THƯ VIỆN CỤ THỂ CHO MIỀN HIỆU SUẤT CAO

VÌ

XỬ LÝ DỮ LIỆU THỦY VĂN

qua

Kalyan Bhetwal

Một luận án

nộp một phần để hoàn thành

của các yêu cầu về mức độ

Thạc sĩ Khoa học Máy tính

Đại học Boise State

Tháng 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

Kalyan Bhetwal

Tiêu đề luận án: Thư viện chuyên biệt cho lĩnh vực hiệu suất cao để xử lý dữ liệu thủy văn

Ngày thi vấn đáp cuối kỳ: 05 tháng 5 năm 2023

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

Tiến sĩ Catherine Olschanowsky	Chủ tịch, Ủy ban giám sát
Tiến sĩ Jim Buffenbarger	Thành viên, Ủy ban giám sát
Tiến sĩ Alejandro N. Flores	Thành viên, Ủy ban giám sát

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

Dành tặng cho gia đình tôi

LỜI CẢM ƠN

Tôi muốn bày tỏ lòng biết ơn chân thành của tôi tới người hướng dẫn của tôi Tiến sĩ Catherine Olschanowsky đã hướng dẫn, hỗ trợ và động viên tôi trong suốt quá trình nghiên cứu. Kiến thức sâu rộng, kinh nghiệm và phản hồi của cô ấy vô cùng có giá trị đối với những người thành công hoàn thành thành công luận văn này.

Tôi cũng muốn cảm ơn các thành viên trong ủy ban luận án của tôi, Tiến sĩ Jim Buffenbarger và Tiến sĩ Alejandro N. Flores, vì những bình luận và gợi ý sâu sắc của họ, đã đóng góp rất nhiều vào việc cải thiện công trình này.

Cuối cùng, tôi muốn bày tỏ lòng biết ơn sâu sắc nhất tới gia đình tôi, vì tình yêu thương, sự ủng hộ và động viên không lay chuyển, nếu không có những điều đó luận văn này sẽ không thể hoàn thành đã có thể thực hiện được.

TÓM TẮT

Các nhà thủy văn phải xử lý nhiều gigabyte dữ liệu để mô phỏng thủy văn, điều này tốn thời gian và tài nguyên làm giảm hiệu suất. Các vấn đề về hiệu suất chủ yếu là do các nhà khoa học miền thích sử dụng Python, giao dịch hiệu suất cho năng suất. Trong luận án của mình, tôi chứng minh rằng sử dụng tĩnh kỹ thuật biên dịch để biên dịch Python để tạo mã C cùng với một số tối ưu hóa làm giảm thời gian và tài nguyên cho việc xử lý dữ liệu thủy văn. Tôi đã phát triển một Thư viện dành riêng cho miền (DSL) là một tập hợp con của Python và biên dịch thành Khung đa diện thưa thớt - Biểu diễn trung gian (SPF-IR), cho phép cơ hội cho việc tối ưu hóa như hợp nhất giảm đọc không có sẵn trong Python. Chúng tôi đã hợp nhất tệp I/O để thực hiện tính toán trên các khối dữ liệu nhỏ (tính toán luồng) để giảm dung lượng bộ nhớ.

Mã C mà chúng tôi tạo ra từ SPF-IR cho thấy tốc độ tăng trung bình là 2,58 lần các triển khai được tối ưu hóa bằng tay hiện có và có thể loại bỏ hoàn toàn nhịp độ yêu cầu lưu trữ rary. Người dùng DSL vẫn có thể tận hưởng sự dễ sử dụng của Python nhưng hiệu suất tốt hơn mã C.

MỤC LỤC

DANH SÁCH CÁC BẢNG

DANH SÁCH CÁC HÌ NH ẢNH

DANH SÁCH CÁC TỪ VIẾT TẮT

HPC – Máy tính hiệu suất cao

DSL – Ngôn ngữ hoặc thư viện dành riêng cho miền

SPF – Khung đa diện thưa thớt

IR – Biểu diễn trung gian

API – Giao diện lập trình ứng dụng

GB – Giga Byte

I/O – Đầu vào/Đầu ra

HUC – Mã đơn vị thủy văn

CONUS – Lục địa Hoa Kỳ

PDFG – Đồ thị luồng dữ liệu đa diện

DANH SÁCH CÁC BIỂU TƯỢNG

$\sqrt{2}$ căn bậc hai của 2

Ký hiệu lambda λ, thường được sử dụng trong phép tính lambda nhưng đôi khi cũng được sử dụng cho bước sóng

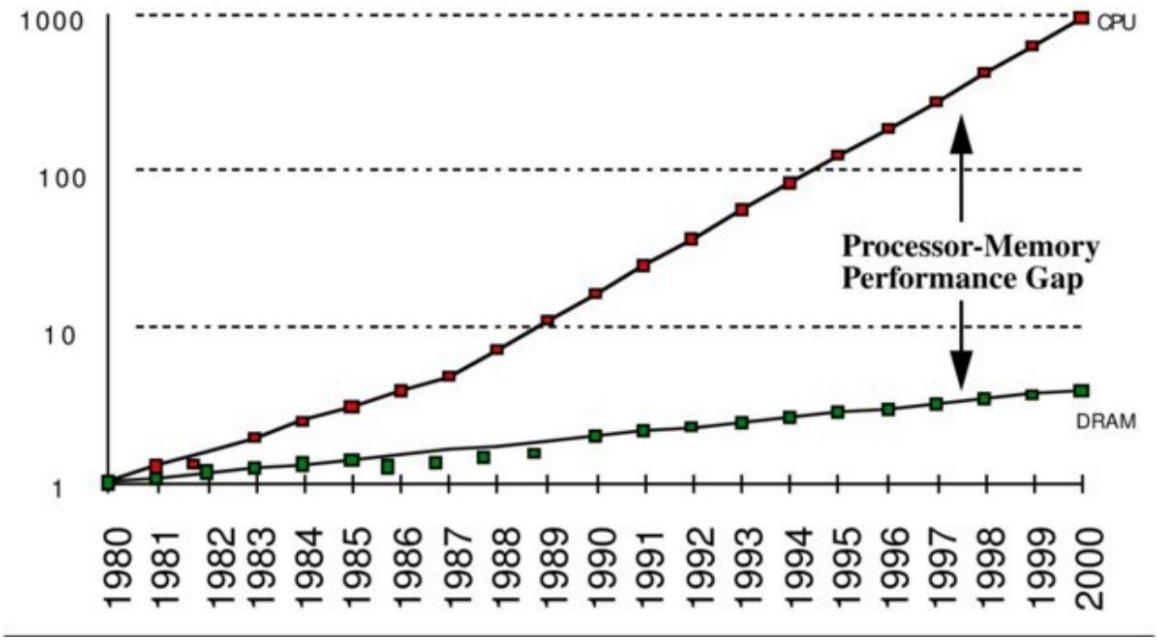
CHƯƠNG 1

GIỚI THIỆU

1.1 Giới thiệu

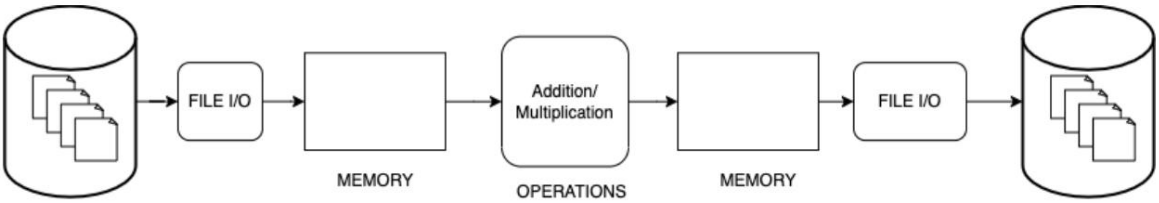
Chúng tôi sử dụng Thủy văn tính toán để hiểu rõ hơn về các hệ thống thủy văn và đóng góp ý kiến vào nhiều lĩnh vực khác nhau bao gồm nhưng không giới hạn ở các quyết định về nông nghiệp, chính sách sản xuất và nghiên cứu tác động môi trường [1, 2]. Thủy văn tính toán sử dụng khối lượng lớn áp suất đa chiều, lực ép và dữ liệu khác được thu thập qua một khoảng thời gian. Tiền xử lý và hậu xử lý các tập tin dữ liệu thủy văn bằng cách sử dụng các cách tiếp cận hiện tại quá chậm. Phải mất nhiều thời gian (giờ) để hoàn thành xử lý. Quá trình xử lý chậm chủ yếu là do phần mềm. Phần mềm để xử lý dữ liệu thủy văn được viết bằng Python và được phát triển để thuận tiện và năng suất. Chúng tôi đề xuất một giải pháp làm thay đổi tối thiểu quá trình phát triển môi trường trong khi vẫn đạt được hiệu suất mã C được tối ưu hóa.

Có một số lý do cho thời gian và nguồn lực đáng kể cần thiết để xử lý dữ liệu thủy văn. Các chương trình này tải các khối dữ liệu lớn vào bộ nhớ để xử lý, tạo ra nút thắt trong việc sử dụng bộ nhớ và làm giảm hiệu suất của hệ thống hiệu suất. Hình 1.2 cho thấy việc triển khai phần mềm thủy văn hiện có. Ví dụ, nếu kích thước tệp là 1 GB, chúng ta cần 1 GB bộ nhớ để tải tệp đó từ đĩa để xử lý. Nó tạo ra một dấu chân bộ nhớ lớn. Có một khoảng cách lớn giữa bộ nhớ và tốc độ CPU như thể hiện trong hình 1.1. Trong các phép tính này,



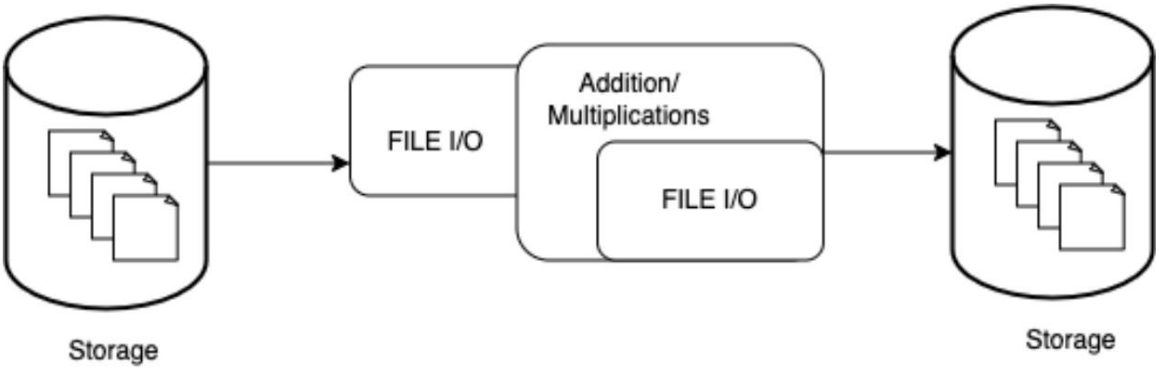
Hình 1.1: Khoảng cách hiệu suất CPU-bộ nhớ [11]

xử lý các tập tin lớn, có khả năng cao là bộ nhớ đệm bị lỗi, điều này càng làm tăng làm giảm hiệu suất.



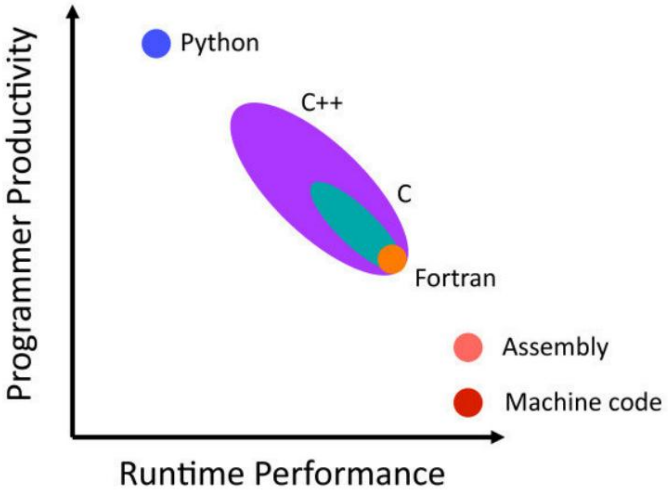
Hình 1.2: Luồng dữ liệu và dấu chân bộ nhớ của hệ thống thủy văn hiện tại

Python đổi hiệu suất lập trình lấy năng suất. Các nhà khoa học tên miền thường viết phần mềm xử lý dữ liệu bằng Python vì giao diện dễ sử dụng ở cấp độ cao. Tuy nhiên, điều này phải trả giá bằng hiệu suất thời gian chạy. Mặc dù nó tiết kiệm được nhiều thời gian để viết mã, mất nhiều thời gian để thực hiện. Grosse-Kunstleve, Ralf W., et al. [5] về công trình của họ như thể hiện trong hình 1.4 đã chứng minh rằng thời gian chạy hiệu suất của Python kém hơn các ngôn ngữ biên dịch như C. Hiệu suất



Hình 1.3: Luồng dữ liệu và dấu chân bộ nhớ của hệ thống được đề xuất

của những chương trình này là rất quan trọng vì nó sẽ tiết kiệm cả thời gian và chi phí để có được những kết quả.



Hình 1.4: Biểu đồ định tính so sánh hiệu suất thời gian chạy và năng suất của lập trình viên cho các ngôn ngữ lập trình khác nhau

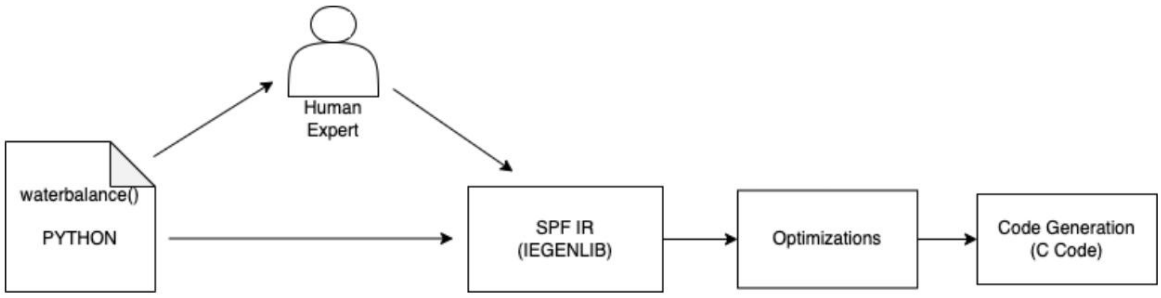
Các kỹ thuật biên dịch sẽ cải thiện hiệu suất của các chương trình này tồn tại nhưng không thể được Python sử dụng trực tiếp. Phân tích mã sâu hơn cho thấy rằng chúng ta có thể thực hiện tối ưu hóa, bao gồm cả hợp nhất giảm đọc và nhiều hơn nữa. Những tối ưu hóa này không thể trực tiếp thực hiện được trong Python. Các nhà khoa học máy tính sử dụng đa diện

biên dịch để tối ưu hóa hiệu suất trong điện toán hiệu suất cao [6].
Khung đa diện thưa thớt (SPF) mở rộng khung đa diện để hỗ trợ
giới hạn vòng lặp không affine. IEGenLib là một thư viện C++ nguồn mở để triển khai
thuật toán thao tác các tập hợp và quan hệ với các ký hiệu hàm chưa được giải thích
để kích hoạt Khung đa diện thưa thớt [16]. API tính toán cung cấp một
giao diện hướng đối tượng bao bọc SPF Intermediate Representation (IR) [12].

Trong công trình này, chúng tôi trình bày một Thư viện dành riêng cho miền (DSL) được thiết kế để tối ưu hóa
hiệu suất xử lý dữ liệu thủy văn. DSL là một tập hợp con của Python
biên dịch thành SPF-IR và cung cấp tất cả các chức năng cần thiết thường được sử dụng
trong xử lý dữ liệu thủy văn. Hình 1.2 cho thấy luồng dữ liệu cho đề xuất
hệ thống. Chúng tôi kết hợp I/O tệp với các phép tính, giúp giảm dung lượng bộ nhớ.
Điều này cho phép chúng ta tải dữ liệu từ đĩa vào bộ nhớ một cách tối ưu, thực hiện ngay lập tức
tính toán trên chúng và lưu trữ dữ liệu trở lại đĩa. Giao diện Python
cung cấp cho người dùng duy trì năng suất như nhau trong khi vẫn tận hưởng hiệu suất của
ngôn ngữ lập trình C.

1.2 Tổng quan nghiên cứu

Luận án này khám phá việc tạo ra một thư viện để cải thiện hiệu suất của thủy văn
xử lý dữ liệu trong khi giảm thiểu tác động đến năng suất của lập trình viên. Hình-
ure 1.3 cho thấy tổng quan nghiên cứu cấp cao của dự án. Thư viện xây dựng
một biểu diễn SPF-IR của phép tính mong muốn thay vì tính toán đúng
xa. Khi một lệnh chỉ ra rằng phép tính phải hoàn tất, chẳng hạn như dữ liệu
được đưa ra một tập tin, chúng tôi áp dụng tối ưu hóa khác nhau trong SPF-IR và tạo ra
mã C.



Hình 1.5: Sơ đồ luồng của hệ thống đề xuất

Những đóng góp nghiên cứu của chúng tôi có thể được tóm tắt thành ba điểm dưới đây:

Đóng góp 1: Thiết kế một đường ống trong Python phản ánh PFtools hiện tại
API. Đường ống này bao gồm I/O.

Đóng góp 2: Triển khai bộ sưu tập xử lý dữ liệu thủy văn chung
các thủ tục đang trong quá trình thực hiện.

Đóng góp 3: Chuyển đổi và đánh giá hiệu suất của các đường ống được tối ưu hóa.
Điều này bao gồm việc đánh giá chi phí chung do việc thực hiện bị chậm trễ.

1.3 Đóng góp 1: Thiết kế một đường ống trong Python để tái tạo flects API PFtools hiện tại. Đường ống này bao gồm

Đầu vào/Đầu ra.

Một đường ống trong thư viện của chúng tôi là một tập hợp các phép tính mà người dùng cuối sẽ thực hiện cho một nhiệm vụ cụ thể. Đường ống bắt đầu với đầu vào tệp và thoát khi chúng ta viết
tập tin trở lại đĩa.

1.4 Đóng góp 2: Đã triển khai một bộ sưu tập chung quy trình xử lý dữ liệu thủy văn trong đường ống.

Chúng tôi đã triển khai các thủ tục xử lý dữ liệu chung như trung bình và tổng. các thủ tục được trình bày thông qua thư viện Python.

1.5 Đóng góp 3: Chuyển đổi và đánh giá hiệu suất quản lý của các đường ống được tối ưu hóa. Điều này bao gồm một đánh giá giảm thiểu chi phí thực hiện chậm trễ.

Chúng tôi tạo mã C từ thư viện Python. Mã C có tệp đầu vào được hợp nhất với các phép tính. Mã C được tạo ra sẽ được biên dịch và thực thi.

CHƯƠNG 2

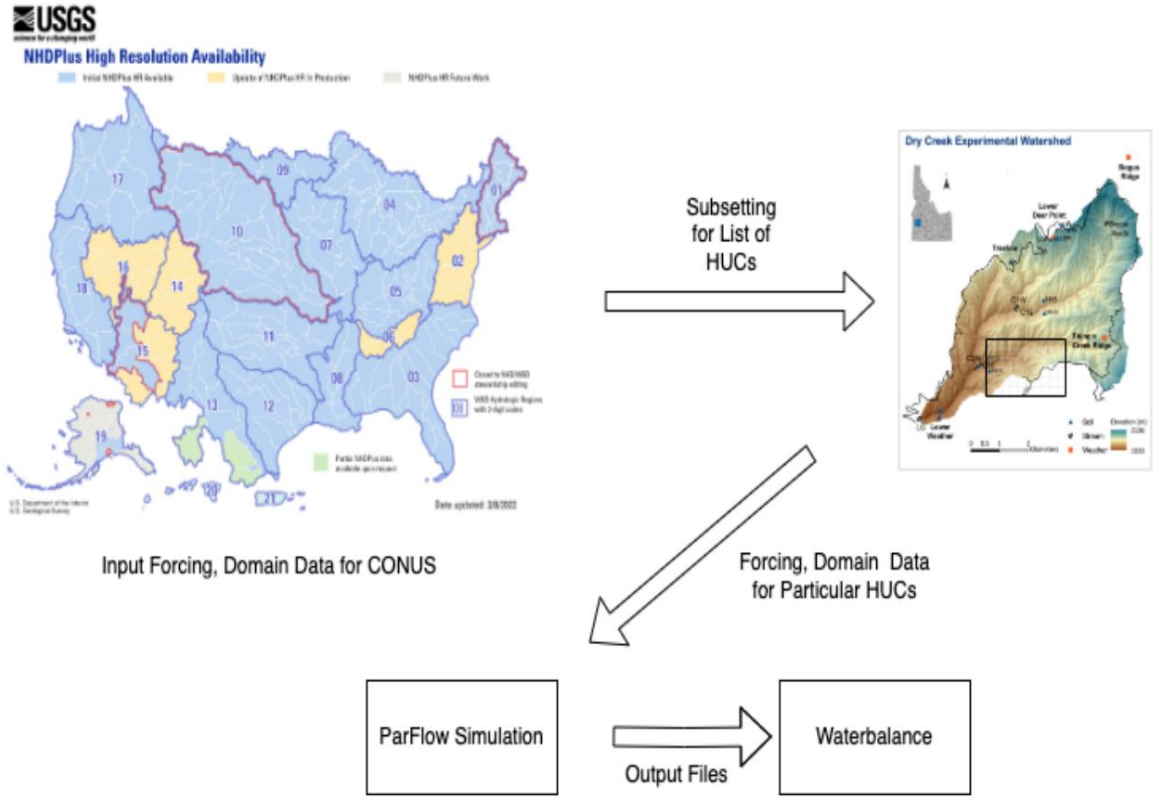
LÝ LỊCH

Trong phần này, chúng tôi thảo luận về nhiều chủ đề khác nhau cần thiết để hiểu công việc này. Chúng tôi là lấy PFtools làm ví dụ điển hình trong luận án này để kiểm tra giả thuyết của chúng tôi-sis. Đối với việc tối ưu hóa, chúng tôi biểu diễn cơ sở mã PFtools trong đa diện thưa thớt khung sử dụng API tính toán.

2.1 Công cụ PFtools

PFtools là một tập hợp các công cụ để xử lý trước và xử lý sau dữ liệu cho Mô phỏng ParFlow [9]. Các nhà khoa học miền sử dụng Python trong PFtools để thuận tiện, đi kèm với chi phí hiệu suất. PFTools đọc các tệp nhị phân parflow (PFB) dưới dạng nhập và ghi lại tệp PFB sau khi mô phỏng hoàn tất. Hình 4 cho thấy quy trình thủy văn cho thấy các bước khác nhau. Các tệp PFB lưu trữ áp suất, nhiệt độ, độ ẩm và nhiều thông tin hơn cho toàn bộ lục địa Hoa Kỳ Các tiểu bang (CONUS). Chúng tôi có một khu vực quan tâm được gọi là Đơn vị thủy văn (HUC) cho chạy mô phỏng. Chúng tôi thực hiện phân nhóm để trích xuất dữ liệu cho một khu vực cụ thể lãi suất. Phần mềm mô phỏng Parflow sử dụng kết quả của phép chia tập hợp con. Parflow là song song hàng loạt và hiệu quả cao thường được chạy trong siêu máy tính. Nhưng, các ứng dụng PFtools được viết bằng Python và không được tối ưu hóa. Luận văn này

xoay quanh việc cải thiện hiệu suất của phần mềm để xử lý trước và sau
dữ liệu xung quanh mô phỏng Parflow.



Hình 2.1: Quy trình tính toán cân bằng nước

2.2 Mô hình đa diện

Tính toán khoa học liên quan đến việc lặp lại trên không gian dữ liệu lớn bằng cách sử dụng các vòng lặp lồng nhau
để tính toán. Những mô phỏng này có thể được hưởng lợi rất nhiều từ tính song song và dữ liệu
địa phương. Một khuôn khổ đa diện là một biểu diễn toán học của các vòng lặp cho nhiều
chuyển đổi và tối ưu hóa vòng lặp tự động.

Hãy xem xét ví dụ bên dưới, trong đó có giới hạn vòng lặp afin.

```
cho(int i=0;i<M;i++)
    { cho(int j=0; j<N;j++){
       S0: s(i,j) = x(i)+j;
    }
}
```

Biểu diễn đa diện của phép tính được đưa ra như sau:

```
S0: s(i, j) = x(i) + j;

D0: Tôi = {[i, j]: 0 ≤ i < M    0 ≤ j < N} E0: {[i, j]    [0, i, 0, j, 0]}

R0: {[i, j]    [i]}

W0: {[i, j]    [i, j]}
```

Việc thực hiện các vòng lặp của chương trình trên có thể được biểu diễn trong Polyhedral mô hình như một tập hợp tất cả các kết hợp hợp lệ của bộ [i, j].

Tôi = {[i, j] : 0 ≤ i < M   0 ≤ j < N}

Không gian lặp kết hợp với sự phụ thuộc dữ liệu cung cấp thứ tự một phần của việc thực hiện các phép tính đã cho. Để thay đổi thứ tự thực hiện, chúng tôi áp dụng quan hệ với không gian lặp. Chúng ta có thể biểu thị một quan hệ cho việc hoán đổi vòng lặp cho ví dụ trên như sau:

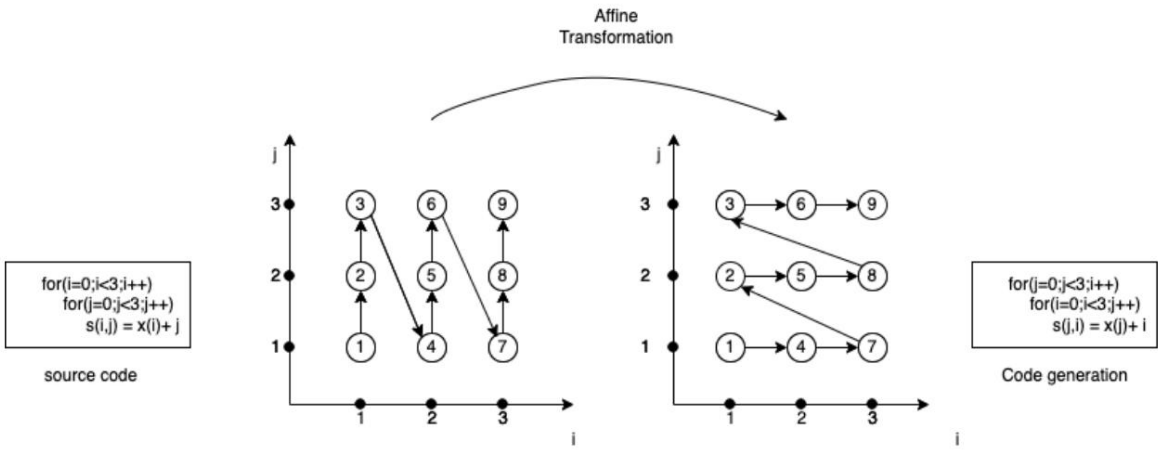
R = {[i, j]   [j, i]}

$^{\text{TÔI}'}$ = R(I)

Tổng hợp mã sau khi chuyển đổi ở trên sẽ cho ra chương trình sau.

```
cho(int j=0;i<M;i++)
    { cho(int i=0; j<N;j++){
       S0: s(i,j) = x(i) +j;
    }
}
```

Khung đa diện giúp việc chuyển từ mã sang toán học trở nên đơn giản hơn.
biểu diễn ical và quay lại mã sau khi áp dụng các phép biến đổi.
khuôn khổ đa diện bị giới hạn trong việc truy cập dữ liệu affine. Tuy nhiên, truy cập dữ liệu thưa thớt
không có bản chất affine. Khung đa diện thưa thớt khắc phục được những hạn chế
của khuôn khổ đa diện bằng cách hỗ trợ các ràng buộc phi affine trong không gian lặp
như những chức năng chưa được giải thích.



Hình 2.2: Khung đa diện

2.2.1 Miền lặp

Khung đa diện biểu diễn mỗi điểm trong các vòng lặp lồng nhau này như một mạng lưới
điểm trong đa diện. Hình 2.2 cho thấy biểu diễn đa diện của một mã. Mỗi

điểm trong mạng được biểu diễn như một tập hợp. Một mối quan hệ được áp dụng cho tập hợp cho sự biến đổi.

2.2.2 Lên lịch

Một lịch trình cung cấp thứ tự từ điển của một tuyên bố. Nó được biểu diễn như một mối quan hệ. Kích thước của lịch trình cung cấp số lượng các vòng lặp lồng nhau. Chúng ta có thể thực hiện phép biến đổi affine mà không vi phạm mối quan hệ phụ thuộc dữ liệu sau biểu diễn đa diện. Các phép biến đổi có thể giúp song song hóa và dữ liệu địa phương bằng cách tìm ra lịch trình tốt nhất cho các vòng lặp đó.

2.3 API tính toán

API tính toán là giao diện hướng đối tượng cho SPF IR [12]. API cung cấp chức năng tương tác với SPF và tạo dữ liệu đa diện đồ thị dòng chảy (PDFG). API tính toán tích hợp CHiLL [4], CodeGen+ [3], Omega [8] và IEGenLib [16]. API cung cấp một lớp tính toán để thể hiện một phép tính hoặc một loạt các phép tính. Công việc này tạo ra một đối tượng tính toán chứa các không gian dữ liệu, các câu lệnh, các phụ thuộc dữ liệu và các lịch trình thực hiện cho một phép tính cụ thể. Hình 5 cho thấy việc triển khai phép nhân vectơ thưa thớt sự trùng lặp trong API tính toán.

2.3.1 Tạo mã

Tạo mã là một chức năng quan trọng khác có trong API tính toán. Đây là giai đoạn cuối cùng và quan trọng nhất, trong đó mã thực tế được tạo ra cho mục đích đã cho. đặc tả tính toán sử dụng CodeGen+. Codegen+ sử dụng omega cho đa hình

Phép nhân ma trận-vectơ thưa thớt

```
/*Vectơ thưa thớt nhân đối với (i =
0; i < N; i++) { đối với (j=rowptr[i];
      j<rowptr[i+1];j++) { k = col[j]; y[i] += A[j] * x[k];


}}*/
Tính toán* sparseComp = tính toán mới ();
sparseComp->addDataSpace("y", "int*");
sparseComp->addDataSpace("A", "int*");
sparseComp->addDataSpace("x", "int*");
Stmt* sparseS0 = new Stmt( "y(i)
+= A(k) * x(k)", // Mã nguồn // miền lặp "{[i,j,k]:
0<=i<N && rowptr(i)<=j<rowptr(i+1)
&& k=col(j)}", "{[i,j,k]->[0,i,0,j,0,k,0]}", // Hàm lập lịch

  { {"y", "{[i,j,k]->[i]}"},
    {"A", "{[i,j,k]->[j]}"},
    {"x", "{[i,j,k]->[k]}"}}, // Dữ liệu đọc
{ {"y", "{[i,j,k]->[i]}"} } // Dữ liệu ghi

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

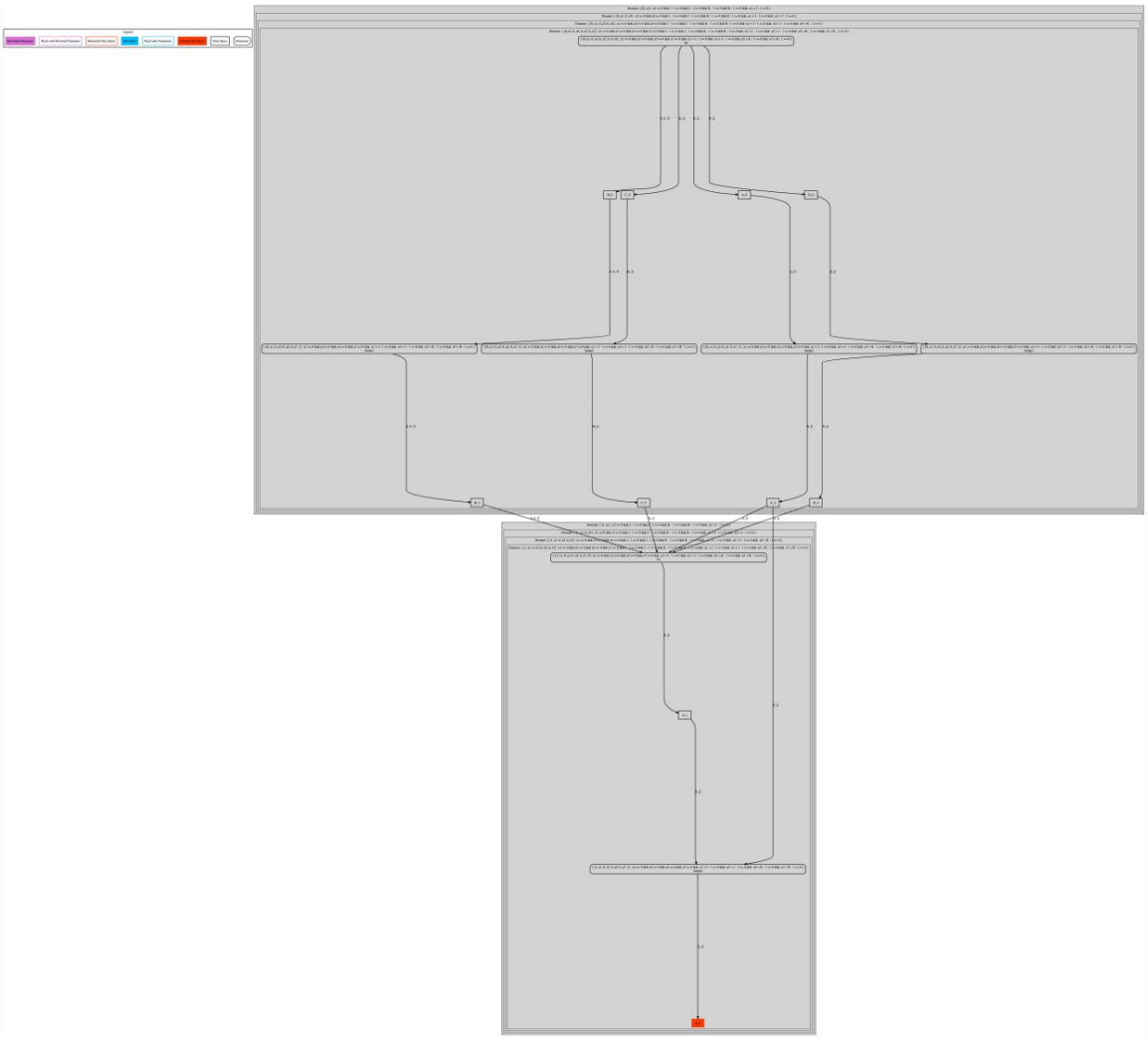
Hình 2.3: Biểu diễn phép nhân ma trận-vectơ thưa thớt trong API tính toán.

quét dra. Omega có một số hạn chế nhất định khi nó không thể xử lý được những thông tin không được diễn giải chức năng. Codegen+ khắc phục hạn chế này bằng cách thay đổi chức năng chưa được giải thích trong IEGenLib để tuân thủ Omega.

2.4 Biểu đồ luồng dữ liệu đa diện

Biểu đồ luồng dữ liệu đa diện (PDFG) là một biểu diễn trực quan dưới dạng đồ thị cho các phụ thuộc dữ liệu giữa một loạt các phép tính [15]. PDFG giúp cải thiện hiệu suất các kỹ sư đưa ra nhiều quyết định tối ưu hóa khác nhau như loại bỏ mã chết vì nó cung cấp cái nhìn toàn diện về luồng dữ liệu.

Trong luận án này, chúng tôi sử dụng PDFG để phân tích các mẫu luồng dữ liệu trong PFTools API để đưa ra quyết định tối ưu hóa. Hình 2.4 cho thấy Thời gian Tensor Matricized Tích Khatri-Rao (MTTKRP). Chúng ta có thể tạo PDFG cho phép tính (hiển thị trong Hình 2.5) cho phép chúng ta xem luồng dữ liệu và thực hiện tối ưu hóa quyết định về chúng.



Hình 2.4: PDFG cho MTTKRP

Phép nhân ma trận-vectơ thưa thớt

```
/*MTTKRP
     đối với (i = 0; i < I; i++)
          đối với (j = 0; j < J; j++)
            đối với (k = 0; k < K; k++)
               đối với (r = 0; r < R; r++)
                 A[i,r] += X[i,j,k]*B[j,r]*C[k,r];
*/
Tính toán* sparseComp = tính toán mới ();
sparseComp->addDataSpace("y", "int*");
sparseComp->addDataSpace("A", "int*");
sparseComp->addDataSpace("x", "int*");
Stmt* sparseS0 = Stmt mới (
"y(i) += A(k) * x(k)", // Mã nguồn
// miền lặp lại
"{[i,j,k]: 0<=i<N && rowptr(i)<=j<rowptr(i+1) && k=col(j)}",
"{[i,j,k]->[0,i,0,j,0,k,0]}", // Hàm lập lịch
{ {"y", "{[i,j,k]->[i]}"},
     {"A", "{[i,j,k]->[j]}"},
     {"x", "{[i,j,k]->[k]}"}}, // Đọc dữ liệu
{ {"y", "{[i,j,k]->[i]}"} } // Ghi dữ liệu
);
sparseComp->addStmt(sparseS0);
```

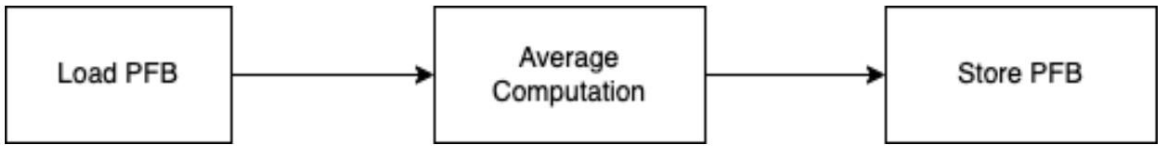
Hình 2.5: Biểu diễn phép nhân MTTKRP trong API tính toán.

CHƯƠNG 3

ĐƯỜNG ỐNG API PFTOOLS

PFtools tuân theo các mẫu chung để xử lý các tệp dữ liệu. Chúng lấy tệp PFB của
kích thước lớn (> 2,6 GB) làm đầu vào. Các tệp nhị phân Parflow lưu trữ các thông tin quan trọng
về áp suất, nhiệt độ và độ ẩm. Tệp ở định dạng nhị phân. Nó chứa
Dữ liệu 3 chiều được tổ chức theo cấu trúc khối hình chữ nhật. Bắt đầu của mỗi
khối trong tệp pfb chứa thông tin về tọa độ x, y và z bắt đầu
và kích thước của khối. Hình 3.2 cho thấy mô hình truy cập dữ liệu. Dữ liệu là
đọc từng khối một. Mỗi kích thước khối là nx $^{*}_{\text{mới}}$ * nz được đưa ra bởi khối
thông tin tiêu đề có sẵn trong tệp. Dữ liệu liên tục dọc theo trục x trong
bộ nhớ vì vậy bất kỳ tính toán nào sử dụng dữ liệu này sẽ được hưởng lợi từ vị trí bộ đệm nếu
phép tính đọc liên tục dọc theo trục x.

Tệp đầu vào được đọc đầy đủ và tải vào bộ nhớ. Các phép tính như
tính toán tổng và trung bình được thực hiện. Chúng tôi lưu trữ kết quả trở lại đĩa để
được sử dụng thêm trong các phép tính khác. Hình 3.1 minh họa cấu trúc hoàn chỉnh của
một đường ống để tính toán trung bình. Chúng ta có thể chia một đường ống thành ba đường ống cụ thể
các phần: đọc, tính toán và lưu trữ.



Hình 3.1: Đường ống API PFTools để tính toán giá trị trung bình

3.1 Đọc PFB từ đĩa

Phép tính đầu tiên trong đường ống là đọc dữ liệu từ đĩa. Danh sách 3.1 cho thấy đoạn mã để tải dữ liệu vào bộ nhớ. Vòng lặp bên ngoài đi qua mỗi khối dữ liệu. 3 vòng lặp bên trong đọc dữ liệu từ các lưới đó và lưu trữ dữ liệu trong mảng "m dữ liệu".

```
1 cho (nsg = 0;nsg<m_numSubgrids; nsg++){
2 cho (k=0; k<nz; k++){
3 cho(i=0;i<ny;i++){
4 cho(j=0;j<nx;j++){
5 m_data[chỉ mục + j] = tmp;
                        }
                    }
                }
6 7 8 9 }
```

Liệt kê 3.1: Vòng lặp để đọc dữ liệu từ tệp

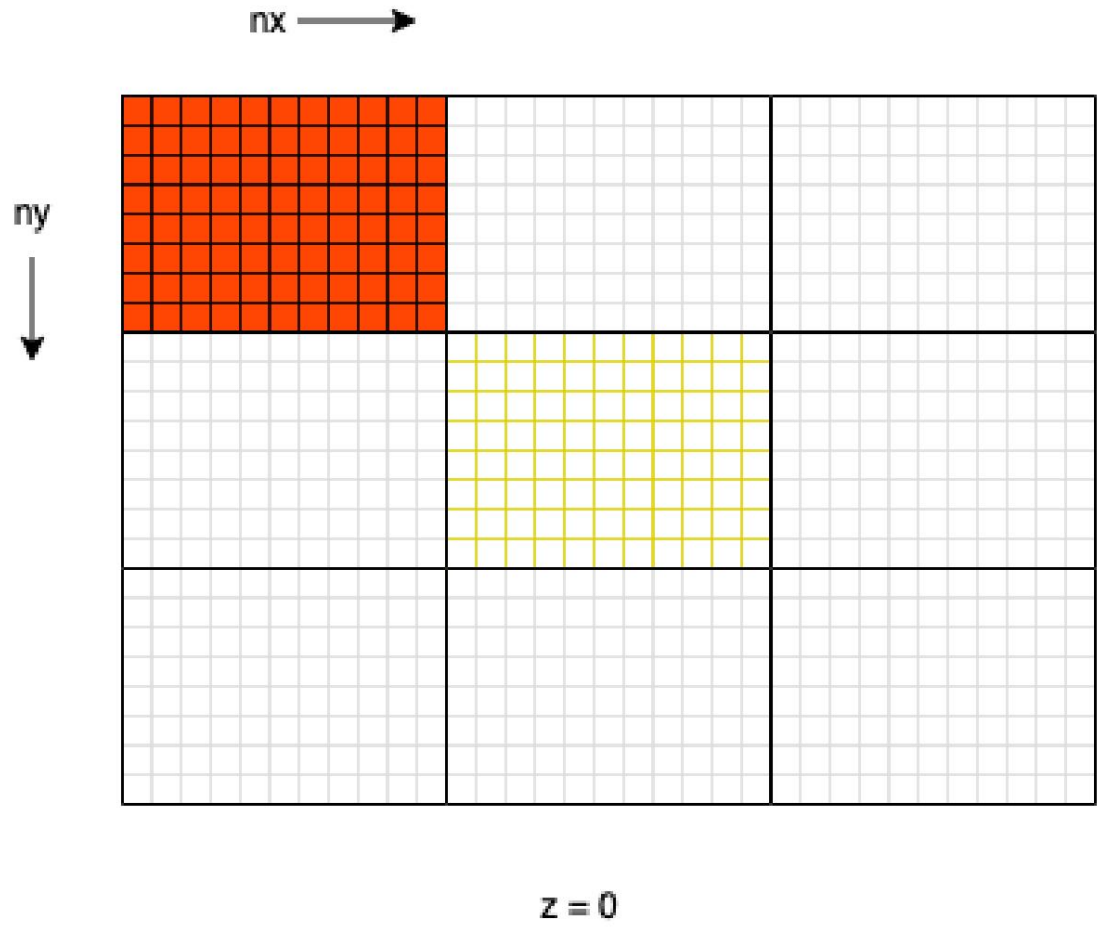
3.2 Tính toán

Chúng tôi thực hiện một số phép tính trên dữ liệu thu được từ các tệp nhị phân parflow. Một số phép tính bao gồm trung bình, tổng, tối đa và tối thiểu.

Trong luận văn này, chúng tôi thảo luận về tính toán trung bình như một ví dụ.

3.2.1 Trung bình

Hãy xem xét danh sách 3.2 được sử dụng để tính giá trị trung bình của dữ liệu dọc theo trục z. Như chúng ta đã thảo luận trong phần trước 3.1, dữ liệu liên tục dọc theo trục x. Nhưng



Hình 3.2: Tổ chức dữ liệu theo mô hình lưới

vòng lặp trong cùng là vòng lặp trung bình dọc theo trục z và chạy liên tục trên trục z. Điều này tạo ra nhiều bước nhảy trong các mẫu truy cập bộ nhớ. Trong khi đọc dữ liệu từ đĩa, chúng ta đọc dữ liệu trong các khối có kích thước nhỏ hơn nhưng trong khi tính toán giá trị trung bình, chúng ta sử dụng khối dữ liệu lớn hơn.

```
1 cho(int x = 0; x < m_nx; x++){
      for(int y = 0; y < m_ny; y++){
2 3       tổng = 0;
4           đối với (int z = 0; z < m_nz; ++z){
5                   tổng +=m_data[static_cast<long long>(z)*m_ny*m_nx+y*m_nx+x];
6         }
7         trung bình[(x+y*m_nx)] = tổng/m_nz
```

```
        }
8 9 }
```

Liệt kê 3.2: Phần mã để tính giá trị trung bình

3.3 Ghi PFB vào Đĩa

Sau khi các phép tính được thực hiện, dữ liệu được ghi lại vào đĩa trong

định dạng nhị phân parflow để sử dụng thêm trong mô phỏng. Quá trình đọc và ghi

tương tự và nhất quán. Phần 3.3 hiển thị phần mã để ghi dữ liệu vào đĩa.

Chúng tôi lặp lại các chiều z, y và x để ghi dữ liệu trở lại.

```
 1 cho(iz=calcOffset(m_nz,m_r,nsg_z); iz < calcOffset(m_nz,m_r,nsg_z+1);iz++){
 2 cho (iy = calcOffset (m_ny, m_q, nsg_y); iy < calcOffset (m_ny, m_q, nsg_y + 1); iy++) {
 3 uint64_t* buf = (uint64_t*)&(m_data[iz*m_nx*m_ny+iy*m_nx+calcOffset(
                m_nx,m_p,nsg_x)]);
 4          dài dài j;
 5          đối với (j = 0; j <x_extent; j++)
 6              uint64_t tmp = buf[j];
 7              tmp = bswap64(tmp);
 8              writeBuf[j] = *(double*)(&tmp);
 9          }
10          fwrite(writeBuf.data(),sizeof(double),x_extent,fp);
11      }
12 }
```

Liệt kê 3.3: Phần mã để ghi dữ liệu trở lại đĩa

3.4 Tóm tắt chương

Từ các mã trong danh sách 3.1, 3.2 và 3.3, chúng ta có thể thấy rằng chúng ta đang lặp lại x, y,

và các chiều z liên tục trên cùng một dữ liệu cho các phép tính của chúng tôi. Nó làm cho

cảm giác để biểu diễn các chương trình này trong một khuôn khổ Đa diện để lý luận về chúng

và áp dụng các phép biến đổi khác nhau cho chương trình được tối ưu hóa. Trong chương tiếp theo,

chúng tôi thảo luận về việc biểu diễn các phép tính này trong SPF và chuyển đổi chúng.

CHƯƠNG 4

THƯ VIỆN/NGÔN NGỮ

Người dùng được cung cấp một thư viện Python chứa các hàm thường có trong
API PFtools. Thư viện bao gồm một tập hợp các hàm cần thiết cho thủy văn
công cụ xử lý dữ liệu. Mỗi chức năng bao gồm một loạt các phép tính. Những
các chức năng được thể hiện thông qua thư viện.

Thư viện là một tập hợp con của Python có thể được sử dụng trong bất kỳ trình thông dịch Python nào.
Khi người dùng cuối viết một chương trình ví dụ để tính giá trị trung bình và chạy nó trong
Trình thông dịch Python, không có hoạt động nào xảy ra cho đến khi người dùng yêu cầu
đầu ra tệp. Trong thời gian này chúng tôi chỉ ghi lại hoạt động mà người dùng đang cố gắng thực hiện
thực hiện. Ngay khi chúng ta đạt đến điểm mà I/O thực tế được yêu cầu, một loạt
công thức tối ưu hóa được áp dụng. Công thức tối ưu hóa chính là sự kết hợp I/O tệp. Chúng tôi
sử dụng phương pháp tìm kiếm để xác định kích thước khối dữ liệu tốt nhất được tải vào bộ nhớ
từ đĩa sau đó được sử dụng trong tính toán. Ngay khi chúng ta tính toán
kết quả cho khối đã tải, chúng tôi lưu trữ nó trở lại trên đĩa. Sau khi chúng tôi hoàn tất
tối ưu hóa chúng tôi đưa ra mã C++. Mã này sau đó được biên dịch bằng bất kỳ C++ nào
trình biên dịch và được sử dụng để thực thi chương trình của người dùng.

4.1 Thư viện từ góc nhìn của người dùng cuối

Người dùng cuối được tiếp cận với một loạt các chức năng thông qua thư viện được viết bằng Python. Thư viện này bao bọc API tính toán cho phép tất cả các chức năng thể hiện đã gửi các tính toán cần thiết trong khuôn khổ Sparse Polyhedral cùng với chức năng tạo mã. Người dùng có thể nhập thư viện và gọi các hàm như theo nhu cầu của họ. Mỗi lần một hàm được gọi, thư viện chỉ theo dõi các chức năng và cuối cùng khi đầu ra được yêu cầu, thư viện kết hợp tất cả các chức năng và tạo ra mã C sau khi áp dụng một loạt các tối ưu hóa.

```
nhập khẩu iegenlib
iegenlib.readFile("snakeriver.pfb")
iegenlib.compute_mean(trục="z")
iegenlib.storeFile("snakeriver.pfb")
```

Liệt kê 4.1: Chương trình tính giá trị trung bình

4.2 Biểu diễn API tính toán

Trong chương trước, chúng ta đã xác định tất cả các chức năng tiềm năng cần được được triển khai cho các công cụ xử lý dữ liệu thủy văn. Giai đoạn đầu tiên cho thư viện của chúng tôi là đại diện cho các chức năng này trong API tính toán. Hãy xem xét một phần nhỏ của mã hiện đang được sử dụng để tính giá trị trung bình của một tệp dọc theo trục z.

```
cho(int x = 0; x < m_nx; x++){
    for(int y = 0; y < m_ny; y++){
        trung bình[x+m_nx*y] = 0;
        đối với (int z = 0; z < m_nz; z++){
            trung bình[x+m_nx*y] +=m_data[static_cast<long long>(z)*m_ny*m_nx+y*
                m_nx+x];
        }
        trung bình[(x+y*m_nx)] = trung bình[x+m_nx*y] /m_nz
    }
}
```

Biểu diễn API tính toán tương ứng là:

```
dataReads1 = iegenlib.PairVector([])
dataWrites1 = iegenlib.PairVector([("trung bình","{[x,y]->[x,y]}")])
s1 = iegenlib.Stmt("trung bình[x+m_nx*y] = 0;",
                        "{[x,y]:0<=y<m_ny && 0<=x<m_nx}",
                        "{[x,y]->[0,x,0,y,1]}",
                        dữ liệu Đọc1,
                        dữ liệu Viết1)
    parflowio_mean.addStmt(s1)


    dataReads2 = iegenlib.PairVector([("m_data","{[x,y,z]->[z,y,x]}")])
    dataWrites2 = iegenlib.PairVector([("trung bình","{[x,y,z]->[x,y]}")])

    s2 = iegenlib.Stmt("trung bình[x+m_nx*(y)]+=m_data[(dài dài)(z)*m_ny*m_nx+(y)*
        m_nx+x];",
                        "{[x,y,z]:0<=y<m_ny && 0<=x<m_nx && 0<=z<m_nz}",
                        "{[x,y,z]->[0,x,0,y,1,z,0]}",
                        dữ liệu Đọc2,
                        dữ liệu Viết2)
    parflowio_mean.addStmt(s2)

    dataReads3 = iegenlib.PairVector([("trung bình","{[y,x]->[y,x]}")])
    dataWrites3 = iegenlib.PairVector([("trung bình","{[y,x]->[y,x]}")])

    s3 = iegenlib.Stmt("mean[x+m_nx*y] = Mean[x+m_nx*y]/m_nz;",
                        "{[x,y]:0<=y<m_ny && 0<=x<m_nx}",
                        "{[x,y]->[0,x,0,y,2]}",
                        Đọc dữ liệu3,
                        dữ liệuViết3)
    parflowio_mean.addStmt(s3)
```

Để biểu diễn từng câu lệnh trong chương trình gốc trong API tính toán, chúng ta cần bốn thành phần cụ thể.

- Câu lệnh: Phần câu lệnh chứa chuỗi câu lệnh thực tế trong mã nguồn
- Không gian lặp: Không gian lặp cung cấp miền lặp của các lồng vòng lặp
- Lịch trình thực hiện: Lịch trình thực hiện cung cấp cho chúng ta thứ tự thực hiện của các tuyên bố

- Đọc dữ liệu: Đọc dữ liệu ghi cung cấp các mối quan hệ đọc dữ liệu
- Ghi dữ liệu: Ghi dữ liệu cung cấp các mối quan hệ đọc dữ liệu

4.3 Chức năng

Các hàm được cung cấp trong thư viện của chúng tôi là hàm Python thực hiện một nhiệm vụ cụ thể trong API PFtools. Mỗi hàm chứa một loạt các câu lệnh tất cả tạo thành một phép tính. Hàm bắt đầu bằng việc tạo ra một phép tính đối tượng và trả về đối tượng tính toán.

```
        def readFile(tên tệp): comp =
             iegenlib.Computation() comp.addStmt(s0)
             comp.addStmt(s1) trả
             về comp
```

4.4 Thêm phép tính

Khi chúng ta kết hợp hai hoặc nhiều hàm cho một chương trình hoàn chỉnh, chúng ta cần thêm tất cả các phép tính để tạo thành một phép tính duy nhất. Lịch trình thực hiện cần điều chỉnh để tạo thành một phép tính đơn mới với thứ tự thực hiện tăng dần.

```
   comp1 = readFile("snakeriver.pfb")
   comp2 = compute_mean(trục='z')
   comp3 = storeFile("snakeriver.pfb")
```

API tính toán cung cấp chức năng thêm phép tính vào một phép tính khác. Hãy xem xét nếu lịch trình thực hiện của “comp1” kết thúc bằng mối quan hệ “[0] [3]” sau chúng ta thêm comp2 vào comp1 lịch trình thực thi của câu lệnh đầu tiên sẽ bắt đầu bằng “{[0] [4]}”. Chúng ta có thể thực hiện điều này bằng cách sử dụng lệnh bên dưới:

```
   comp1.appendComputation(comp2, "{[0]}","{[0]->[4]}")
```

Trong hàm appendComputation, đối số đầu tiên là phép tính cần thực hiện
được thêm vào, đối số thứ ba lấy lịch trình thực hiện mới từ đó
lịch trình thực hiện mới sẽ bắt đầu cho comp2 trong tính toán kết hợp.

4.5 Công thức tối ưu hóa

Chúng tôi có cơ hội tối ưu hóa chương trình để giảm thiểu việc sử dụng bộ nhớ
và thời gian chạy. Thư viện cũng cung cấp một hàm để áp dụng một loạt các tối ưu hóa.
Chúng tôi đã khám phá mã ở lần đầu tiên và thấy rằng chúng tôi có thể sắp xếp lại các vòng lặp để tốt hơn
mẫu truy cập dữ liệu. Chúng ta cũng có thể hợp nhất nhiều vòng lặp để kết hợp các vòng lặp với nhà sản xuất
mẫu hình tiêu dùng.

4.5.1 Biến đổi vòng lặp

Trong phép biến đổi vòng lặp, chúng ta thay đổi thứ tự lồng nhau của vòng lặp. Vòng lặp bên ngoài có thể
được chuyển đến phần bên trong. Mục tiêu chính của phép biến đổi vòng lặp là
cải thiện mô hình truy cập dữ liệu sao cho dữ liệu tối đa có thể được tái sử dụng từ
bộ nhớ đệm.

```
1 nhập iegenlib 2 3

parflowio = iegenlib.Computation() 4 dataReads4
= iegenlib.PairVector([("m_data","{[x,y,z]->[z,y,x]}")]) 5 dataWrites4 =
iegenlib.PairVector([("mean","{[x,y,z]->[x,y]}")]) 6 7 s4 = iegenlib.Stmt("mean[x+y*m_nx]

+ =m_data[(dài dài)(z)*m_ny*m_nx+y*m_nx+x];
        ",
                          "{[x,y,z]:0<=y<m_ny && 0<=x<m_nx && 0<=z<m_nz}", "{[x,y,z]-
                          >[0,x,0 ,y,1,z,0]}", dataReads4,
8 9 10                    dataWrites4)
11                        12
parflowio.addStmt(s4) 13

14 in(parflowio.codeGen())
```

Liệt kê 4.2: Phần đặc tả tính toán để tính giá trị trung bình

Mã được tạo ra từ đặc tả tính toán ở trên là:

```
#undef s0
#undef s_0
#define s_0(x, y, z) nghĩa là[x+y*m_nx]+=m_data[(long long)(z)*m_ny*m_nx+y*m_nx+x
        ];
#định nghĩa s0(__x0, x, __x2, y, __x4, z, __x6) s_0(x, y, z); int t1 = 0; int t2 =


0; int t3 = 0;
int t4 = 0; int
t5 = 1; int t6 =
0; int t7 = 0;
nếu (m_nz >= 1 &&
m_ny >= 1) { đối
với (t2 = 0; t2 <=

m_nx-1; t2++) {

        đối với (t4 = 0; t4 <= m_ny-1; t4++) {
            for(t6 = 0; t6 <= m_nz-1; t6++)
            { s0(0,t2,0,t4,1,t6,0);
          }
        }
}
}

#undef s0
#undef s_0
```

Liệt kê 4.3: Mã được tạo ra để tính toán giá trị trung bình

Chúng tôi đã thiết lập trong chương 3 rằng dữ liệu là liên tục trong x-
hướng. Mã được tạo ra khi truy cập dữ liệu theo hướng z trước tiên. Vì vậy,
sẽ có nhiều bước nhảy trong bộ nhớ để truy cập dữ liệu. Chúng ta có thể hưởng lợi từ
lưu trữ đệm nếu chúng ta có thể đọc theo hướng x. Chúng tôi sử dụng phép biến đổi vòng lặp cho việc này.
API tính toán cung cấp một giao diện dễ dàng để thực hiện chuyển đổi vòng lặp. Chúng tôi
cần xác định một mối quan hệ có thể ánh xạ lịch trình thực hiện ban đầu của chúng ta tới một mục tiêu
lịch trình thực hiện. Đối với ví dụ của chúng tôi, để thay đổi thứ tự vòng lặp của các vòng lặp từ [x,y,z]
với [z,y,x] chúng ta cần áp dụng mối quan hệ dưới đây.

```
rel = iegenlib.Relation("{[0,x,0,y,0,z,0]-> [0,z,0,y,0,x,0]}")
parflowio.addTransformation(stmtIndex=0,rel=rel)
```

Liệt kê 4.4: Mối quan hệ và chức năng cho phép biến đổi vòng lặp

Trong danh sách 4.4, chúng ta có thể thấy các vòng lặp được chuyển đổi trong đó hướng x được đọc đầu tiên đó là kết quả mong muốn.

```
#undef s0
#undef s_0
#define s_0(x, y, z) nghĩa là[x+y*m_nx]+=m_data[(long long)(z)*m_ny*m_nx+y*m_nx+x
        ];
#định nghĩa s0(__x0, z1, __x2, y1, __x4, x1, __x6) s_0(x1, y1, z1);


int t1 = 0; int
t2 = 0; int t3 =
0; int t4 = 0;
int t5 = 0; int t6
= 0; int t7 = 0;

nếu (m_nx >= 1
&& m_ny >= 1) {

đối với (t2 = 0; t2 <= m_nz-1; t2++) {

        đối với (t4 = 0; t4 <= m_ny-1; t4++) {
             for(t6 = 0; t6 <= m_nx-1; t6++)
                { s0(0,t2,0,t4,0,t6,0);
          }
      }
}
}

#undef s0
#undef s_0
```

Liệt kê 4.5: Mã được tạo ra sau khi chuyển đổi vòng lặp

4.5.2 Lát gạch

Để thực hiện sự hợp nhất giữa phần đọc tệp và tính toán trung bình, chúng tôi cần phải lát gạch trước. Giới hạn trên của các vòng lặp cho tệp đọc và trung bình tính toán khác nhau. Chúng tôi phá vỡ các vòng lặp để tính toán trung bình trong các ô có kích thước giống với các vòng lặp của phần đọc tệp của mã. Xem xét danh sách 4.5 và 4.6.

Liệt kê 4.5 được sử dụng để đọc dữ liệu từ tệp trong khi liệt kê 4.6 được sử dụng để tính toán giá trị trung bình. Nếu chúng ta sắp xếp lại các vòng lặp trong danh sách 4.6 để khớp với vòng lặp trong danh sách 4.5 tức là nếu cả hai danh sách đều có cùng một vòng lặp lồng nhau "[z,y,x]", thì chúng ta có thể xếp gạch và hợp nhất chúng.

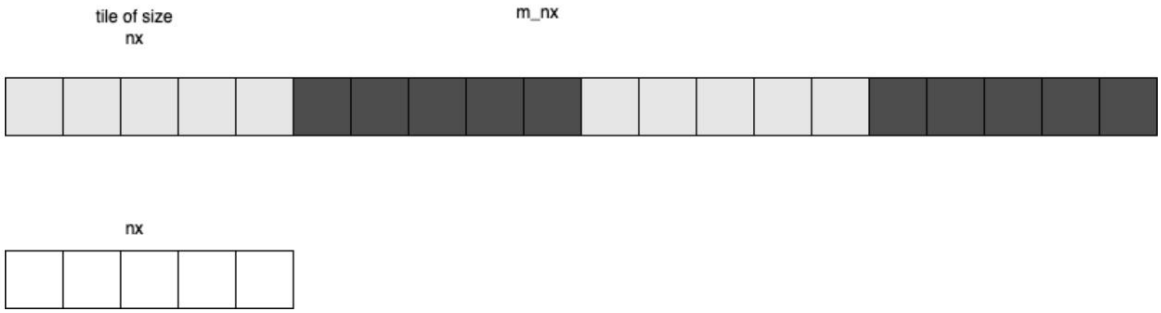
Liệt kê 4.5 và 4.6 đang lặp lại trên cùng một miền dữ liệu. Chỉ trong liệt kê 4.5 một khối dữ liệu đang được xem xét tại một thời điểm trong khi danh sách 4.6 xem xét toàn bộ dữ liệu. Việc thiết lập miền dữ liệu cho danh sách 4.5 và 4.6 là một công việc riêng biệt trong chính nó. Đối với công việc của chúng tôi, chúng tôi coi chúng là tương đương.

```
42 cho(t2 = 0; t2 <= m_numSubgrids-1; t2++) {
43 s3(3,t2,0);
44 s4(3,t2,1);
45 nếu (ny >= 1) {
46 cho(t4 = 0; t4 <= nz-1; t4++) {
47 for(t6 = 0; t6 <= ny-1; t6++) {
48 s5(3,t2,2,t4,0,t6,0);
49 cho(t8 = 0; t8 <= nx-1; t8++) {
              s6(3,t2,2,t4,0,t6,1,t8,0);
            }
          }
        }
50 51 52 53 54 }
55 }
```

Liệt kê 4.6: Đọc dữ liệu

```
1 cho(t2 = 0; t2 <= m_ny-1; t2++) {
2 cho(t4 = 0; t4 <= m_nx-1; t4++) {
3       s8(4,t2,0,t4,1);
4 cho(t6 = 0; t6 <= m_nz-1; t6++) {
5 s9(4,t2,0,t4,1,t6,0);
        }
        s10(4,t2,0,t4,2);
      }
6 7 8 9 }
```

Liệt kê 4.7: Tính giá trị trung bình



Hình 4.1: Khối có kích thước m nx được xếp thành kích thước nx

4.5.3 Sự kết hợp vòng lặp sản xuất-tiêu dùng

Trong Loop fusion, chúng tôi kết hợp hai hoặc nhiều câu lệnh trong các lồng vòng lặp khác nhau thành một lồng vòng lặp. Dữ liệu được tạo ra trong Liệt kê 4.5 có thể được sử dụng trực tiếp bởi Liệt kê 4.6. Kết quả của sự hợp nhất vòng lặp sản xuất-tiêu dùng là:

```
cho(t2 = 0; t2 <= m_numSubgrids-1; t2++) {
s3(3,t2,0);
s4(3,t2,1);
nếu (ny >= 1) {
      đối với (t4 = 0; t4 <= nz-1; t4++) {
        for(t6 = 0; t6 <= ny-1; t6++) {
          s5(3,t2,2,t4,0,t6,0);
          đối với (t8 = 0; t8 <= nx-1; t8++) {
s6(3,t2,2,t4,0,t6,1,t8,0);
s9(3,t2,2,t4,0,t6,1,t8,1);
          }
        }
      }
}
}
```

Liệt kê 4.8: Kết hợp đầu vào và tính toán trung bình

4.6 Tạo mã

API tính toán cũng cung cấp chức năng tạo mã từ đối tượng tính toán. Chúng tôi tạo mã C ngay khi chúng tôi có một đường ống hoàn chỉnh

tức là người dùng yêu cầu đầu ra. Sau đó, mã được biên dịch bằng bất kỳ chuẩn C++ nào
trình biên dịch chuẩn.

CHƯƠNG 5

KẾT QUẢ

5.1 Thiết lập thử nghiệm

Tất cả các thí nghiệm đều được chạy trên máy chủ verde được lưu trữ tại Đại học Princeton.
Máy chủ verde là cụm HPC với 96 CPU của model AMD EPYC 7402 24-Core
Bộ xử lý. 96 CPU được phân phối trong 16 nút numa với mỗi nút bao gồm
của 6 CPU.

Tất cả các mã được tạo ra đều được biên dịch bằng trình biên dịch GCC phiên bản 8.5.0 20210514.

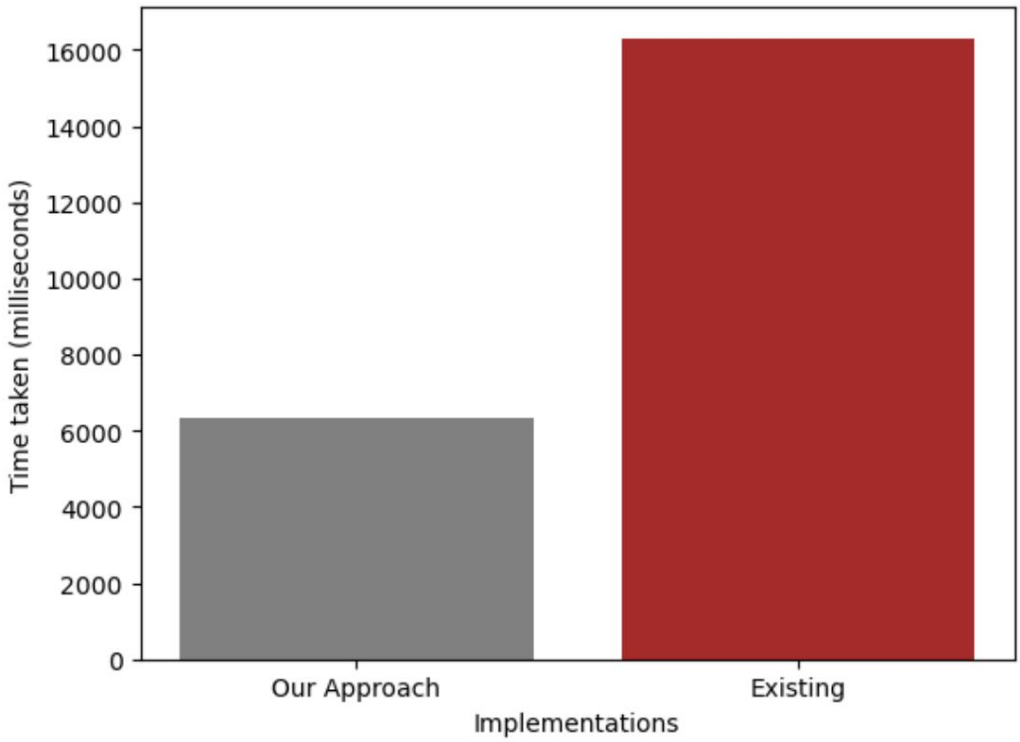
5.2 Đánh giá hiệu suất

Chúng tôi đánh giá hiệu suất của phương pháp tiếp cận của chúng tôi với các triển khai dựa trên Python hiện có
tation. Chúng tôi đã chọn năm tệp PFB có kích thước 2,6 GB để làm chuẩn mực. Chúng tôi đã ghi lại
thời gian để tính toán giá trị trung bình cho cách tiếp cận hiện tại và cách triển khai của chúng tôi.

5.2.1 Thời gian thực hiện

Bảng 5.1 cho thấy thời gian thực hiện cho cả hai triển khai. Chúng ta có thể rõ ràng
thấy rằng cách tiếp cận của chúng tôi nhanh hơn nhiều so với cách triển khai hiện tại với mức trung bình
tăng tốc 2,58 lần. Hình 5.1 cho thấy thời gian thực hiện trung bình cho cả hai
việc triển khai.

Có chi phí phát sinh trong việc tạo mã C, biên dịch và chạy chúng. Vì các chương trình này được chạy liên tục mã thực thi bị trì hoãn (codegen và thực hiện) được khấu hao.



Hình 5.1: So sánh thời gian thực hiện của việc triển khai của chúng tôi với việc triển khai hiện có

5.2.2 Sử dụng bộ nhớ

Với cách tiếp cận của chúng tôi, chúng tôi có thể loại bỏ hoàn toàn việc lưu trữ tạm thời cần thiết cho dấu chân bộ nhớ khi chúng ta thực hiện ngay các phép tính của khối của dữ liệu đang được đọc từ tệp.

Tệp	Cách tiếp cận của chúng tôi (ms)	Hiện tại (ms)	Tăng tốc
NLDAS.Press.001777 đến 001800.pfb	6254	11206	1,79
NLDAS.Press.000433 đến 000456.pfb	6427	16401	2,55
NLDAS.Press.000697 đến 000720.pfb	6543	17011	2,60
NLDAS.Press.000841 đến 000864.pfb	6459	19354	3,00
NLDAS.Press.000337 đến 000360.pfb Trung bình	5978	17570	2,94
	6332.2	16308.4	2,58

Bảng 5.1: So sánh thời gian thực hiện phương pháp của chúng tôi so với phương pháp hiện tại và tăng tốc

CHƯƠNG 6

KẾT LUẬN

6.1 Chúng tôi đã làm gì cho đến nay

Chúng tôi đã thiết lập các kỹ thuật biên dịch tĩnh cùng với tối ưu hóa như
sự kết hợp I/O tập tin có thể làm giảm đáng kể thời gian tính toán cần thiết để xử lý
các tệp dữ liệu thủy văn lớn. Chúng tôi đã thấy tốc độ tăng trung bình là 2,58 lần. Ngoài ra
sử dụng phương pháp của chúng tôi, chúng tôi có thể loại bỏ hoàn toàn việc lưu trữ tạm thời đang được
được sử dụng trong việc triển khai hiện tại.

6.2 Hướng đi trong tương lai

Công trình này đã chỉ ra rằng việc sử dụng các kỹ thuật biên dịch tĩnh có thể cải thiện đáng kể
hiệu suất của các hệ thống xử lý dữ liệu thủy văn. Công trình này là bước đệm
hướng tới việc tạo ra các công cụ có thể cải thiện đáng kể hiệu suất xử lý dữ liệu
hệ thống mà không ảnh hưởng đến năng suất của lập trình viên.

6.2.1 Hỗ trợ nhiều công cụ xử lý dữ liệu thủy văn hơn

Các công cụ xử lý dữ liệu thủy văn bao gồm một loạt các chức năng.
thư viện hiện tại chỉ triển khai một tập hợp con chức năng. Chúng ta cần triển khai

nhiều tính năng hơn cho phép người dùng cuối thực hiện các loại tính toán khác nhau liên quan để xử lý dữ liệu thủy văn.

CHƯƠNG 7

CÔNG VIỆC LIÊN QUAN

Công trình này dựa trên nghiên cứu được thực hiện trên nhiều thư viện python và miền cụ thể ngôn ngữ.

7.1 Thư viện Python

Xarray [7] là một gói python nguồn mở mở rộng hàm dữ liệu được gắn nhãn-
Tính khả thi của Pandas đối với các tập dữ liệu giống mảng N chiều (tensor). Nó cung cấp các tính năng để thao tác các tập dữ liệu đa chiều và tính toán ngoài lõi để hỗ trợ cổng song song và tính toán phát trực tuyến trên các tập dữ liệu lớn hơn bộ nhớ được hỗ trợ bởi tối [14].

Dask [14] là một thư viện linh hoạt cho tính toán song song trong Python cung cấp thực hiện đa lõi trên các tập dữ liệu lớn hơn bộ nhớ và thực hiện bị trì hoãn. Dask thể hiện các phép tính dưới dạng biểu đồ nhiệm vụ. Phép tính thực tế trên những biểu đồ này chỉ được thực hiện khi có liên quan đến đầu ra. Dask chia mảng thành nhiều những mảnh nhỏ, được gọi là khối, mỗi mảnh được cho là đủ nhỏ để vừa với bộ nhớ. Nó giúp tính toán tập dữ liệu lớn hơn bộ nhớ.

Dask và Xarray cung cấp nhiều tính năng khác nhau như song song và tải chậm. Những tính năng này các tính năng là cần thiết để tăng hiệu suất và năng suất. Nhưng chúng không cung cấp các tính năng như cải thiện vị trí dữ liệu để sử dụng phân cấp bộ nhớ đệm. Biểu diễn mã

trong khuôn khổ SPF-IR, chúng ta có thể tìm thấy những chuyển đổi pháp lý để cải thiện tính cục bộ của dữ liệu. Do đó, chúng ta có thể hưởng lợi từ hệ thống phân cấp bộ nhớ đệm. Ngoài ra, thư viện được đề xuất cung cấp thực hiện khác nhau.

7.2 Ngôn ngữ/Trình biên dịch dành riêng cho miền

Halide [13] là một ngôn ngữ lập trình được phát triển để nâng cao hình ảnh quy mô lớn và hiệu suất xử lý mảng. Halide tách các thuật toán khỏi lịch trình. Chính Ưu điểm là người dùng có thể tìm kiếm nhiều lịch trình để tìm ra lịch trình tối ưu.

PolyMages [10] cũng là một ngôn ngữ dành riêng cho miền để tối ưu hóa tự động của các đường ống xử lý hình ảnh. Polymage sử dụng khuôn khổ đa diện để chuyển tiếp hình thành và tạo mã, cung cấp các tính năng như hợp nhất phức tạp, lát gạch và tối ưu hóa lưu trữ.

Cách tiếp cận của chúng tôi kết hợp đầu vào và đầu ra để tối ưu hóa dữ liệu thủy văn hệ thống xử lý. Nó cho phép đọc luồng dữ liệu một cách tối ưu, thực hiện tính toán trên chúng và ngay lập tức ghi chúng trở lại đĩa. Nó làm giảm dấu chân bộ nhớ và cải thiện hiệu suất của các chương trình. Halide và Polymage không xem xét tập tin đầu vào/đầu ra trong đường ống tối ưu hóa của họ.

TÀI LIỆU THAM KHẢO

[1] Stephen J Burges. Xu hướng và hướng đi trong thủy văn học. Nghiên cứu tài nguyên nước, 22(9S):1S–5S, 1986.

[2] Liz Carolan, Fiona Smith, Vassilis Protonotarios, Ben Schaap, Ellen Broad, Jack Hardinges và William Gerry. Làm thế nào chúng ta có thể cải thiện nông nghiệp, thực phẩm và dinh dưỡng bằng dữ liệu mở. London, Vương quốc Anh: Viện Dữ liệu Mở, 2015.

[3] 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.

[4] Chun Chen, Jacqueline Chame và Mary Hall. CHiLL: Một khuôn khổ để biên soạn các chuyển đổi vòng lặp cấp cao. Báo cáo kỹ thuật 08-897, Đại học Nam California, tháng 6 năm 2008.

[5] Ralf W Grosse-Kunstleve, Thomas C Terwilliger, Nicholas K Sauter và Paul D Adams. Chuyển đổi fortran sang c++ tự động với fable. Mã nguồn cho sinh học và y học, 7(1):1–11, 2012.

[6] Tobias Grosser, Armin Groesslinger và Christian Lengauer. Polly—thực hiện tối ưu hóa đa diện trên biểu diễn trung gian cấp thấp. Thư xử lý song song, 22(04):1250010, 2012.

[7] Stephan Hoyer và Joe Hamman. xarray: Mảng và tập dữ liệu được gắn nhãn Nd trong python. Tạp chí phần mềm nghiên cứu mở, 5(1), 2017.

[8] 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.

[9] Benjamin NO Kuffour, Nicholas B Engdahl, Carol S Woodward, Laura E Con-don, Stefan Kollet và Reed M Maxwell. Mô phỏng dòng chảy bề mặt-dưới bề mặt kết hợp với parflow v3. 5.0: khả năng, ứng dụng và phát triển liên tục của mô hình thủy văn tích hợp, song song hàng loạt, nguồn mở. Phát triển mô hình khoa học địa chất, 13(3):1373–1397, 2020.

[10] Ravi Teja Mullapudi, Vinay Vasista và Uday Bondhugula. Polymage: Tối ưu hóa tự động cho các đường ống xử lý hình ảnh. ACM SIGARCH Computer Architecture News, 43(1):429–443, 2015.

[11] David A Patterson và John L Hennessy. Tổ chức và thiết kế máy tính, phiên bản ARM: giao diện phần mềm phần cứng. Morgan Kaufmann, 2016.

[12] 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. IEEE, 2021.

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

[14] Matthew Rocklin. Dask: Tính toán song song với các thuật toán bị chặn và lập lịch tác vụ. Trong Biên bản báo cáo của hội nghị khoa học về python lần thứ 14, tập 130, trang 136. Citeseer, 2015.

[15] Alina Sbˆırlea, Jun Shirako, Louis-No¨el Pouchet và Vivek Sarkar. Tối ưu hóa đa diện cho ngôn ngữ đồ thị luồng dữ liệu. Trong Ngôn ngữ và trình biên dịch cho máy tính song song: Hội thảo quốc tế lần thứ 28, LCPC 2015, Raleigh, NC, Hoa Kỳ, ngày 9-11 tháng 9 năm 2015, Bài báo đã chọn đã sửa đổi 28, trang 57–72. Springer, 2016.

[16] Michelle Mills Strout, Geri Georg và Catherine Olschanowsky. Thao tác tập hợp và 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.