Thiết kế nhân ma trận thưa với véctơ trong tính toán song song và ứng dụng

Ngô Thị Nhạn

Trường Đại học Khoa học Tự nhiên
Khoa Toán - Cơ - Tin học
Luận văn Thạc sĩ ngành: Bảo đảm toán học cho máy tính và hệ thống tính toán;
Mã số: 60 46 35
Người hướng dẫn: PGS. TS. Nguyễn Hữu Điển
Năm bảo vệ: 2011

Abstracts. Trình bày tổng quan về xử lý song song, thuật toán song song và giới thiệu lập trình song song với MPI sử dụng Visual của Microsoft. Trình bày về các thuật toán thiết kế cho nhân ma trận thưa với véc tơ song song. Trình bày một số kết quả thực nghiệm trên một số bộ dữ liệu cho chương trình nhân ma trận thưa với véc tơ song song.

Keywords. Nhân ma trận; Véctơ; Tính toán song song; Toán tin

Content

LỜI MỞ ĐẦU

Việc nghiên cứu thiết kế các máy tính song song, và các thuật toán song song cũng như các ngôn ngữ lập trình hỗ trợ lập trình song song bắt đầu được quan tâm từ những năm 70, cho đến nay các ứng dụng của chúng đã lan rộng khắp các lĩnh vực của đời sống như đánh giá khả năng rủi ro về tài chính: dùng để mô hình hoá các xu hướng trên thị trường… Hỗ trợ quyết định như phân tích thị trường, dự báo thời tiết… Trí tuệ nhân tạo như thiết kế robot… Xử lý ảnh ứng dụng trong công nghệ nhận dạng… Điều khiển tự động… Mô phỏng, trình chiếu hệ thống lớn … Bài toán nh ân ma tr ận th ưa v ới v éc t ơ, hay gặp trong các lời giải lặp của hệ phương trình tuyến tính, hệ phương trình giá trị riêng, véc tơ riêng... Khi ma trận kích thước lớn, việc thực hiện nhân ma trận với véc tơ lặp đi lặp lại nhiều lần yêu cầu khối lượng xử lý lớn mà xử lý tuần tự không đáp ứng được vì vậy việc nhân ma trận thưa với véc tơ song song là cần thiết và có vai trò quan trọng.

Trong phạm vi luận văn này trình bày ba phần chính, C**hương 1** trình bày tổng quan về xử lý song song, thuật toán song song và giới thiệu lập trình song song với MPI sử dụng Visual của Microsoft; C**hương 2** trình bày về các thuật toán thiết kế cho nhân ma trận thưa với véc tơ song song; C**hương 3** trình bày một số kết quả thực nghiệm trên một số bộ dữ liệu cho chương trình nhân ma trận thưa với véc tơ song song.

Chương 1 - TỔNG QUAN VỀ XỬ LÝ SONG SONG

Chương này trình bày một số kiến thức tổng quan về xử lý song song bao gồm hệ thống song song, các mô hình lập trình song song, nguyên lý thiết kế thuật toán song song và kiến trúc cụm máy tính của trung tâm tính toán hiệu năng cao. Luận văn đã giới thiệu về giao diện truyền thông điệp MPI nhằm mục đích áp dụng lập trình MPI với ngôn ngữ *C* cho bài toán nhân ma trận thưa với véc tơ song song.

1.1 Hệ thống song song

Kiến trúc xử lý song song

a) Máy tính song song phân chia theo cách thức thực hiện chương trình

Có nhiều cách để phân loại máy tính song song, người ta thường sử dụng cách phân loại máy tính song song của M.J. Flynn (1966). Cách phân loại này dựa vào sự phân phối dữ liệu và phân phối các lệnh trên mỗi bộ xử lý.

Bốn cấu trúc máy tính song song được phân loại bởi Flynn đó là:

- **Mô hình SISD** (đơn luồng lệnh, đơn luồng dữ liệu)

Máy tính loại SISD chỉ có một CPU, ở mỗi thời điểm thực hiện một chỉ lệnh và chỉ đọc, ghi một mục dữ liệu. Đây chính là mô hình máy tính truyền thống kiểu von Neumann .

- **Mô hình SIMD** (đơn luồng lệnh, đa luồng dữ liệu)

Máy tính loại SIMD có một đơn vị điều khiển để điều khiển nhiều đơn vị xử lý thực hiện theo một luồng các câu lệnh. Đây chính là mô hình máy tính phổ biến có trên thị trường như: ILLIAC IV, DAP và Connection Machine CM-2.

- **Mô hình MISD** (đa luồng lệnh, đơn luồng dữ liệu)

Máy tính MISD có thể thực hiện nhiều chương trình (nhiều lệnh) trên cùng một mục dữ liệu, nên còn được gọi là MPSD - đa chương trình, đơn luồng dữ liệu. Hoạt động của máy tính theo kiến trúc loại này giống như hệ tuần hoàn nên còn được gọi là hệ tâm thu.

- **Mô hình MIMD** (đa luồng lệnh, đa luồng dữ liệu)

Máy tính loại MIMD còn gọi là đa bộ xử lý, trong đó mỗi bộ xử lý có thể thực hiện những luồng lệnh (chương trình) khác nhau trên các luồng dữ liệu riêng.

Hầu hết các hệ thống MIMD đều có bộ nhớ riêng và cũng có thể truy cập vào được bộ nhớ chung (toàn cục) khi cần, do vậy giảm thiểu được sự trao đổi giữa các bộ xử lý trong hệ thống.

Đây là kiến trúc phức tạp nhất, nhưng nó là mô hình hỗ trợ xử lý song song cao nhất và đã có nhiều máy tính được sản xuất theo kiến trúc này như: BBN Butterfly, Aliant FX, iSPC của Intel, …

b) Máy tính song song phân chia theo kiến trúc phần cứng

Kiến trúc máy tính là một phần rất quan trọng quyết định hiệu quả của công việc, đối với một máy tính song song có một số loại kiến trúc phần cứng cơ bản sau:

- **Máy tính song song với bộ nhớ chia sẻ**

Trong máy tính song song với bộ nhớ chia sẻ, bộ nhớ là chung cho tất cả các bộ xử lý. Bộ nhớ được phân thành các mô đun nhớ, mỗi mô đun này có kênh vào ra riêng. Mỗi bộ xử lý đọc dữ liệu mà nó cần trong bộ nhớ chung, xử lý chúng, rồi lại ghi kết quả vào bộ nhớ.

- **Máy tính song song với bộ nhớ phân tán**

Trong mô hình này mỗi CPU có bộ nhớ riêng, mỗi CPU truy nhập đến bộ nhớ riêng của mình với tốc độ nhanh, còn truy cập đến bộ nhớ của CPU khác thì chậm hơn. Trong mô hình này mỗi bộ xử lý ngoài việc sử dụng dữ liệu trong bộ nhớ riêng, nó còn

có thể đọc trên một hoặc nhiều kênh truyền thông từ các bộ xử lý khác mà nó cần, thực hiện xử lý, rồi lại có thể truyền kết quả cho bộ xử lý khác.

- **Mô hình hỗn hợp**

Mô hình này là sự kết hợp giữa các mô hình đã trình bày ở trên, nó dung hoà được các ưu nhược điểm của hai kiến trúc trên, các bộ xử lý được liên kết với nhau thông qua một mạng giao tiếp tạo thành một hệ thống liên cụm, mỗi cụm có kiến trúc dùng chung.

1.2 Các mô hình lập trình song song

1.3.1 Mô hình chia sẻ bộ nhớ

Trong mô hình này, các bộ xử lý đều được kết nối tới một bộ nhớ chung, thông qua các phương tiện phần cứng hoặc phần mềm. Có các cơ chế khác nhau như khóa/semaphores sử dụng để kiểm soát truy cập vào bộ nhớ chung đó.

1.3.2 Mô hình tuyến

Trong mô hình tuyến (hay mô hình luồng) của lập trình song song, một tiến trình đơn có thể bao gồm nhiều tuyến thực hiện đồng thời. Mỗi luồng có dữ liệu cục bộ, nhưng đồng thời chia sẻ toàn bộ tài nguyên của tiến trình chủ. Điều này tiết kiệm các chi phí liên quan đến việc sao chép các tài nguyên của chương trình cho mỗi tuyến.

1.3.3 Mô hình song song dữ liệu

Mô hình song song dữ liệu hay gặp khi phần lớn công việc song song tập trung vào việc thực hiện các tác vụ trên một tập dữ liệu. Tập dữ liệu thường được tổ chức thành một cấu trúc thông dụng, như mảng hay khối lập phương. Một tập các tác nhiệm làm việc trên cùng một cấu trúc dữ liệu, tuy nhiên, mỗi tác nhiệm làm việc trên một phân vùng khác nhau của cùng một cấu trúc dữ liệu. Các tác nhiệm thực hiện cùng một tác vụ trên phân vùng làm việc của chúng.

1.3.4 Mô hình truyền thông điệp

Trong mô hình truyền thông điệp, các tiến trình chia sẻ với nhau qua các kênh truyền thông. Kênh là khái niệm trừu tượng của đường truyền thông vật lý giữa các tiến trình. Các kênh được truy cập bởi hai phương thức gửi và nhận thông điệp: *send()* và *receive()*. Khi một tiến trình gửi đi một thông điệp vào kênh truyền và có một tiến trình khác yêu cầu nhận thông điệp đó thì diễn ra sự trao đổi dữ liệu và sự trao đổi được hoàn tất khi dữ liệu đã được chuyển từ nơi gửi tới nơi nhận.

1.4 Nguyên lý thiết kế thuật toán song song

Có thể có nhiều thuật toán song song khác nhau cùng giải quyết một bài toán, tùy thuộc vào cách dữ liệu được truy xuất, cách phân rã dữ liệu thành những tác vụ, cách ánh xạ các tác vụ này vào các bộ vi xử lý và cách đồng bộ các tiến trình.

1.4.1 Các giai đoạn của thiết kế thuật toán song song

Có ba phương pháp thiết kế thuật toán song song:

- Song song hoá những thuật toán tuần tự đã có
- Thiết kế thuật toán song song mới trên cơ sở thuật toán song song đã có
- Thiết kế thuật toán song song hoàn toàn mới thích ứng với những cấu trúc song song.

Dù sử dụng phương pháp nào thì quá trình thiết kế cũng thường bao gồm các giai đoạn sau:

Phân hoạch bài toán

Mục tiêu của giai đoạn phân hoạch là tìm kiếm khả năng thực hiện song song của bài toán. Phân hoạch là thao tác phân tích một tính toán thành nhiều tác vụ nhỏ. Giai đoạn

thiết kế này thường độc lập với kiến trúc máy và đảm bảo thuật toán đặt ra có độ tương thích cao với các hệ máy tính song song.

Truyền thông

Sau khi bài toán được chia nhỏ thành các tác vụ thì chúng ta phải xác định mối quan hệ giữa các tác vụ đó rồi dựa trên các quan hệ này để chỉ ra cách truyền thông thích hợp giữa các tác vụ.

Tích hợp

Mục tiêu của giai đoạn này là đưa ra cách tích hợp một số tác vụ thích hợp thành một tác vụ đủ lớn cho một máy tính thực hiện. Việc lựa chọn cách tích hợp tốt dựa trên việc cân nhắc một số vấn đề như: các đòi hỏi về truyền thông, kiến trúc thực thi thuật toán, khả năng và chi phí thực hiện.

Ánh xạ

Sau khi đã tích hợp lại thành những tác vụ thích hợp, ta cần ánh xạ các tác vụ này vào kiến trúc máy tính thực thi thuật toán. Giai đoạn này chỉ rõ nơi mà mỗi tác vụ được thực hiện. Nói cách khác, mỗi thao tác sẽ được gán cho một bộ xử lý riêng để số bộ xử lý được sử dụng lớn nhất và chi phí truyền thông nhỏ nhất.

1.4.2 Đánh giá độ phức tạp của thuật toán song song

Độ phức tạp về thời gian là tiêu chí quan trọng trong đánh giá thuật toán song song. Nói chung, độ phức tạp về thời gian của thuật toán song song ta cần xét đến hai loại thao tác: thao tác tính toán cơ bản và thao tác định tuyến dữ liệu.

Thời gian thực hiện song song, ký hiệu là t_p gồm hai phần t_{comp} và t_{comm}

$$t_p = t_{comp} + t_{comm}.$$

Trong đó, t_{comp} là thời gian tính toán và t_{comm} là thời gian truyền thông dữ liệu.

1.5 Lập trình song song với MPI

Giao diện truyền thông điệp chuẩn MPI (*MessagePassing Interface*) là một giao diện chuẩn cho phép nhiều máy tính giao tiếp với nhau, được sử dụng trong các cụm máy tính và siêu máy tính. MPI là giao thức độc lập ngôn ngữ sử dụng cho các máy tính song song. MPI là một giao diện lập trình ứng dụng truyền thông điệp với mục đích là đem lại hiệu năng cao, khả năng mở rộng và linh hoạt.

Các khả năng của MPI

Giao diện MPI là một phương tiện để cung cấp các chức năng topo ảo, sự đồng bộ hoá và giao tiếp giữa tập hợp các tiến trình (các nút/máy chủ/máy tính) theo cách độc lập với ngôn ngữ, với cú pháp đặc tả ngôn ngữ, cộng thêm một số đặc điểm khác.

Chức năng của thư viện MPI bao gồm (nhưng không hạn chế) các hoạt động gửi nhận điểm-điểm, lựa chọn topo tiến trình logic dạng hình học phẳng hay đồ thị, trao đổi dữ liệu giữa các cặp tiến trình, phối hợp kết quả từng phần của quá trình tính toán, các nút đồng bộ cũng như các thông tin về mạng …

1.6 Kiến trúc cụm máy tính

Hệ thống song song của trung tâm tính toán hiệu năng cao có hai dạng kiến trúc bó như sau:

- Kiến trúc bó IBM 1600.
- Kiến trúc bó IBM 1350.

Chương 2 – THUẬT TOÁN SONG SONG NHÂN MA TRẬN THƯA VỚI VÉC TƠ

2.1 Ma trận thưa

2.1.1 Đặt vấn đề

Bài toán

Cho ma trận thưa A cấp $m \times n$ và véc tơ dày v độ dài n. Kết quả tính toán là véc tơ u độ dài m, $u = Av$.

Trong chương này trình bày bài toán nhân ma trận thưa với véc tơ song song, tính thưa của ma trận có thể là bất thường, không có qui tắc. Nhưng thật may mắn trong bài toán nhân ma trận với véc tơ thì ma trận là không thay đổi trong quá trình tính toán.

Các bài toán về ma trận thưa đã được đề cập, cùng với các bộ dữ liệu từ những bài toán thực tế đã được tập hợp bởi các nhóm nghiên cứu như: Matrix maket, Harwell-Boeing ngày nay gọi là Rutherford-Boeing… Mục tiêu cho việc xây dựng các bộ sưu tập này đó là các nhà nghiên cứu kiểm nghiệm các bộ dữ liệu, cho các thuật toán khác nhau trên các máy khác nhau, để có thể sử dụng như một tập phổ biến các bộ thử nghiệm cho trường hợp đặc biệt như ma trận thưa.

Các kết quả đã đạt được

Trong những năm gần đây, đã có nhiều kết quả đạt được từ bài toán nhân ma trận thưa với véc tơ song song. Có hai phương pháp đã được sử dụng cho phân hoạch dữ liệu của bài toán, phương pháp phân hoạch dữ liệu một chiều và phương pháp phân hoạch hai chiều. Các kết quả đã đạt được từ hai phương pháp này cho kết quả trong tính toán, cân bằng tải tính toán khá tốt, và cúng đã được quan tâm cho cả trong truyền thông. Hai véc tơ v và u cũng đã được phân phối cho các bộ xử lý nhưng thường áp đặt bằng cách xem chúng là các ma trận vuông.

Trong luận văn trình bày phương pháp phân hoạch hai chiều phân phối dữ liệu cho ma trận, sau đó thực hiện phân phối các véc tơ bằng phương pháp phân hoạch dựa theo biên địa phương.

Để khai thác tính thưa của ma trận thưa, trước tiên chúng ta phải lựa chọn được cấu trúc dữ liệu thích hợp, thường chỉ lưu các phần tử khác không của ma trận. Tiếp theo, đi vào thiết kế thuật toán nhân ma trận thưa với véc tơ song song với mục tiêu đề ra là cực tiểu hoá chi phí truyền thông và đạt cân bằng tải tốt.

2.1.2 Cấu trúc dữ liệu cho ma trận thưa

Lợi ích chính của việc khai thác tính thưa là làm giảm bộ nhớ sử dụng (các phần tử 0 không được lưu trữ) và thời gian tính toán (các phép toán với 0 được bỏ qua hoặc được làm cho đơn giản). Có các cấu trúc hay dùng đó là:

- Cấu trúc lược đồ bộ ba (coordinate scheme)
- Cấu trúc lưu trữ hàng nén (CRS-Compressed Row Storage).
- Cấu trúc lưu trữ cột nén (CCS-Compressed Column Storage).
- Cấu trúc lưu trữ hàng (cột) nén có gia số (ICRS-Incremental Compressed Row Storage).

2.2 Nhân ma trận thưa với véc tơ song song

2.2.1 Thuật toán song song

Để thực hiện nhân ma trận thưa với véc tơ song song hiệu quả đòi hỏi phân phối dữ liệu phải phù hợp với tính toán trên đó. Cụ thể, yêu cầu phân phối ma trận và các véc tơ đều khắp trên các bộ xử lý của máy tính song song, có nghĩa là mỗi bộ xử lý sẽ có số phần tử khác không gần như nhau. Phương pháp phân hoạch dữ liệu cho bài toán nhân ma trận thưa với véc tơ song song trình bày ở đây gồm hai bước như sau: trước tiên, ma trận sẽ được phân phối, ở đây sẽ xác định được truyền thông và cân bằng tải tính toán. Sau đó các véc tơ đầu vào và đầu ra sẽ được phân phối, ở bước này xác định được cân bằng tải truyền thông. Sự độc lập, riêng rẽ của hai bước sẽ giúp cho chúng ta dễ dàng hơn trong việc tối ưu phân phối dữ liệu.

Trong thuật toán song song nhân ma trận thưa với véc tơ tổng quát, thành phần véc tơ v_j cần thiết cho tính toán $a_{ij}.v_j$ phải đạt được trước khi bắt đầu bước (2). Điều này được thực hiện trong bước truyền thông (1). Bộ xử lý mà phải nhận v_j được nhận biết từ ma trận thưa địa phương mà cần thành phần v_j này. Thành phần v_i phải nhận về mỗi bộ xử lý mà cần nó, nó có thể được sử dụng nhiều lần cho các phần tử khác 0 địa phương a_{ij} trong cùng cột j của ma trận. Nếu cột j chứa ít nhất một phần tử khác 0 địa phương thì v_j là cần thiết; ngược lại v_j không cần thiết. Do đó, để thuận tiện ta xác định tập chỉ số J_s các cột khác rỗng địa phương, tương tự tập chỉ số hàng khác rỗng I_s. Chúng ta nhận về v_j nếu và chỉ nếu $j \in J_s$, thực hiện ở bước (1) của thuật toán. Tập J_s có thể được biểu diễn bằng mảng *colindex* độ dài $|J_s|$, tương tự *rowindex* cho tập I_s. Chúng ta xem các mảng *rowindex* và *colindex* như phần cấu trúc dữ liệu cho ma trận thưa địa phương. Tổng thành phần u_{is} mà khác 0 phải được gửi tới u_i nếu hàng i địa phương là không rỗng, tức là, nếu $i \in I_s$, thực hiện trong bước (3). Cuối cùng, bộ xử lý chịu trách nhiệm tính u_i sẽ cộng các giá trị khác 0 đã nhận được u_{it}, $0 \le t < P,\ t \ne s$, thực hiện ở bước (4).

2.2.2 Phân phối ma trận

Giả sử ma trận thưa A có kích thước $m \times n$ với $m, n \ge 1$, các phần tử a_{ij}, với $0 \le i < m$ và $0 \le j < n$. Véc tơ đầu vào v là véc tơ độ dài n và véc tơ đầu ra u là véc tơ độ dài m. Chúng ta không tính khả năng thưa có thể có của các véc tơ. Máy tính song song có số bộ xử lý dạng $P = 2^q$ bộ xử lý, $q \ge 0$, mỗi bộ xử lý có bộ nhớ cục bộ riêng.

Một phân hoạch k chiều của A là một tập $\{A_0, ..., A_{k-1}\}$ khác rỗng, các tập con của A rời nhau đôi một và thỏa mãn $\bigcup_{r=0}^{k-1} A_r = A$.

Dưới đây là một số định nghĩa:

Định nghĩa 2.1. Cho A là ma trận thưa $m \times n$ và cho $A_0, ..., A_{k-1}$ là các tập con của A, $k \ge 1$.

p_i là số tập con mà có phần tử khác không trong hàng i của ma trận A, với $0 \le i < m$,

q_j là số tập con mà có phần tử khác không trong cột j của ma trận A, với $0 \le j < n$.

Ký hiệu $p'_i = \max(p_i - 1, 0)$, $q'_j = \max(q_j - 1, 0)$. Khi đó dung lượng truyền thông cho k tập $A_0, ..., A_{k-1}$ xác định như sau:

$$V(A_0, ..., A_{k-1}) = \sum_{i=0}^{m-1} p'_i + \sum_{j=0}^{n-1} q'_j$$

Đặc biệt hàm dung lượng truyền thông V vẫn xác định khi k tập con rời nhau đôi một mà không tạo nên phân hoạch k chiều. Nếu $k = P$ và các tập con tạo nên phân hoạch P chiều và nếu chúng ta phân mỗi tập con cho một bộ xử lý, thì $V(A_0, ..., A_{P-1})$ chính là dung lượng truyền thông trong thuật toán song song nhân ma trận thưa với véc tơ ở trên.

Định lý 2.2. Cho A là ma trận cấp $m \times n$ và $A_0, ..., A_{k-1}$ là các tập con của A, $k \ge 2$. Khi đó dung lượng truyền thông giữa k tập,

$$V(A_0, ..., A_{k-1}) = V(A_0, ..., A_{k-3}, A_{k-2} \cup A_{k-1}) + V(A_{k-2}, A_{k-1})$$

Định lý trên là khái quát cho các tập con tùy ý, nhận xét bởi Catalyurek và Aykanat, trong trường hợp mà mỗi tập con A_r bao gồm các cột (nguyên vẹn) của ma trận. Định lý

chỉ ra rằng để xem xét truyền thông tăng thêm từ việc phân chia một tập con của ma trận, chúng ta chỉ cần xem xét trên chính tập con này.

Chúng ta cũng định nghĩa hàm tính lượng công việc tính toán cực đại của một bộ xử lý khi nhân ma trận cục bộ với véc tơ. Để đơn giản chúng ta biểu diễn bằng số lượng phép nhân, bỏ qua các phép cộng.

Định nghĩa 2.3. Cho *A* là ma trận cấp $m \times n$ và $A_0,...,A_{k-1}$ là các tập con rời nhau đôi một của A, với $k \geq 1$. Thì lượng tính toán cực đại cho k tập con $A_0,...,A_{k-1}$ là:

$$W(A_0,...,A_{k-1}) = \max_{0 \leq r < k} nz(A_r).$$

Mục tiêu đề ra là đưa ra thuật toán phân hoạch ma trận A, phân hoạch *P* chiều, thỏa mãn tiêu trí cân bằng tải:

$$W(A_0,...,A_{P-1}) \leq (1+\varepsilon)\frac{W(A)}{P},$$

và có dung lượng truyền thông $V(A_0,...,A_{P-1})$ thấp. Ở đây, $\varepsilon > 0$ là tham số mất cân bằng tải cho phép. Giá trị ε nhỏ có nghĩa rằng cân bằng tải phải gần như hoàn hảo.

Thuật toán đệ qui cho phân hoạch ma trận thưa, gọi là thuật toán Mondrian, cố gắng để phân hoạch nửa ma trận hiện thời theo cách tốt nhất có thể mà không xét đến phân hoạch tiếp theo. Khi $q = \log_2 P$ mỗi mức phân hoạch, cho phép mất cân bằng tải là ε / q. Giá trị ε / q chỉ sử dụng một lần và được điều chỉnh lại cho thích hợp với kết quả sau mỗi mức đệ qui. Chẳng hạn, phần mà lượng công việc nhỏ hơn sẽ cho giá trị mất cân bằng tải lớn hơn phần kia. Giá trị tương ứng của ε là dựa vào số phần tử khác không cực đại *maxnz* cho phép trên mỗi bộ xử lý. Hướng phân hoạch trong thuật toán được lựa chọn là hướng luân phiên.

Kết quả của thuật toán phân hoạch đệ qui ở trên cho ta phân hoạch *P* chiều của ma trận *A*, Bộ xử lý *P(s)* đạt được một tập con $I_s \times J_s$ của ma trận ban đầu, nhưng các hàng và các cột của nó không nhất thiết là liên tiếp.

2.2.3 Phân phối véc tơ

Giả sử sau khi thực hiện phân phối ma trận cho *P* bộ xử lý, sử dụng thuật toán phân phối ma trận thưa của Mondrian. Tiếp theo ta phân phối hai véc tơ *v* và *u* cho *P* bộ xử lý sao cho số cực đại gửi và nhận của các bộ xử lý trong bước (1) và (3) là nhỏ nhất. Bởi vì truyền thông của *u* và *v* trong bước (1) và (3) tương ứng thao tác theo cách tương tự nhau, nên thuật toán để cực tiểu hóa truyền thông cho *v* có thể được sử dụng để cực tiểu hóa truyền thông cho *u*. Do đó ở đây chỉ đề cập tới phân phối của *v*. Ở đây không quan tâm số phần tử khác không trong mỗi cột, mà chỉ quan tâm tới những bộ xử lý mà sở hữu các phần tử khác không này. Và chúng ta đưa ma trận *A* cấp $m \times n$ về ma trận $\tilde{A}$ cấp $P \times n$ với $\tilde{a}_{sc} = 1$ nếu bộ xử lý *s* sở hữu ít nhất một phần tử khác không trong cột *c*, và $\tilde{a}_{sc} = 0$ không sở hữu phần tử khác không nào trong cột *c*. Ma trận $\tilde{A}$ được gọi là ma trận truyền thông.

2.2.3.1 Cận dưới địa phương cho số truyền thông cực đại

Dưới đây là một số cận dưới cho giá trị truyền thông cực đại $\max(\text{Nsend}, \text{Nreceive})$ ở trên. Cận dưới trực quan nhất cho giá trị $\max(\text{Nsend}, \text{Nreceive})$ trong mỗi phân phối véc tơ như sau:

Định lý 2.4 *(Cận dưới 1)*: Cho một phân phối của ma trận *A* trên *P* bộ xử lý, khi đó

$\max(\text{Nsend, Nreceive}) \geq \lceil \text{V/P} \rceil$.

Trong đó $V \equiv \sum_{c=0}^{n-1} \max(k(c)-1, 0) = \sum_{s=0}^{p-1} Ns(s) = \sum_{s=0}^{p-1} Nr(s)$.

Định lý 2.5 *(Cận dưới 2):* Ma trận *A* đã được phân phối trên *P* bộ xử lý, có P_a bộ xử lý kích hoạt, có ít nhất một cột được phân cho nhiều hơn hai bộ xử lý, tức là $|B(s)| > 0$, thì cận dưới tốt hơn cho giá trị $\max(\text{Nsend, Nreceive})$ cho bởi:

$\max(\text{Nsend, Nreceive}) \geq \lceil \text{V/P}_\text{a} \rceil$.

Các cận dưới tiếp theo dưới đây không phải là một cải tiến của cận dưới 1 hay cận dưới 2, nó có thể cao hơn hoặc thấp hơn $\lceil V / P_a \rceil$ hay $\lceil V / P \rceil$ phụ thuộc vào phân phối ma trận. Nếu giả sử rằng tất cả các cột được phân phối đều khắp trên các bộ xử lý. Để cực tiểu $\max(\text{Ns(s), Nr(s)})$ của bộ xử lý *s* mà không xem xét tới các bộ xử lý khác, chúng ta phải đạt được *Ns(s) = Nr(s)*, bởi vì khi *Ns(s)* đạt được bằng việc di chuyển một cột tới một bộ xử lý khác, *Nr(s)* lại tăng lên và ngược lại. Mục tiêu cân bằng *Ns(s)* và *Nr(s)* trong khi chúng là nhỏ nhất. Với *Nr(s)* không quan trọng xem bao nhiêu bộ xử lý cùng trên các cột mà xử lý bởi *s*: mỗi cột sở hữu giảm *Nr(s)* đi 1. Vì vậy để giảm *Nr(s)* tốt nhất là sở hữu nhiều cột có thể, không liên quan tới số bộ xử lý trên nó. Tuy nhiên với *Ns(s)* thì vấn đề lại là bao nhiêu bộ xử lý trên các cột mà *s* sở hữu: và nó càng ít thì càng tốt. Vì vậy, khi chúng ta khởi tạo với tất cả các cột được sở hữu bởi các bộ xử lý khác (chẳng hạn, *Nr(s)=|B(s)|*, $C(s) = \phi$ và $\overline{C}(s) = B(s)$), tốt nhất cho *s* trước tiên sở hữu tất cả các cột trong $\overline{C}(s)$ với hai bộ xử lý, rồi tất cả các cột trong $\overline{C}(s)$ với ba bộ xử lý,…, cho tới khi *Ns(s)=Nr(s)*. Trong hầu hết các trường hợp thường không đạt được dấu bằng mà chỉ tăng *Ns(s)* nhiều nhất có thể nhưng vẫn thoả mãn $Ns(s) \leq Nr(s)$. Giá trị đạt được theo cách này sẽ cho ta giá trị *max(Ns(s),Nr(s))* và được gọi là *local(s)*.

Ta thấy rằng tập các cột *C(s)* đạt được luôn thoả mãn hai tính chất cơ bản sau:

(i) $k(c_1) \leq k(c_2)$, với mọi $c_1 \in C(s)$ và $c_2 \in \overline{C}(s)$,

(ii) $Nr(s) - k_{\overline{C},\min} + 1 \leq Ns(s) \leq Nr(s)$,

với $k_{\overline{C},\min} \equiv \min_{c \in \overline{C}(s)} k(c)$. Nếu chúng ta định nghĩa $local \equiv \max_{0 \leq s < p} local(s)$ thì chúng ta đạt được cận dưới tiếp theo dưới đây.

Định lý 2.6 *(Cận dưới 3)*: Với mỗi phân phối véc tơ thì cận dưới cho giá trị $\max(\text{Nsend, Nreceive})$ như sau:

$\max(Nsend, Nreceive) \geq local$.

Với các ma trận phân phối tốt có thể đạt $local < \lceil V / P_a \rceil$, bởi vì các giá trị địa phương của các bộ xử lý xem xét các cột chi phí rẻ (chẳng hạn các cột có ít bộ xử lý) để sở hữu chúng và tránh các cột chi phí đắt hơn. Vì vậy cận dưới tốt nhất là:

Định lý 2.7 *(Cận dưới 4):* Với mỗi phân phối véc tơ thì cận dưới cho giá trị $\max(\text{Nsend, Nreceive})$ như sau:

$\max(Nsend, Nreceive) \geq \max(\lceil V / P_a \rceil, local)$.

Đây là cận tốt để xác định cận dưới địa phương tổng quát.

Định lý 2.8 *(Cận dưới 5):* Cho tập các cột đã sở hữu *C(s)* với mỗi bộ xử lý *s* cận dưới tổng quát cho giá trị $\max(\text{Nsend, Nreceive})$ sẽ là:

$$\max(Nsend, Nreceive) \geq \max(\lceil V / P_a \rceil, local_{gen}),$$

Trong đó $local_{gen} \equiv \max_{0 \leq s < p} local_{gen}(s)$.

2.2.3.2 Phân phối các véc tơ độc lập

Trong phần này thực hiện phân phối các thành phần véc tơ của *v* và *u* cho các bộ xử lý, ở đây các thành phần v_i và u_i có thể được phân cho các bộ xử lý khác nhau. Bởi vậy các phân phối của *u* và *v* có thể là không đồng thời. Có nghĩa rằng chúng ta có thể phân phối *v* và *u* một cách độc lập.

Thực tế cho thấy, bộ xử lý mà đạt số truyền thông cực đại max(Nsend, Nreceive), bộ xử lý p_{max}, sở hữu nhiều cột hơn. Chẳng hạn số các cột với *k(c) = 2* là nhiều hơn và số các cột với *k(c) = 10* là ít hơn. Sẽ là có lợi nếu đảm bảo rằng các bộ xử lý mà có xu hướng trở thành p_{max} sở hữu các cột rẻ hơn (ít bộ xử lý trên nó). Điều này có thể được thực hiện bằng thuật toán sử dụng giá trị biên địa phương, bởi vì giá trị biên địa phương này xem xét trước tiên các cột rẻ nhất. Trực giác cho ta thấy rằng các bộ xử lý với giá trị biên địa phương cao thường dễ trở thành p_{max} hơn các bộ xử lý có giá trị thấp hơn. Nếu một thuật toán có thể tính giá trị biên địa phương trước, và rồi phân từng cột rẻ nhất cho bộ xử lý với giá trị biên địa phương *local(s)* cao nhất lúc đó, như thế có thể tránh được phân phối xấu.

a) Thuật toán biên địa phương

Thuật toán biên địa phương (LA), được chia làm hai pha. Pha thứ nhất được gọi là phần biên địa phương thực tế và pha thứ hai kết thúc phân phối theo cách tham lam. Để thuật toán làm việc, chúng ta cần giá trị cận dưới địa phương tổng quát thể hiện cận dưới địa phương thời điểm hiện tại. Giá trị này có thể điều chỉnh trong quá trình chạy thuật toán. Cận dưới địa phương tổng quát của bộ xử lý *s* được gọi là *L(s)*.

Trong thuật toán, pha 1:

Bước 1, tất cả các giá trị được khởi tạo.

Bước 2, bộ xử lý được chọn là bộ xử lý mà chưa đạt đến cận dưới địa phương *L(s)* của nó và có *L(s)* cao nhất, bởi vì bộ xử lý này được mong đợi ở thời điểm đó trở thành p_{max} dựa vào thử sai biên địa phương.

Bước 3, cột chưa được sở hữu mà rẻ nhất trong $C_L(p)$ được phân cho bộ xử lý *p*, bởi vậy *C(p)* vẫn thoả mãn (i).

Bước 4, cột c phải được gỡ bỏ khỏi $C_L(q)$, với $q \neq p$, bởi vì cột này đã được sở hữu bởi bộ xử lý khác (gọi là *p*), vì vậy $C_L(q)$ phải được tính lại.

Từ bước 2 tới 4 được lặp lại cho tới khi *Ls(s) = Ns(s)* với mọi bộ xử lý. Một bộ xử lý không đạt thêm bất kỳ cột nào nếu *Ls(s) = Ns(s)*, vì cái này sẽ không thấp hơn *max(Ns(s),Nr(s))* và có thể tăng *max(Ns(q),Nr(q))* với một vài bộ xử lý *q* khác. Bởi vì *q* có thể cần các cột này để đạt được cận dưới địa phương tổng quát.

Trong pha 2, các cột còn lại sẽ được phân:

Bước 1 của pha 2, các cột mà làm tăng số truyền thông sẽ được phân bố tham lam thoả mãn max(Nsend, Nreceive) là nhỏ nhất.

Bước 2 các cột mà không có bộ xử lý trên nó thì được phân cho bộ xử lý mà có số cột được phân là ít nhất.

b) Đánh giá thuật toán

Thời gian: Thuật toán mất $O(nz+P^2)$ thời gian nếu $n < nz$.

Bộ nhớ: Tất cả n mảng *procindex(c)[]* sử dụng *O(nz)* bộ nhớ, *P* mảng *B(s)[]* cũng vậy. Các mảng *needed(s)[]* và *avail(s)[]* mất $O(P^2)$ bộ nhớ. Như vậy bộ nhớ cho thuật này cũng là $O(nz + P^2)$.

2.2.3.3 Phân phối các véc tơ đồng thời

Ở đây xét cho trường hợp ma trận vuông, các thành phần của hai véc tơ u_i và v_i được phân cho cùng bộ xử lý. Ta nói phân phối của hai véc tơ *u* và *v* là đồng thời.

a) Thuật toán biên địa phương

Mục tiêu ở đây là cực tiểu hoá $max(Nsend_r,Nreceive_r)+max(Nsend_c,Nreceive_c)$, ở đây '*r*' và '*c*' có nghĩa chỉ hàng và cột tương ứng. Thay vì phân một cột cho một bộ xử lý thì chúng ta sẽ phân một cặp gồm hàng *i* và cột *i* cho một bộ xử lý và ta gọi là cặp *rowcol*. Thay vì giá trị *L(s)* thì sẽ cần các giá trị $L_r(s)$ và $L_c(s)$ cho mỗi bộ xử lý.

c) Đánh giá thuật toán

Thời gian: Thời gian cho toàn bộ thuật toán là $O(nz \,.\, P)$ thời gian.

Bộ nhớ: Toàn bộ thuật toán mất $O(nz + P^2)$.

Chương 3 – KẾT QUẢ THỰC NGHIỆM

Chương này nhằm đưa ra một số kết quả khi chạy chương trình nhân ma trận thưa với véc tơ song song từ một số bộ dữ liệu được lấy từ các bộ sưu tập ma trận thưa trực tuyến, như Matrix Maket, Rutherford-Boeing, …, từ đó so sánh và đánh giá lại các thuật toán đã trình bày. Chương trình cài đặt chỉ mới thực hiện nhân ma trận thưa với véc tơ song song, nhưng về sau chúng ta có thể sử dụng nó cho việc thiết kế và cài đặt các bài toán giải các hệ phương trình tuyến tính và các hệ phương trình giá trị riêng, và mức cao hơn, giải cho hệ phi tuyến, hệ phương trình vi phân từng phần, và các bài toán khác.

Kết quả

Các bộ dữ liệu, sau khi sử dụng thuật toán Mondrian thực hiện phân phối ma trận, tiếp theo thực hiện phân phối các thành phần véc tơ bằng thuật toán biên địa phương (LA), với *P* từ 2 tới 64, cho phép mất cân bằng tải $\varepsilon = 0.03$.

Thực hiện nhân ma trận thưa với véc tơ song song. Để cho đơn giản thì véc tơ *v* có độ dài *n* được khởi tạo các phần tử toàn 1. Để đánh giá thuật toán LA, ta thống kê cho các cận dưới $\lceil V / P_a \rceil, \lceil V / P \rceil$, cận dưới địa phương *LB* (Lower Bound), giá trị *max(Nsend, Nreceive),* và lượng truyền thông V.

Đánh giá:

Từ các bảng kết quả thuật toán phân phối các thành phần véc tơ dựa theo biên địa phương (LA) cho thấy, cận dưới địa phương *LB* luôn lớn hơn $\lceil V / P \rceil$ và $\lceil V / P_a \rceil$. Trong nhiều trường hợp, cận dưới $\lceil V / P_a \rceil$ là gần bằng với *LB*, tức là nó cũng là khá tốt, nhưng *LB* là cận dưới tốt nhất. Lượng truyền thông cực đại *max(Nsend, Nreceive)* trong thuật toán LA lớn hơn *LB*, và không quá khác xa so với *LB*.

Như vậy, hầu hết các trường hợp là tốt, đặc biệt là cho các bộ dữ liệu càng lớn. Nhưng vẫn có một số trường hợp LA chưa được tốt, đây chính là hạn chế của thuật toán.

KẾT LUẬN

Luận văn đã trình bày được một số nội dung sau:

- Một số kiến thức tổng quan về xử lý song song bao gồm hệ thống song song, các mô hình lập trình song song, nguyên lý thiết kế thuật toán song song và kiến trúc cụm máy tính của trung tâm tính toán hiệu năng cao. Luận văn đã giới thiệu về

giao diện truyền thông điệp MPI nhằm mục đích áp dụng lập trình MPI với ngôn ngữ *C* cho bài toán nhân ma trận thưa với véc tơ song song.

- Thuật toán song song nhân ma trận thưa với véc tơ.
- Cài đặt chương trình nhân ma trận thưa với véc tơ song song, sử dụng thuật toán phân phối ma trận Mondrian, cài đặt thuật toán biên địa phương phân phối véc tơ cho trường hợp tổng quát, ma trận hình chữ nhật.
- Chạy thử nghiệm trên một số bộ dữ liệu được lấy từ bộ sưu tập trực tuyến, http://www.cise.ufl.edu/research/sparse/matrices.
- Từ các kết quả nhận được, luận văn đã chỉ ra rằng thuật toán biên địa phương là khá tốt cho bài toán nhân ma trận thưa với véc tơ với mục tiêu cực tiểu hoá dung lượng truyền thông cực đại và cân bằng tải tính toán cũng như truyền thông.

Trong tương lai luận văn tiếp tục nghiên cứu các thuật toán phân phối ma trận thưa, áp dụng phương pháp phân phối đa mức và phương pháp phân hoạch siêu đồ thị, cải tiến thuật toán phân phối véc tơ bằng phương pháp heuristic.

References

Tiếng Việt

1. Đoàn Văn Ban, Nguyễn Mậu Hân (2006), *Xử lý song song và phân tán*, NXB KHKT.
2. Nguyễn Hữu Điển (2006), *Một số vấn đề về thuật toán*, NXB giáo dục.

Tiếng Anh

3. A. Pinar and C. Aykanat (1997), "Sparse matrix decomposition with optimal load balancing", in Proceedings International Conference on High Performance Computing (HiPC'97), pp. 224-229.
4. B. Hendrickson and R. Leland (1995), A *multilevel algorithm for partitioning graphs*, in Proceedings Supercomputing '95, ACM Press/IEEE Press.
5. B. Hendrickson (1998), "Graph partitioning and parallel solvers: Has the emperor no clothes?*"*, in Proceedings Irregular'98, A. Ferreira, J. Rolim, H. Simon, and S.-H. Teng, eds., vol. 1457 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, pp. 218-225.
6. B. Hendrickson and T. G. Kolda (2000), "Graph partitioning models for parallel computing", Parallel Computing, pp. 1519 – 1534.
7. B. Hendrickson and T. G. Kolda (2000), "Partitioning rectangular and structurally unsymmetric sparse matrices for parallel processing", SIAM Journal on Scientic Computing, pp. 2048-2072.
8. Brendan Vastenhouw and Rob H. Bisseling (2002), *A two-dimensional data distribution method for parallel sparse matrix-vector multiplication*, Preprint Nr. 1238, Dept. of Mathematics, Utrecht University.
9. Cameron Hughes, Tracey Hughes (2003), *Parallel and Distributed Programming Using C++*, Addison Wesley.
10. David Ashton, William Gropp, Ewing Lusk: *Installation and User's Guide to MPICH, a Portable Implementation of MPI Version 1.2.5*, Agonne National Laboratory.
11. G. H. Golub and C. F. Van Loan (1996), *Matrix Computations*, Johns Hopkins Studies in the Mathematical Sciences, The Johns Hopkins University Press, Baltimore, MD, third ed.

12. G. Karypis and V. Kumar (1998), “A fast and high quality multilevel scheme for partitioning irregular graphs”, SIAM Journal on Scientic Computing, pp. 359-392.
13. H. Roosta (2000), *Parallel Processing and Parallel Algorithms*, Springer-Verlag.
14. L. F. Romero and E. L. Zapata (1995), “Data distributions for sparse matrix vector multiplication”, Parallel Computing, pp. 583-605.
15. M. J. Berger and S. H. Bokhari (1987), “A partitioning strategy for nonuniform problems on multiprocessors”, IEEE Transactions on Computers, C-36, pp. 570-580.
16. R. H. Bisseling (1993), “Parallel iterative solution of sparse linear systems on a transputer network”, in Parallel Computation, A. E. Fincham and B. Ford, vol. 46 of the Institute of Mathematics and its Applications Conference Series, Oxford University Press, Oxford, UK, pp. 253-271.
17. R. H. Bisseling and W. F. McColl (1994), “Scientic computing on bulk synchronous parallel architectures”, in Technology and Foundations: Information Processing '94, vol. I, B. Pehrson and I. Simon, eds., vol. 51 of IFIP Transactions A, Elsevier Science Publishers, Amsterdam, pp. 509-514.
18. R. H. Bisseling (2004), *Parallel Scientìic Computation, a structured approach using BSP and MPI*, Oxford University.
19. T. Bui and C. Jones (1993), “A heuristic for reducing fill in sparse matrix factorization”, in Proceedings Sixth SIAM Conference on Parallel Processing for Scientic Computing, SIAM, Philadelphia, PA, pp. 445-452.
20. T. A. Davis (1994-2001), “University of Florida sparse matrix collection”, online collection, http://www.cise.ufl.edu/research/sparse/matrices, Computer and Information Sciences Department, University of Florida, Gainesville, FL.
21. U. V. Catalyurek and C. Aykanat (1999), “Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication”, IEEE Transactions on Parallel and Distributed Systems, 10(7), pp. 673-693.
22. U. V. Catalyurek and C. Aykanat (2001), “A hypergraph-partitioning approach for coarse-grain decomposition”, in Proceedings Supercomputing 2001, ACM.
23. Wouter Meesen (2003), *Distributing vector components of a parallel sparse matrix-vector multiplication*, Utrecht.