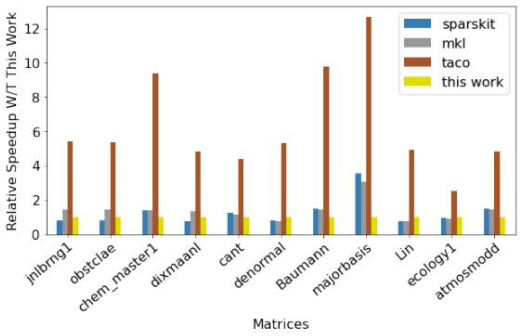
(a) COO cho CSC.



(b) CSR đến CSC.



(c) COO của CSR.

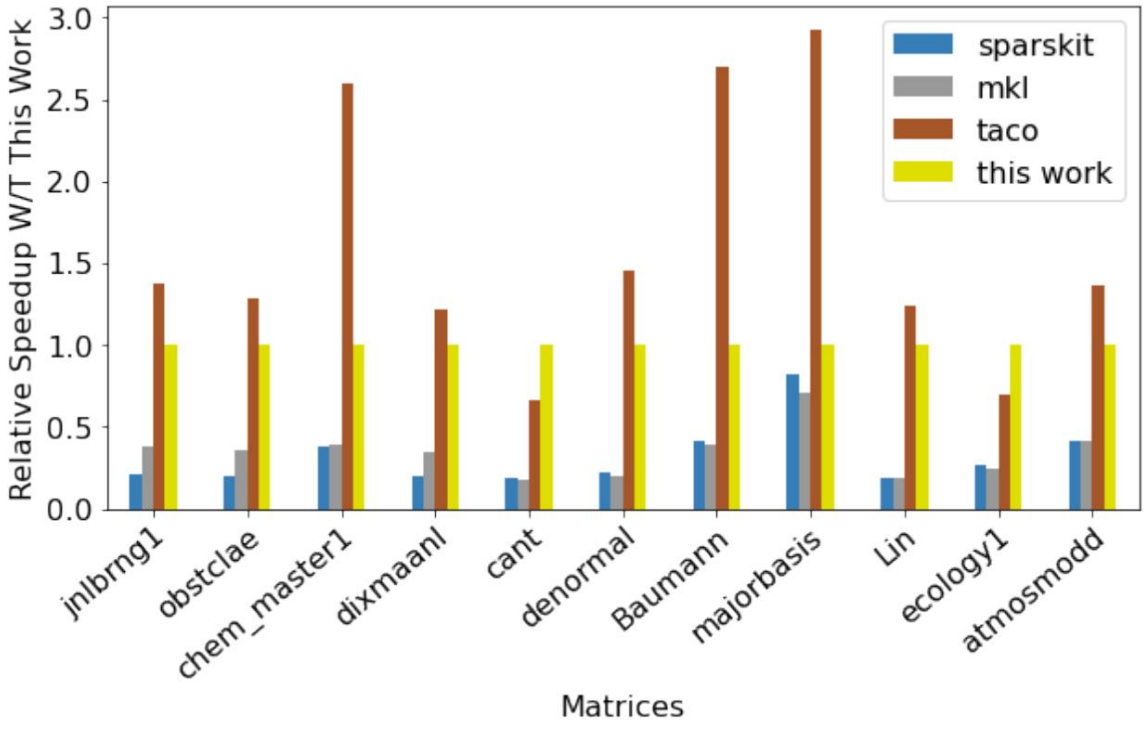


(d) COO đến DIA (Tối ưu hóa W/O) Xem Hình 4.9.

Hình 4.8: Kết quả hiệu suất của mã tổng hợp được tạo ra cho COO CSC, CSR CSC, COO CSR và COO_DIA. COO được cho là được sắp xếp theo thứ tự từ điển.

tenxơ ban đầu thành các hạt nhân nhỏ hơn và sau đó áp dụng một thuật toán sắp xếp Morton nhanh để sắp xếp từng hạt nhân khối. Điều này dẫn đến hiệu suất được cải thiện đáng kể so với kết quả của chúng tôi, vì họ chỉ sắp xếp các phần nhỏ tại một thời điểm. Chuyển đổi tenxơ theo thứ tự morton của chúng tôi thói quen kéo dài toàn bộ tenxơ.

Chi phí chung phát sinh do trừu tượng hóa hoán vị có thể được khấu hao bằng cách phân bổ chèn và sắp xếp alen hóa. Việc hoán vị định dạng nguồn cũng có thể được thực hiện trong nơi có khả năng làm giảm chi phí sao chép từ nguồn sang định dạng đích. Chúng tôi hiện không khám phá điều này vì chúng tôi cho rằng bản gốc tenxơ nguồn sẽ cần phải có sẵn sau khi tổng hợp. Một bản sao nhanh hơn từ nguồn đến



Hình 4.9: COO đến DIA với tìm kiếm nhị phân được sử dụng để tận dụng tính đơn điệu của mảng bù trừ tổng hợp được sử dụng trong sao chép.

đích đến có thể được thực hiện bằng cách sử dụng memcopy trực tiếp mà chúng tôi không khám phá trong bài viết này.

Kết quả hiệu suất cho COO DIA trong Hình 4.8d cho thấy hiệu suất khá cạnh tranh mance với các thư viện viết tay nhưng trung bình chậm hơn 5 lần so với TACO. Đây là một phần là do thực tế là các tối ưu hóa của chúng tôi không thể hợp nhất các vòng lặp tạo ra độ lệch và sao chép mã. Thuật toán tổng hợp tạo ra mã để thực thi các thuộc tính chỉ mục của UF không xác định. UF bù trừ trong trường hợp này có một thuộc tính mảng chỉ mục phải được thực thi trước khi UF có hiệu lực để được sử dụng trong mã được sao chép nhằm ngăn chặn sự hợp nhất cơ hội cho mã sao chép và mã bù trừ. Hiệu suất giảm dần theo số lượng của đường chéo. Xem xét kỹ hơn majorbasis (xem Hình 4.8d) cho thấy hiệu suất tệ nhất, số đường chéo có số khác không là 22 trong khi hiệu suất tốt nhất thực hiện sinh thái1 có 5 đường chéo. Mã tổng hợp của chúng tôi thử mọi lần lặp lại để

tìm d thỏa mãn các ràng buộc off(d)+i = j trước khi sao chép giá trị vào
tenxơ đích thích hợp. Ràng buộc này mô tả một hoạt động tìm kiếm tuyến tính,
khi được thay thế bằng tìm kiếm nhị phân sẽ cho kết quả tốt hơn như được hiển thị
trong Hình 4.9. Tìm kiếm nhị phân có thể thực hiện được nhờ các lượng từ phổ quát bật tắt
(Xem Bảng 4.1). Sự thay đổi này cho thấy kết quả được cải thiện vì chúng ta nhanh hơn 3,1 lần và 3,54 lần
chậm hơn SPARSKIT và MKL và chậm hơn TACO 1,4 lần theo giá trị trung bình hình học.

Tóm lại, mã tổng hợp có khả năng cạnh tranh với công nghệ tiên tiến nhất, ở một số khía cạnh.
trường hợp đánh bại hiệu suất. Chúng tôi dự đoán rằng tối ưu hóa tích cực hơn sẽ
mang lại kết quả tốt hơn.

Bảng 4.1: Mô tả định dạng cho COO, MortonCOO (MCOO), Morton COO 3D (MCOO3)

Định dạng	Bản đồ & Truy cập dữ liệu	Phạm vi miền & Thuộc tính mảng
Giám đốc điều hành	RACOO AD = {[n, ii, jj] [i, j]| hàng1(n) = i cột1(n) = j ii = i jj = j 0 ≤ i < NR 0 ≤ n < NNZ 0 ≤ j < NC} DICO ACOO = {[n, ii, jj] [n]}	phạm vi(hàng1) = {0 ≤ i < NR} miền(hàng1) = {0 ≤ x < NNZ} phạm vi(col1) = {0 ≤ i < NC} miền(col1) = {0 ≤ x < NNZ}
COO3D RACOO3D	AD = {[n, ii, jj, kk] [i, j, k]| hàng1(n) = i cột1(n) = j ii = i jj = j 0 ≤ i < NR kk = k 0 ≤ k < NZ z1(n) = k 0 ≤ n < NNZ 0 ≤ j < NC} DICOO3D ACOO3D = {[n, ii, jj, kk] [n]}	phạm vi(hàng1) = {0 ≤ i < NR} miền(hàng1) = {0 ≤ x < NNZ} phạm vi(col1) = {0 ≤ i < NC} miền(col1) = {0 ≤ x < NNZ} phạm vi(z1) = {0 ≤ k < NZ} miền (z1) = {0 ≤ x < NNZ}
MCOO RAMCOO	AD = {[n, ii, jj] [i, j]| rowm(n) = i colm(n) = j ii = i 0 ≤ i < NR 0 ≤ n < NNZ jj = j 0 ≤ j < NC} DIMCOO AMCOO = {[n, ii, jj] [n]}	phạm vi(hàngm) = {0 ≤ i < NR} miền(rowm) = {0 ≤ x < NNZ} phạm vi(cột) = {0 ≤ i < NC} miền(cột) = {0 ≤ x < NNZ} n1, n2 : n1 < n2 MORT TRÊN(hàngm(n1), cột(n1)) < MORT TRÊN(hàngm(n2) , cột(n2))
MCOO3 RAMCOO3	AD = {[n, ii, jj, kk] [i, j, k]| hàng1(n) = i cột1(n) = j ii = i jj = j 0 ≤ j < NC kk = k 0 ≤ k < NZ z1(n) = k 0 ≤ i < NR 0 ≤ n < NNZ} DIMCOO3 AMCOO3 = {[n, ii, jj, kk] [n]}	phạm vi(hàng1) = {0 ≤ i < NR} miền(hàng1) = {0 ≤ x < NNZ} phạm vi(col1) = {0 ≤ i < NC} miền(col1) = {0 ≤ x < NNZ} phạm vi(z1) = {0 ≤ k < NZ} miền (z1) = {0 ≤ x < NNZ} n1, n2 : n1 < n2 MORT TRÊN(hàng1(n1), col1(n1), z1(n1)) < MORT TRÊN(hàng1(n2), cột1(n2), z1(n2))

Bảng 4.2: Mô tả định dạng cho Sorted-COO(SCOO), CSR, DIA và CSC.

Định dạng	Bản đồ & Truy cập dữ liệu	Phạm vi miền & Thuộc tính mảng
SCOO RASCOO DISCOO	AD = {[n, ii, jj] [i, j]| hàng(n) = i cột(n) = j ii = i jj = j 0 ≤ i < NR 0 ≤ j < NC 0 ≤ n < NNZ} ASCOO = {[n, ii, jj] [n]}	phạm vi(hàng) = {0 ≤ i < NR} miền(hàng) = {0 ≤ x < NNZ} phạm vi(cột) = {0 ≤ i < NC} miền (cột) = {0 ≤ x < NNZ} n1, n2 : n1 < n2 hàng(n1) ≤ hàng(n2)
CSR RACSR DICSR	AD = {[ii, k, jj] [i, j]| ii = i jj = j col2(k) = j 0 ≤ ii < NR rowptr(ii) ≤ k k < rowptr(ii + 1)} ACSR = {[ii, k, jj] [k]}	phạm vi(rowptr) = {0 ≤ n ≤ NNZ} miền(rowptr) = {0 ≤ x ≤ NR} phạm vi(col2) = {0 ≤ i < NC} miền(col2) = {0 ≤ x < NNZ} ii1, ii2 : ii1 < ii2 rowptr(ii1) ≤ rowptr(ii2) k1, k2 : k1 < k2 mờ0(k1) ≤ mờ0(k2)
CSC	RACSC AD = {[jj, k, ii] [i, j]| 0 ≤ jj < NC colptr(jj) <= k k < colptr(jj + 1)} DICSC ACSC = {[ii, k, jj] [k]}	phạm vi(colptr) = {0 ≤ n ≤ NNZ} miền(colptr) = {0 ≤ x ≤ NC} phạm vi(hàng) = {0 ≤ i < NR} miền (hàng) = {0 ≤ x < NNZ} jj1, jj2 : jj1 < jj2 colptr(jj1) ≤ colptr(jj2) k1, k2 : k1 < k2 mờ1(k1) <= mờ1(k2)
DIA	RADIA AD = {[ii, d, jj] [i, j]| i = ii 0 ≤ i < NR 0 ≤ d < ND j = i + off(d) 0 <= j < NC} DIDIA ADIA = {[ii, d, jj] [kd]|} kd = ND ii + d}	miền(tắt) = {0 ≤ x ≤ ND} d1, d2 : d1 < d2 tắt(d1) < tắt(d2)

Bảng 4.4: Thống kê ma trận dùng để đánh giá COO CSR, CSR CSC, COO DIA. – –

Ma trận	Kích thước	NNZ
pdb1HYS	36,4K × 36,4K	4,3 triệu
jnlbrng1	40,0K × 40,0K	199K
obstclae	40,0K × 40,0K	199K
chem master1	40,4K × 40,4K	201K
rma10	46,8K × 46,8K	2,4 triệu
dixmaanl	60,0K × 60,0K	300K
cant	62,5K × 62,5K	4.0 triệu
shyy161	76,5K × 76,5K	330K
consph	83,3K × 83,3K	6.0 triệu
denormal	89,4K × 89,4K	1,2 triệu
Baumann	112K × 112K	748K
cop20k A	121K × 121K	2,6 triệu
shipsec1	141K × 141K	3,6 triệu
majorbasis	160K × 160K	1,8 triệu
scircuit	171K × 171K	959K
mac econ fwd500	207K × 207K	1,3 triệu
pwtk	218K × 218K	11,5 triệu
Lin	256K × 256K	1,8 triệu
sinh	1,00M × 1,00M	5.0 triệu
thái1	1,00M × 1,00M	3,1 triệu
webbase1M atmosmodd	1,27M × 1,27M	8,8 triệu

Bảng 4.5: Các tenxơ được sử dụng để đánh giá COO3D–MCOO3.

Tensor mờ		Chế độ NNZ		Thời gian thực hiện	
				Chào-coo	Của chúng tôi
darpa 22K ×	22K × 24M	3	28 triệu	11,85	20.13
phát thanh	23M × 23M × 166	3	100 phút	49,35	78,24
fb-s	39M × 39M × 532	3	140M	70,52	114,45

Bảng 4.6: Hỗ trợ chuyển đổi định dạng thưa thớt tự động trong công việc của chúng tôi so với người khác.

Định dạng Mô tả Hỗ trợ			
Dụng cụ	Bản đồ	Đặt hàng lại	Phổ quát Các số lượng
BÁNH TACO [31]		×	×
Nandy và cộng sự [39]	×		
Venkat và cộng sự [54]	×		
Công việc này			

CHƯƠNG 5

CÔNG VIỆC LIÊN QUAN

Chương này thảo luận về công trình trước đây có liên quan chặt chẽ nhất đến công trình này với trọng tâm trên các công cụ mô hình đa diện, đồ thị phụ thuộc chương trình, mô hình đa diện thưa thớt công cụ, sắp xếp lại tenxơ thưa thớt, chuyển đổi bố cục thưa thớt tự động (xem Bảng 4.6), chuyển đổi bố cục thưa thớt viết tay và tổng hợp chương trình.

5.1 Công cụ mô hình đa diện

Các công cụ như Polly [26], Pluto [12], Loopy [38], PolyMage [37] sử dụng đa diện mô hình để chuyển đổi các mã thông thường. Polly tự động phát hiện và chuyển đổi im- các phần mã quan trọng trong LLVM IR, phá vỡ các hạn chế của hầu hết các công cụ giới hạn sang một ngôn ngữ nguồn duy nhất. PolyMage là một ngôn ngữ dành riêng cho miền tự động hóa việc tạo ra các triển khai hiệu quả của các đường ống xử lý hình ảnh. Halide [44] là một trình biên dịch khác để tạo mã cho các thuật toán tính toán hình ảnh. Halide tách biệt thuật toán và đặc tả lập lịch, do đó cho phép tối ưu hóa các nhà khoa học viết các lịch trình khác nhau để có hiệu suất tối ưu. Công việc của họ cũng sử dụng tự động điều chỉnh để tạo ra mã hiệu quả bằng cách thực hiện tìm kiếm ngẫu nhiên để tìm ra mã tốt lịch trình cho thuật toán.

Pluto [12] là một công cụ chuyển đổi nguồn sang nguồn hoàn toàn tự động giúp tối ưu hóa chương trình cho tính song song và tính cục bộ. Pluto sử dụng lập trình tuyến tính số nguyên để quyết định

trên mã tối ưu sử dụng tính song song và tính cục bộ như một phần của các hàm chi phí của nó. Loopy, là một công cụ cho phép lập trình viên mô tả sự biến đổi vòng lặp ở cấp độ cao và xác minh tính chính xác của phép biến đổi. Loopy, giống như Polly, được triển khai trong LLVM. isl [56] là một công cụ để thao tác các tập hợp và mối quan hệ trong mô hình đa diện. Điều này công cụ tạo thành cơ sở cho các phép biến đổi afin được sử dụng trong tất cả các công cụ đã thảo luận trước đó trong phần này [12, 37, 38, 44].

5.2 Biểu đồ phụ thuộc chương trình

PDFG dựa trên một dòng nghiên cứu do Ferrante và cộng sự khởi xướng, [25] với làm việc trên đồ thị phụ thuộc chương trình. Công việc hiện tại chứng minh lợi ích của poly- tối ưu hóa luồng dữ liệu hedral. Olschanowsky và cộng sự đã chứng minh lợi ích này trên một chuẩn mực động lực học chất lỏng tính toán [40]. Davis et al. đã tự động hóa các thí nghiệm từ công việc trước đó sử dụng đồ thị luồng dữ liệu vĩ mô đã sửa đổi [22]. Đồng thời Mô hình lập trình Bộ sưu tập (CnC) [14] là một ngôn ngữ xử lý luồng và luồng dữ liệu đo lường nơi một chương trình là một đồ thị của các nút tính toán giao tiếp với nhau khác. DFGR [48] dựa trên các mô hình lập trình CnC và Habanero-C [7] và cho phép các nhà phát triển thể hiện các chương trình ở cấp độ cao với biểu đồ luồng dữ liệu như một biểu diễn trung gian. Công việc của chúng tôi sử dụng biểu đồ luồng dữ liệu để tập trung vào chuỗi tối ưu hóa mã trong khi DFGR và CnC khám phá tính song song. Luồng dữ liệu có trạng thái đa đồ thị (SDFG) [8] là một biểu diễn trung gian lấy dữ liệu làm trung tâm cho phép tách biệt định nghĩa mã khỏi việc tối ưu hóa của nó. Công việc của chúng tôi khác với SDFG do việc sử dụng mô hình đa diện. Các đồ thị không phải là biểu diễn trung gian, nhưng một cái nhìn về biểu diễn đó. Bất kỳ hoạt động đồ thị nào được thực hiện để chuyển đổi đồ thị được dịch sang các mối quan hệ và được áp dụng cho biểu diễn đa diện cơ bản

sự kiện.

5.3 Công cụ mô hình đa diện thưa thớt

Công việc được thực hiện trên việc biểu diễn các truy cập bộ nhớ gián tiếp trong một phép tính sử dụng mô hình đa diện đã chứng kiến sự phát triển của các công cụ như Omega [30] và Chill [45]. Omega [30] là một thư viện C++ để thao tác các mối quan hệ và tập hợp số nguyên. Codegen+ [18] được xây dựng trên omega và tạo mã với chức năng quét đa diện trong sự hiện diện của các hàm không được diễn giải. Chill [45] là một phép biến đổi trình biên dịch đa diện và khuôn khổ tạo mã sử dụng Codegen+ để tạo mã. Nó cho phép người dùng để chỉ định các chuỗi chuyển đổi thông qua các tập lệnh. Công việc của chúng tôi khác với làm việc như chúng tôi đại diện cho một cái nhìn toàn diện về một phép tính và chúng tôi hỗ trợ chính xác hơn sự biến đổi khi có sự tính toán thưa thớt.

5.4 Sắp xếp lại Tensor thưa thớt

Sắp xếp lại tenxơ thưa thớt liên quan đến việc thay đổi thứ tự các mục không phải số không trong tenxơ thưa thớt định dạng để cải thiện vị trí không gian hoặc thời gian. Điều này bao gồm các kỹ thuật tìm kiếm: BFS-MCS tìm kiếm theo chiều rộng trên họ tìm kiếm số lượng lớn nhất và Lexi- Đặt hàng mở rộng của thứ tự từ vựng kép của ma trận thành tenxơ [35]. Một cách tiếp cận để sắp xếp lại tenxơ là sử dụng phép biến đổi cosin rời rạc (DCT) để nén Mạng nơ-ron tích chập (CNN) [35]. Công việc của chúng tôi tương tự như loại công việc này khi chúng tôi giới thiệu một cách tiếp cận chính thức để chỉ định các chức năng sắp xếp lại cho tự động tổng hợp các thói quen chuyển đổi.

5.5 Tự động chuyển đổi bố cục thưa thớt

Bảng 4.6 hiển thị công việc trên các chuyển đổi bố cục dữ liệu tự động. Bản đồ bao gồm công việc sử dụng một hàm để mô tả mối quan hệ giữa không gian thưa thớt và không gian dày đặc, sắp xếp lại là một lớp công việc với việc sắp xếp dữ liệu xáo trộn và phổ quát các lượng tử mô tả các thuộc tính của mảng và tích hợp thông tin đó trong sự phụ thuộc phân tích và tối ưu hóa.

Các kỹ thuật dựa trên tập lệnh giới thiệu một tập hợp các chuyển đổi khi được kết hợp trong thứ tự nhất định để tạo điều kiện thuận lợi cho việc chuyển đổi bố cục dữ liệu [39, 53, 54]. Trình biên dịch chuyển đổi các đội hình được sử dụng như các khối xây dựng để viết các tập lệnh chuyển đổi từ một dữ liệu định dạng sang một định dạng khác. Cách tiếp cận này yêu cầu phải viết thủ công 2n tập lệnh; một cho mỗi sự kết hợp có thể có của các định dạng. Công việc của chúng tôi khác với công việc này vì chúng tôi tập trung vào chuyển đổi định dạng và sử dụng phương pháp tiếp cận dựa trên mô tả bằng cách sử dụng bản đồ để tự động tổng hợp mã giữa các định dạng. Cách tiếp cận của chúng tôi tương tự như khi chúng tôi xây dựng trên khung đa diện; chúng tôi cũng hỗ trợ sắp xếp lại và sử dụng phổ quát các bộ lượng hóa mở ra cơ hội cho các thử nghiệm phụ thuộc và tối ưu hóa.

Arnold et al. [6] sử dụng một ngôn ngữ nhỏ chức năng (LL) để mô tả bố cục thưa thớt thông qua quá trình chuyển đổi của nó từ một ma trận dày đặc. LL cho phép tạo ra và chứng minh tính đúng đắn của tính toán thưa thớt theo các bố cục khác nhau. Tuy nhiên, vòng lặp các phép biến đổi như hợp nhất và ghép lát không được hỗ trợ.

TACO [19, 20, 31] là trình biên dịch đại số tenxơ định nghĩa các bố cục thưa thớt bằng cách sử dụng tập hợp các tên cho mỗi chiều của tenxơ được gọi là định dạng mức. Các hàm mức là được xác định cho định dạng này để hỗ trợ các hoạt động nguyên thủy của kích thước, bao gồm lặp lại, truy cập và lắp ráp. Chuyển đổi định dạng được thực hiện trong TACO bằng cách lập bản đồ đến và đi từ không gian dày đặc, phân tích số liệu thống kê cấu trúc của tenxơ,

và lắp ráp bố cục đích bằng cách sử dụng các hàm cấp độ. Tuy nhiên, công việc này,
không xem xét các thuộc tính như lượng từ phổ quát và sắp xếp lại như được thể hiện trong
Bảng 4.6.

Bik et al. [10] công việc tenxơ thưa thớt bổ sung cho công việc của chúng tôi, họ mô tả
định dạng thưa thớt với các thuộc tính mức độ và tạo mã tính toán thưa thớt trong MLIR.
Trong trường hợp cần chuyển đổi từ định dạng này sang định dạng khác, mức
các thuộc tính có thể được dịch sang mô tả thưa thớt cấp cao của chúng tôi sau đó
thuật toán tổng hợp được áp dụng. Tổng hợp của chúng tôi tạo ra một trung gian cấp cao
biểu diễn có thể được mở rộng để tạo mã cho MLIR.

5.6 Bố trí Tensor tối ưu

Bik et al. [11] và SIPR [43] tối ưu hóa tính toán liên quan đến tenxơ thưa thớt bằng cách gợi ý
đưa ra các bố cục hiệu quả hơn từ việc phân tích tĩnh quá trình tính toán. Sparso [46]
tối ưu hóa một chuỗi tính toán tenxơ bằng cách sắp xếp lại tập thể theo ngữ cảnh
phân tích và khám phá thuộc tính ma trận. Sparso có thể tự động xác định dựa trên
về thông tin tĩnh và thời gian chạy khi nào bố cục nên được chuyển đổi bằng cách sử dụng được xác định trước
thói quen thư viện. Tuy nhiên, những tác phẩm này không nhắm mục tiêu vào thói quen chuyển đổi: hoặc
loại trừ chúng khỏi việc xem xét hoặc coi chúng như một hộp đen.

5.7 Chuyển đổi định dạng thưa thớt thủ công

Sparskit cung cấp nhiều chức năng khác nhau để xử lý các ma trận thưa thớt. Nó
giúp dịch một dạng ma trận sang các dạng khác [47]. Nó hỗ trợ 12 loại lưu trữ khác nhau
định dạng cho ma trận. Intel MKL là một thư viện khác cung cấp nhiều thói quen và
chức năng để thực hiện các phép tính trên các ma trận thưa thớt [58]. Đôi khi để có được

một định dạng đích, một định dạng trung gian phải được chuyển đổi trước. Cách tiếp cận của chúng tôi khác với cách tiếp cận này vì chúng ta cần n mô tả để tự động tổng hợp 2 quy trình chuyển đổi n .

5.8 Tổng hợp chương trình

Trình giải các lý thuyết modulo về khả năng thỏa mãn (SMT) và trình chứng minh logic bậc cao (HOL) thường được sử dụng để kiểm tra xem mã được tạo ra có đồng ý với các thông số kỹ thuật được cung cấp hay không Tổng hợp một chương trình chung sẽ dẫn đến một không gian tìm kiếm không thể quản lý được của mã được tạo ra, do đó các ràng buộc và phương pháp tiếp cận được cung cấp.

Phác thảo [13,49,50] là một cách tiếp cận tổng hợp chương trình giới hạn phạm vi của tổng hợp thành các chi tiết cấp thấp trong một bản phác thảo thuật toán hoặc các bản phác thảo siêu dữ liệu được cung cấp bởi lập trình viên. Tổng hợp hướng dẫn cú pháp [4] sử dụng một phản ví dụ hướng dẫn-chiến lược tổng hợp cảm ứng để giải quyết vấn đề tổng hợp theo các chương trình hợp lệ theo một tập hợp cú pháp. Gần đây hơn, Knoth et al. [32] đã giới thiệu một hệ thống kiểu cung cấp phân tích tài nguyên khấu hao tự động để sử dụng như một phương pháp tìm kiếm trong quá trình quá trình tổng hợp.

CHƯƠNG 6

KẾT LUẬN VÀ CÔNG VIỆC TƯƠNG LAI

Chương này cung cấp tóm tắt tổng quan về những đóng góp của luận án này và thảo luận về phương hướng cho công việc trong tương lai.

6.1 Biểu diễn trung gian SPF

Công trình này trình bày một API hướng đối tượng cho thư viện đa diện thưa thớt. API cung cấp một giao diện chuẩn để chỉ định đầy đủ một phép tính trong Poly- thưa thớt khung hedral từ một điểm vào duy nhất. Điểm vào duy nhất được kích hoạt bởi tích hợp chặt chẽ giữa IEGenLib, CodeGen+ và PDFG. PDFG đã được mở rộng để biểu diễn ranh giới vòng lặp không afin, lồng vòng lặp không hoàn hảo và vòng lặp mang theo sự phụ thuộc. API hiện không hỗ trợ thoát sớm, vòng lặp nơi các giới hạn được sửa đổi trong các lồng vòng lặp và các vòng lặp không có giới hạn như vòng lặp while. Trong tương lai, chúng tôi hy vọng sẽ khám phá các kỹ thuật như từ [9] đến khắc phục những hạn chế của việc biểu diễn các vòng lặp không giới hạn và các vị từ thoát.

Chúng tôi cung cấp hỗ trợ cho việc kết hợp các phép tính thông qua nội tuyến. Điều này làm tăng khả năng tái sử dụng và độ tin cậy. Mã được tạo ra bởi một phép tính bao gồm trạng thái các macro và khi cần, các khai báo biến. Điều này có nghĩa là trong tương lai, quyết định bố trí bộ nhớ cho lưu trữ tạm thời có thể được thực hiện hoàn toàn trong SPF.

Các công cụ trong tương lai sẽ sử dụng API này để tạo SPF IR, thao tác nó và sản xuất mã được tối ưu hóa từ các ứng dụng cũ và tùy chỉnh cấp cao hoặc dành riêng cho miền ngôn ngữ. Ngoài ra, API này sẽ được sử dụng khi chuyển đổi lặp đi lặp lại một hợp chất đưa ra; PDFG sẽ được hiển thị ở mọi bước để hướng dẫn chuyên gia thực hiện quyết định.

6.2 Tổng hợp mã

Trong công trình này, chúng tôi giới thiệu một cách tiếp cận để mô tả chính thức các định dạng tenxơ thưa thớt và tổng hợp mã dịch giữa chúng bằng cách sử dụng khuôn khổ đa diện thưa thớt.
Công trình này trình bày một định nghĩa chính thức về các định dạng tenxơ thưa thớt và một cách tiếp cận để tổng hợp sự chuyển đổi giữa các định dạng. Cách tiếp cận này là duy nhất trong đó nó hỗ trợ các ràng buộc về thứ tự không được hỗ trợ bởi các phương pháp tiếp cận khác và tổng hợp mã chuyển đổi trong một bộ biểu diễn trung gian cấp cao có khả năng áp dụng các phép biến đổi có thể cấu hình như hợp nhất vòng lặp và tạm thời giảm lưu trữ. Chúng tôi chứng minh rằng mã tổng hợp cho COO đến CSR với tối ưu hóa nhanh hơn 2,85 lần so với TACO, Intel MKL và SPARSKIT trong khi COO phức tạp đến DIA chậm hơn TACO 1,4 lần nhưng nhanh hơn SPARSKIT và Intel MKL sử dụng giá trị trung bình hình học của thời gian thực hiện.

6.3 Hướng đi trong tương lai

Đối với công việc trong tương lai, chúng tôi dự định khám phá thêm nhiều chuyển đổi trong SPF để cải thiện hiệu suất chung của công việc của chúng tôi. Tự động hướng dẫn người dùng lựa chọn định dạng thưa thớt tốt nhất sẽ là một hướng thú vị. Kết hợp bố cục dữ liệu chuyển đổi như một phần của đường ống biên dịch có sẵn là hướng chúng ta sẽ khám phá.

Trong công việc tổng hợp, không có cách chính thức nào để xác minh tính hợp lệ của thưa thớt các mô tả. Các mô tả thưa thớt không được mô tả đúng trong công việc của chúng tôi gây ra sự dừng lại vấn đề trong thuật toán của chúng tôi. Tất cả các mô tả thưa thớt được xác định trong tác phẩm này đã được được thử nghiệm để hoạt động và tạo ra mã chính xác theo kinh nghiệm. Một công việc thú vị trong tương lai sẽ là để xác minh chính thức tính hợp lệ của một mô tả định dạng thưa thớt. Khám phá hỗ trợ đối với việc chuyển đổi kiểu dữ liệu trừu tượng là một hướng có thể cho công việc trong tương lai. hướng đi tuyệt vời cho công việc này là tích hợp đầy đủ với các trình biên dịch đã có sẵn như MLIR, LLVM và GCC. Tích hợp với trình biên dịch sẽ có nghĩa là bố cục dữ liệu có thể được tự động chuyển đổi để có hiệu suất tối ưu.

TÀI LIỆU THAM KHẢO

[1] Bộ sưu tập trình biên dịch gnu (gcc) nội bộ: gimple.

[2] Thư viện IEGenLib, kho lưu trữ github. https://github.com/CompOpt4Apps/IEGenLib/tree/SC16 hiện vật SC16 IEGenLib, 2019.

[3] Khalid Ahmad, Hari Sundar và Mary Hall. Phép nhân vectơ ma trận thưa hỗn hợp độ chính xác dựa trên dữ liệu cho GPU. ACM Trans. Archit. Code Optim., 16(4), tháng 12 năm 2019.

[4] Rajeev Alur, Rastislav Bodik, Garvit Juniwal, Milo MK Martin, Mukund Raghothaman, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak và Abhishek Udupa. Tổng hợp theo hướng dẫn cú pháp. Trong năm 2013 Phương pháp chính thức trong thiết kế có sự hỗ trợ của máy tính, trang 1–8, 2013.

[5] Aaron Orenstein Anna Rift, Tobi Popoola. Khung đa diện thưa thớt inpec-tor/executor spf-ie, 2020.

[6] Gilad Arnold, Johannes H¨olzl, Ali Sinan K¨oksal, Rastislav Bod´ık, và Mooly Sagiv. Chỉ định và xác minh mã ma trận thưa thớt. Trong Biên bản Hội nghị quốc tế ACM SIGPLAN lần thứ 15 về lập trình hàm, ICFP '10, trang 249–260, New York, NY, Hoa Kỳ, 2010. Hiệp hội máy tính.

[7] Rajkishore Barik, Zoran Budimlic, Vincent Cav`e, Sanjay Chatterjee, Yi Guo, David Peixotto, Raghavan Raman, Jun Shirako, Sa˘gnak Ta¸sırlar, Yonghong Yan, Yisheng Zhao và Vivek Sarkar. Dự án nghiên cứu phần mềm đa lõi habanero. Trong Biên bản báo cáo Hội nghị ACM SIGPLAN lần thứ 24 về Ngôn ngữ lập trình hướng đối tượng và Ứng dụng hệ thống, OOPSLA '09, trang 735–736, New York, NY, Hoa Kỳ, 2009. Hiệp hội máy tính.

[8] Tal Ben-Nun, Johannes de Fine Licht, Alexandros N. Ziogas, Timo Schneider và Torsten Hoefler. Đồ thị đa luồng dữ liệu có trạng thái: Một mô hình tập trung vào dữ liệu để di chuyển hiệu suất trên các kiến trúc không đồng nhất. Trong Biên bản báo cáo của Hội nghị quốc tế về điện toán hiệu suất cao, mạng, lưu trữ

và Phân tích, SC '19, New York, NY, Hoa Kỳ, 2019. Hiệp hội Máy tính.

[9] Mohamed-Walid Benabderrahmane, Louis-No¨el Pouchet, Albert Cohen và C´edric Bastoul. Mô hình đa diện có thể áp dụng rộng rãi hơn bạn nghĩ. Trong Compiler Construction, tập LNCS 6011, Berlin, Heidelberg, 2010.
Nhà xuất bản Springer.

[10] Aart JC Bik. Hỗ trợ trình biên dịch cho các phép tính tenxơ thưa thớt trong mlir, 2021. https://youtu.be/x-nHc3hBxHM.

[11] AartJ.C. Bik và HarryA.G. Wijshoff. Về lựa chọn cấu trúc dữ liệu tự động và tạo mã cho các phép tính thưa thớt. Trong Utpal Banerjee, David Gelernter, Alex Nicolau và David Padua, biên tập viên, Ngôn ngữ và trình biên dịch cho tính toán song song, tập 768 của Ghi chú bài giảng về khoa học máy tính, trang 57–75.
Nhà xuất bản Springer Berlin Heidelberg, 1994.

[12] 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.

[13] James Bornholt, Emina Torlak, Dan Grossman và Luis Ceze. Tối ưu hóa tổng hợp với metaasketches. Trong Biên bản báo cáo Hội nghị chuyên đề thường niên lần thứ 43 của ACM SIGPLAN-SIGACT về Nguyên tắc của Ngôn ngữ lập trình, POPL '16, trang 775–788, New York, NY, Hoa Kỳ, 2016. Hiệp hội Máy tính.

[14] Zoran Budimli'c, Michael Burke, Vincent Cav'e, Kathleen Knobe, Geoff Lowney, Ryan Newton, Jens Palsberg, David Peixotto, Vivek Sarkar, Frank Schlimbach và Sa˘gnak Ta¸sırlar. Bộ sưu tập đồng thời Khoa học. Chương trình., 18(3–4):203–217, tháng 8 năm 2010.

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

[16] Chun Chen. Xem xét lại quét đa diện. Trong Biên bản báo cáo Hội nghị ACM SIGPLAN lần thứ 33 về Thiết kế và Triển khai Ngôn ngữ Lập trình, PLDI '12, trang 499–508, New York, NY, Hoa Kỳ, 2012. ACM.

[17] Chun Chen. Xem xét lại quét đa diện. Trong Biên bản báo cáo của hội nghị ACM SIGPLAN lần thứ 33 về Thiết kế và triển khai ngôn ngữ lập trình, PLDI '12, trang 499–508, tháng 6 năm 2012.

[18] Chun Chen. Xem xét lại việc quét đa diện. Trong Biên bản báo cáo của Hội nghị ACM SIGPLAN về Thiết kế và Triển khai Ngôn ngữ Lập trình (PLDI), 2012.

[19] Stephen Chou, Fredrik Kjolstad và Saman Amarasinghe. Định dạng trừu tượng cho trình biên dịch đại số tenxơ thưa thớt. Proc. Chương trình ACM. Lang., 2(OOPSLA):123:1–123:30, tháng 10 năm 2018.

[20] Stephen Chou, Fredrik Kjolstad và Saman Amarasinghe. Tự động tạo các quy trình chuyển đổi định dạng tenxơ thưa thớt hiệu quả. Trong Biên bản báo cáo Hội nghị ACM SIGPLAN lần thứ 41 về Thiết kế và Triển khai Ngôn ngữ Lập trình, PLDI 2020, trang 823–838, New York, NY, Hoa Kỳ, 2020. Hiệp hội Máy tính.

[21] Eddie C. Davis, Michelle Mills Strout và Catherine Olschanowsky. Biến đổi chuỗi vòng lặp thông qua đồ thị luồng dữ liệu vĩ mô. Trong Biên bản báo cáo Hội nghị chuyên đề quốc tế năm 2018 về Tạo mã và Tối ưu hóa, CGO 2018, trang 265ˆa277, New York, NY, Hoa Kỳ, 2018. Hiệp hội Máy tính.

[22] 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.

[23] Timothy A. Davis và Yifan Hu. Ma trận thưa thớt của trường đại học Florida bộ sưu tập. ACM Trans. Math. Softw., 38(1), tháng 12 năm 2011.

[24] Timothy A. Davis và Yifan Hu. Ma trận thưa thớt của trường đại học Florida bộ sưu tập. ACM Trans. Math. Softw., 38(1), tháng 12 năm 2011.

[25] Jeanne Ferrante, Karl J. Ottenstein và Joe D. Warren. Đồ thị phụ thuộc chương trình và việc sử dụng nó trong tối ưu hóa. Giao dịch ACM về Ngôn ngữ lập trình và Hệ thống, 9(3):319–349, tháng 7 năm 1987.

[26] 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.

[27] Mary Hall, Jacqueline Chame, Jaewook Shin, Chun Chen, Gabe Rudy và Malik Murtaza Khan. Công thức chuyển đổi vòng lặp để tạo mã và tự động điều chỉnh. Trong Biên bản Hội thảo về Ngôn ngữ và Trình biên dịch cho Máy tính song song (LCPC), 2009.

[28] Ahmed E. Helal, Jan Laukemann, Fabio Checconi, Jesmin Jahan Tithi, Teresa Ranadive, Fabrizio Petrini và Jeewhan Choi. ALTO: Lưu trữ tuyến tính thích ứng của tenxơ thưa thớt. Biên bản Hội nghị quốc tế về siêu máy tính, trang 404–416, 2021.

[29] Maxime R. Hugues và Serge G. Petiton. Đánh giá và tối ưu hóa định dạng ma trận thưa thớt trên GPU. Biên bản báo cáo - Hội nghị quốc tế IEEE lần thứ 12 về máy tính hiệu suất cao và truyền thông năm 2010, HPCC 2010, trang 122–129, 2010.

[30] Wayne Kelly, Vadim Maslov, William Pugh, Evan Rosser, Tatiana Shpeisman và David Wonnacott. Hướng dẫn giao diện Thư viện Omega. Báo cáo kỹ thuật CS-TR-3445, Đại học Maryland tại College Park, tháng 3 năm 1995.

[31] Fredrik Kjolstad, Shoaib Kamil, Stephen Chou, David Lugato và Saman Amarasinghe. Trình biên dịch đại số tensor. Proc. Chương trình ACM. Lang., 1(OOPSLA):77:1–77:29, tháng 10 năm 2017.

[32] Tristan Knoth, Di Wang, Nadia Polikarpova và Jan Hoffmann. Tổng hợp chương trình hướng dẫn tài nguyên. Trong Biên bản Hội nghị ACM SIGPLAN lần thứ 40 về Thiết kế và Triển khai Ngôn ngữ Lập trình, PLDI 2019, trang 253–268, New York, NY, Hoa Kỳ, 2019. Hiệp hội Máy tính.

[33] Chris Lattner và Vikram Adve. Bộ hướng dẫn LLVM và chiến lược biên dịch. Khoa CS, Đại học Illinois tại Urbana-Champaign, . . . , trang 1–20, 2002.

[34] Jiajia Li, Jimeng Sun và Richard Vuduc. Hicoo: Lưu trữ phân cấp của tenxơ thưa thớt. Trong Biên bản báo cáo Hội nghị quốc tế về tính toán hiệu suất cao, mạng, lưu trữ và phân tích, SC '18, trang 19:1–19:15, Pis-cataway, NJ, Hoa Kỳ, 2018. IEEE Press.

..

[35] Jiajia Li, Bora U¸car, Umit V. C¸ataly¨urek, Jimeng Sun, Kevin Barker và Richard Vuduc. Sắp xếp lại tenxơ thưa hiệu quả và hiệu suất. Trong Biên bản báo cáo Hội nghị quốc tế ACM về siêu máy tính, ICS '19, trang 227–237, New York, NY, Hoa Kỳ, 2019. ACM.

[36] Mahdi Soltan Mohammadi, Kazem Cheshmi, Maryam Mehri Dehnavi, Anand Venkat, Tomofumi Yuki và Michelle Mills Strout. Mở rộng các thuộc tính mảng chỉ mục để phân tích sự phụ thuộc dữ liệu. Trong Ngôn ngữ và trình biên dịch cho tính toán song song (LCPC), 2018.

[37] Ravi Teja Mullapudi, Vinay Vasista và Uday Bondhugula. PolyMage. ACM Thông báo SIGPLAN, 50(4):429–443, tháng 3 năm 2015.

[38] 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.

[39] Payal Nandy, Mary Hall, Eddie C. Davis, Catherine Olschanowsky, Mahdi Soltan Mohammadi, Wei He và Michelle Mills Strout. Các phép trừu tượng để chỉ định các phép biến đổi dữ liệu ma trận thưa thớt. Trong Hội thảo quốc tế lần thứ 8 về các kỹ thuật biên dịch đa diện (IMPACT), ngày 23 tháng 1 năm 2018.

[40] 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.

[41] Tobi Popoola, Ravi Shankar, Anna Rift, Shivani Singh, Eddie C. Davis, Michelle Mills Strout và Catherine Olschanowsky. Giao diện hướng đối tượng cho thư viện đa diện thưa thớt. Trong Hội nghị thường niên lần thứ 45 về máy tính, phần mềm và ứng dụng của IEEE (COMPSAC) năm 2021, trang 1825–1831, 2021.

[42] Tobi Popoola, Tuowen Zhao, Aaron St. George, Kalyan Bhetwal, Michelle Mills Strout, Mary Hall và Catherine Olschanowsky. Gói tái tạo để tổng hợp mã cho chuyển đổi và tối ưu hóa định dạng tenxơ thưa thớt. Tháng 3 năm 2023.

[43] William Pugh và Tatiana Shpeisman. Sipr: Một khuôn khổ mới để tạo mã hiệu quả cho các phép tính ma trận thưa thớt. Trong Biên bản Hội thảo quốc tế lần thứ 11 về Ngôn ngữ và Trình biên dịch cho Máy tính song song, Chapel Hill, Bắc Carolina, tháng 8 năm 1998.

[44] Jonathan Ragan-Kelley, Andrew Adams, Sylvain Paris, Fr´edo Durand, Connelly Barnes và Saman Amarasinghe. Halide: Một 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. Biên bản Hội nghị ACM SIGPLAN lần thứ 34 về Thiết kế và Triển khai Ngôn ngữ Lập trình, trang 519–530, 2013.

[45] Nhóm nghiên cứu Mary Hall. Chill - trình biên dịch vòng lặp nguồn-nguồn cấp cao có thể cấu hình. https://github.com/CtopCsUtahEdu/chill, 2019.

[46] Hongbo Rong, Jongsoo Park, Lingxiang Xiang, Todd A. Anderson và Mikhail Smelyanskiy. Sparso: Tối ưu hóa theo ngữ cảnh của đại số tuyến tính thưa thớt. Trong Hội nghị quốc tế năm 2016 về Kiến trúc song song và Kỹ thuật biên dịch (PACT), trang 247–259, 2016.

[47] Yousef Saad. Sparskit: bộ công cụ cơ bản cho phép tính ma trận thưa thớt, 1994.

[48] Alina Sbirlea, Louis-Noel Pouchet và Vivek Sarkar. Dfgr, một biểu diễn đồ thị trung gian cho các chương trình luồng dữ liệu vĩ mô. Trong Hội thảo lần thứ tư năm 2014 về các mô hình thực thi luồng dữ liệu cho tính toán quy mô cực đại, trang 38–45, năm 2014.

[49] Armando Solar-Lezama. Phương pháp phác thảo để tổng hợp chương trình. Trong Biên bản báo cáo Hội nghị chuyên đề Châu Á lần thứ 7 về Ngôn ngữ lập trình và Hệ thống, APLAS '09, trang 4–13, Berlin, Heidelberg, 2009. Springer-Verlag.

[50] Armando Solar-Lezama, Liviu Tancau, Rastislav Bodik, Sanjit Seshia và Vijay Saraswat. Phác thảo kết hợp cho các chương trình hữu hạn. Trong Biên bản báo cáo Hội nghị quốc tế lần thứ 12 về Hỗ trợ kiến trúc cho Ngôn ngữ lập trình và Hệ điều hành, ASPLOS XII, trang 404–415, New York, NY, Hoa Kỳ, 2006.
Hiệp hội máy móc điện toán.

[51] Michelle Mills Strout, Geri Georg và Catherine Olschanowsky. Tập hợp và thao tác quan hệ cho Khung đa diện thưa thớt. 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), 7760 LNCS:61–75, 2013.

[52] Michelle Mills Strout, Geri George và Catherine Olschanowsky. Tập hợp và thao tác quan hệ cho khuôn khổ đa diện thưa thớt. Trong Biên bản Hội thảo quốc tế lần thứ 25 về Ngôn ngữ và Trình biên dịch cho Máy tính song song (LCPC), tháng 9 năm 2012.

[53] Anand Venkat, Mary Hall và Michelle Strout. Vòng lặp và chuyển đổi dữ liệu cho mã ma trận thưa thớt. SIGPLAN Not., 50(6):521–532, tháng 6 năm 2015.

[54] Anand Venkat, Manu Shantharam, Mary Hall và Michelle Mills Strout.
Các phần mở rộng không phải affine cho việc tạo mã đa diện. Trong Biên bản báo cáo Hội nghị quốc tế IEEE/ACM thường niên về tạo mã và tối ưu hóa, CGO '14, 2014.

[55] Sven Verdoolaege. isl: Thư viện tập hợp số nguyên cho mô hình đa diện. Trong Komei Fukuda, Joris van der Hoeven, Michael Joswig và Nobuki Takayama, biên tập viên, Phần mềm toán học – ICMS 2010, trang 299–302, Berlin, Heidelberg, 2010.
Springer Berlin Heidelberg.

[56] Sven Verdoolaege. isl: Thư viện tập hợp số nguyên cho mô hình đa diện. Trong Komei Fukuda, Joris van der Hoeven, Michael Joswig và Nobuki Takayama, biên tập viên, Ghi chú bài giảng về khoa học máy tính, trang 299–302. Springer, tháng 9 năm 2010.

[57] Kelly W, Maslov V, Pugh W, Rosser E, Shpeisman T và Wonnacott D. Hướng dẫn giao diện Thư viện Omega. Đại học Maryland tại College Park, tháng 3 năm 1995, 1995.

[58] Endong Wang, Qing Zhang, Bo Shen, Zhang Yongyong, Xiaowei Lu, Qing Wu, và Yajuan Wang. Thư viện hạt nhân toán học Intel. Trong Điện toán hiệu năng cao trên Intel® Xeon Phi, trang 167–188. Mùa xuân, 2014.

6.4 Phụ lục hiện vật tổng hợp

6.4.1 Tóm tắt

Hiện vật này giới thiệu một kỹ thuật để chuyển đổi bố cục dữ liệu dựa trên mối quan hệ bị hạn chế giữa các dạng dữ liệu khác nhau. Trong hiện vật này, chúng tôi áp dụng Kỹ thuật này dùng để tạo mã chuyển đổi từ định dạng nguồn này sang định dạng nguồn khác. Chúng tôi cung cấp một container docker để sao chép kết quả. Để dễ dàng thử nghiệm, chúng tôi đã cung cấp tạo ra các mã chuyển đổi được bao quanh bởi các macro cần thiết để đánh giá công việc. Chúng tôi cũng cung cấp các hiện vật từ công nghệ tiên tiến được thảo luận trong công việc của chúng tôi.

6.4.2 Danh sách kiểm tra hiện vật (siêu thông tin)

- Thuật toán: Giải quyết ràng buộc, tạo mã đa diện, tối ưu hóa
- Biên dịch: Được biết là biên dịch với GCC v10
- Chuyển đổi: hợp nhất, loại bỏ mã chết
- Bộ dữ liệu: Bộ sưu tập Ma trận SuiteSparse [24] xem Bảng 6.1
- Số liệu: Tốc độ tăng tương đối
- Đầu ra: Biểu đồ đồ thị trên giấy

- Cần bao nhiêu dung lượng đĩa (khoảng)?: 35 Gig Docker Image

  Không gian

- Cần bao nhiêu thời gian để chuẩn bị quy trình làm việc (khoảng)?: 20 phút

  để tải xuống các tập tin hiện vật

- Cần bao nhiêu thời gian để hoàn thành thí nghiệm (khoảng)?: 2

  giờ

- Có công khai không?: Có

- Giấy phép mã (nếu có sẵn công khai)?: Giấy phép MIT

- Đã lưu trữ (cung cấp DOI)?: Có, tại liên kết sau:

  https://doi.org/10.1145/3554347 [42]

6.4.3 Mô tả

Làm thế nào để truy cập

Vật liệu hiện vật có thể được truy cập công khai bằng liên kết này https://doi.org/10.1145/3554347 [42]. Hiện vật chứa README.MD, License.txt, mã nguồn và tập lệnh cho sao chép kết quả trong bài báo này.

Phụ thuộc phần cứng

Được biết là có thể hoạt động trên kiến trúc amd64 và intel.

Phụ thuộc phần mềm

Thùng chứa Docker

Ma trận	Kích thước	NNZ
pdb1HYS	36,4K × 36,4K	4,3 triệu
jnlbrng1	40,0K × 40,0K	199K
obstclae	40,0K × 40,0K	199K
chem master1	40,4K × 40,4K	201K
rma10	46,8K × 46,8K	2,4 triệu
dixmaanl	60,0K × 60,0K	300K
không thể	62,5K × 62,5K	4.0 triệu
shyy161	76,5K × 76,5K	330K
consph	83,3K × 83,3K	6.0 triệu
denormal	89,4K × 89,4K	1,2 triệu
Baumann	112K × 112K	748K
cop20k A	121K × 121K	2,6 triệu
shipsec1	141K × 141K	3,6 triệu
majorbasis	160K × 160K	1,8 triệu
scircuit	171K × 171K	959K
mac econ fwd500	207K × 207K	1,3 triệu
pwtk	218K × 218K	11,5 triệu
Lin	256K × 256K	1,8 triệu
sinh	1,00M × 1,00M	5.0 triệu
thái1	1,00M × 1,00M	3,1 triệu
webbase1M atmosmodd	1,27M × 1,27M	8,8 triệu

Bảng 6.1: Ma trận thống kê sử dụng trong đánh giá COO CSR, CSR CSC, COO CSC

Bộ dữ liệu

Bảng 6.1 hiển thị dữ liệu được sử dụng để đánh giá COO CSR, CSR CSC, COO CSC và COO DIA có thể được tìm thấy trong container docker lấy từ SuiteSparse Matrix Bộ sưu tập [24]. Bảng 6.2 hiển thị các tenxơ được sử dụng để đánh giá sắp xếp lại Morton.

6.4.4 Cài đặt

Hướng dẫn cài đặt có thể được tìm thấy trong tài liệu README.MD ở thư mục gốc thư mục chứa hiện vật.

Tensor	Kích thước	Cách thức	NNZ
darpa	22K × 22K × 24M	3	28 triệu
fb-m	23M × 23M × 166	3	100 triệu
fb-s	39M × 39M × 532	3	140 triệu

Bảng 6.2: Các tenxơ được sử dụng để đánh giá COO3D MCOO3 _

6.4.5 Quy trình thử nghiệm

Có năm thí nghiệm được tiến hành trong công trình này, chúng tôi đã cung cấp các tập lệnh sẽ chạy cả công nghệ tiên tiến và công việc của chúng tôi cho từng quy trình chuyển đổi. Chúng tôi cũng cung cấp các tập lệnh để tạo ra các số liệu trong bài báo. Các hạt nhân được tạo ra bởi mã của chúng tôi công cụ tạo đã được tối ưu hóa và thêm vào trình điều khiển, đây cũng là trường hợp cho công cụ chuyển đổi định dạng của Taco. Chúng tôi cung cấp một tập lệnh tiện lợi (run.sh) để chạy tất cả các thí nghiệm để sao chép kết quả trong bài báo. Hướng dẫn cho thí nghiệm quy trình làm việc có thể được tìm thấy trong README.MD. Thí nghiệm sẽ tạo ra tất cả 4 biểu đồ từ bài báo.

1. coocsr.png

2.csrcsc.png

3. coodia.png

4. coocsc.png

6.4.6 Đánh giá và kết quả mong đợi

Kết quả phải phù hợp với các số liệu trong bài báo. Mặc dù có thể có một số thay đổi nhỏ trong kết quả.

6.4.7 Ghi chú

Đối với một số bài kiểm tra trong COO DIA, hãy mong đợi sự phân bổ không tốt do có 75 phần trăm số không được giới thiệu khi chuyển đổi từ các ma trận này sang DIA. Trong bài báo, chúng tôi không bao gồm các tenxơ này trong các phần kết quả và sẽ được lọc ra trước khi đồ thị được tạo ra được tạo ra.