1

Tổng tiền tố và Ứng dụng của họ

Guy E. Blelloch

Khoa Khoa học Máy tính
Đại học Carnegie Mellon
Pittsburgh, PA 15213-3890

1.1 Giới thiệu

Các nhà thiết kế thuật toán có kinh nghiệm chủ yếu dựa vào một tập hợp các khối xây dựng và trên các công cụ cần thiết để ghép các khối lại với nhau thành một thuật toán. Các Do đó, hiểu biết về các khối và công cụ cơ bản này là rất quan trọng đối với hiểu các thuật toán. Nhiều khối và công cụ cần thiết để song song thuật toán mở rộng từ các thuật toán tuần tự, chẳng hạn như lập trình động và phân chia để chinh phục, nhưng những thứ khác là mới.

Chương này giới thiệu một trong những cách xây dựng đơn giản và hữu ích nhất khối cho các thuật toán song song: hoạt động tất cả các tiền tố-tổng. Chương de- phạt hoạt động, chỉ ra cách triển khai nó trên PRAM và minh họa nhiều ứng dụng của hoạt động. Ngoài việc là một công trình hữu ích khối, phép toán tất cả tiền tố-tổng là một ví dụ điển hình về phép tính có vẻ như vốn có tuần tự, nhưng có một thuật ngữ song song hiệu quả- nhịp điệu. Hoạt động được định nghĩa như sau:

ĐỊNH NGHĨA

Phép toán tất cả tiền tố-tổng nhận một toán tử kết hợp nhị phân $\oplus$, và một tập hợp có thứ tự gồm n phần tử

$$[a_0, a_1, ..., a_{n-1}],$$

và trả lại bộ đã đặt hàng

$$[a_0, (a_0 \oplus a_1), ..., (a_0 \oplus a_1 \oplus ... \oplus a_{n-1})].$$

Ví dụ, nếu $\oplus$ là phép cộng, thì phép toán tổng tiền tố tất cả trên lệnh có thứ tự bộ

$$[3 \quad 1 \quad 7 \quad 0 \quad 4 \quad 1 \quad 6 \quad 3],$$

sẽ trở lại

$$[3 \quad 4 \quad 11 \quad 11 \quad 14 \quad 16 \quad 22 \quad 25].$$

Việc sử dụng tất cả các phép tính tiền tố là rất rộng rãi. Đây là danh sách vài người trong số họ:

1. So sánh từ vựng các chuỗi ký tự. Ví dụ, để xác định- của tôi rằng "chiến lược" sẽ xuất hiện trước "phân tầng" trong từ điển (xem Vấn đề 2).

2. Để thêm nhiều số chính xác. Đây là những con số không thể được biểu diễn bằng một từ máy duy nhất (xem Vấn đề 3).
3. Để đánh giá đa thức (xem Bài toán 6).
4. Để giải quyết các lần lặp lại. Ví dụ, để giải quyết các lần lặp lại $x_i = a_i x_{i-1} + b_i x_{i-2}$ và $x_i = a_i + b_i / x_{i-1}$ (xem Mục 1.4).
5. Để thực hiện sắp xếp cơ số (xem Phần 1.3).
6. Để thực hiện quicksort (xem Phần 1).
7. Để giải hệ thống tuyến tính tam giác (xem Bài toán 12).
8. Để xóa các phần tử đã đánh dấu khỏi một mảng (xem Phần 1.3).
9. Để cấp phát động bộ xử lý (xem Phần 1.6).
10. Để thực hiện phân tích từ vựng. Ví dụ: để phân tích cú pháp một chương trình thành mã thông báo.
11. Để tìm kiếm các biểu thức chính quy. Ví dụ, để triển khai Chương trình grep UNIX.
12. Để thực hiện một số thao tác trên cây. Ví dụ, để tìm độ sâu của mọi đỉnh trong cây (xem Chương 3).
13. Để gắn nhãn các thành phần trong hình ảnh hai chiều.

Trên thực tế, tất cả các phép toán tổng tiền tố bằng cách sử dụng phép cộng, tối thiểu và tối đa imum rất hữu ích trong thực tế đến nỗi chúng đã được đưa vào dạng nguyên thủy hướng dẫn trong một số máy. Các nhà nghiên cứu cũng đã gợi ý rằng một lớp của phép toán tổng tiền tố tất cả được thêm vào mô hình PRAM dưới dạng “đơn vị thời gian ”nguyên thủy vì việc triển khai phần cứng hiệu quả của chúng.

Trước khi mô tả việc triển khai, chúng ta phải xem xét cách định nghĩa sự ghi nhận của hoạt động tất cả các tiền tố liên quan đến mô hình PRAM. Các định nghĩa nói rằng hoạt động nhận một tập hợp có thứ tự, nhưng không chỉ định cách sắp xếp tập hợp có thứ tự trong bộ nhớ. Một cách để bố trí các yếu tố nằm ở các vị trí liền kề của một vectơ (mảng một chiều). Cách khác là sử dụng danh sách liên kết với các con trỏ từ mỗi phần tử đến phần tử tiếp theo. Nó rõ ra rằng cả hai hình thức hoạt động đều có công dụng. Trong các ví dụ được liệt kê ở trên, ghi nhãn thành phần và một số hoạt động cây yêu cầu danh sách liên kết phiên bản, trong khi các ví dụ khác có thể sử dụng phiên bản vectơ.

Tuần tự, cả hai phiên bản đều dễ dàng tính toán (xem Hình 1.1). Các phiên bản vectơ chuyển xuống vectơ, thêm mỗi phần tử vào một tổng và viết tổng trở lại, trong khi phiên bản danh sách liên kết theo sau các con trỏ trong khi giữ số tiền đang chạy và viết nó trở lại. Các thuật toán trong Hình 1.1 cho cả hai phiên bản vốn có tuần tự: để tính toán một giá trị ở bất kỳ bước nào, kết quả của bước trước đó là cần thiết. Do đó, các thuật toán yêu cầu $O(n)$ thời gian. Để thực hiện song song hoạt động tổng tiền tố, các thuật toán phải

```
proc tất cả tiền tố-tổng (Ra, Vào)
    tôi ← 0
    sum ← Trong [0]
    Hết [0] ← tổng
    while (i <chiều dài)
        i ← i + 1
        sum ← sum + Trong [i]
        Hết [i] ← tổng
```

Phiên bản Vector

```
proc tất cả tiền tố-tổng (Ra, Vào)
    tôi ← 0
    sum ← Trong [0] .value
    Hết [0] ← tổng
    while (Trong [i] .pointer = EOL)
        i ← Trong [i] .pointer
        sum ← sum + Trong [i] .value
        Hết [i] ← tổng
```

Phiên bản danh sách

HÌNH 1.1
Các thuật toán tuần tự để tính toán hoạt động tổng tiền tố tất cả với operator + trên một vectơ và trên một danh sách được liên kết. Trong phiên bản danh sách, mỗi phần tử của Trong bao gồm hai trường: một giá trị (.value) và một con trỏ đến vị trí tiếp theo trong danh sách (.pointer). EOL có nghĩa là con trỏ cuối danh sách.

được thay đổi đáng kể.

Phần còn lại của chương này liên quan đến tiền tố tất cả vectortổng hoạt động. Từ đó, chúng tôi sẽ sử dụng thuật ngữ quét cho thao tác này. 1

ĐỊNH NGHĨA

Thao tác quét là một thao tác tổng hợp tất cả các tiền tố vectơ.

Chương 2, 3 và 4 thảo luận về cách sử dụng phép toán tổng tiền tố trong danh sách liên kết và đưa ra một thuật toán xác định tối ưu cho bài toán trên PRAM.

Đôi khi, nó hữu ích cho mỗi phần tử của vectơ kết quả chứa tổng của tất cả các phần tử trước đó, nhưng không phải của chính phần tử đó. Chúng tôi gọi như vậy một hoạt động, một quét trước.

ĐỊNH NGHĨA

Hoạt động quét trước sử dụng một toán tử kết hợp nhị phân $\oplus$ với danh tính I và một vectơ gồm n phần tử

$$[a_0, a_1, ..., a_{n-1}],$$

1 Thuật ngữ quét bắt nguồn từ ngôn ngữ máy tính APL.

và trả về vectơ

$$[I, a_0, (a_0 \oplus a_1), \ldots, (a_0 \oplus a_1 \oplus \ldots \oplus a_{n-2})].$$

Một bản quét đặt trước có thể được tạo ra từ một lần quét bằng cách dịch chuyển vectơ sang phải một và chèn danh tính. Tương tự, quá trình quét có thể được tạo từ bản quét trước bằng cách dịch chuyển sang trái và chèn vào cuối tổng của phần tử cuối cùng của quét trước và phần tử cuối cùng của vectơ gốc.

1,2 Thực hiện

Phần này mô tả một thuật toán để tính toán hoạt động quét trong song song, tương đông. Đối với bộ xử lý p và vectơ có độ dài n trên EREW PRAM, thuật toán có độ phức tạp thời gian là O (n / p + lg p). Thuật toán đơn giản và rất thích hợp để thực hiện trực tiếp trong phần cứng. Chương 4 cho thấy làm thế nào thời gian của hoạt động quét với một số nhà khai thác nhất định có thể được giảm xuống O (n / p + lg p / lg lg p) trên CREW PRAM.

Trước khi mô tả hoạt động quét, chúng tôi xem xét một vấn đề đơn giản hơn, chỉ tạo ra phần tử cuối cùng của quá trình quét. Chúng tôi gọi đây là giảm hoạt động.

ĐỊNH NGHĨA

Phép toán rút gọn nhận một toán tử kết hợp nhị phân $\oplus$ với danh tính i, và một tập hợp có thứ tự $[a_0, a_1, \ldots, a_{n-1}]$ gồm n phần tử, và trả về giá trị $a_0 \oplus a_1 \oplus \ldots \oplus a_{n-1}$.

Một lần nữa, chúng ta chỉ xem xét trường hợp tập có thứ tự được giữ trong một vectơ. Một cây nhị phân cân bằng có thể được sử dụng để thực hiện thao tác rút gọn bằng cách đặt cây trên các giá trị và sử dụng $\oplus$ để tính tổng các cặp ở mỗi đỉnh (xem Hình 1.2a). Tính đúng đắn của kết quả phụ thuộc vào $\oplus$ là liên kết. Các tuy nhiên, toán tử không cần phải có tính chất giao hoán vì thứ tự của toán hạng được duy trì. Trên EREW PRAM, mỗi cấp độ của cây có thể được thực hiện song song, do đó, việc triển khai có thể chuyển từ phần lá đến rễ cây (xem Hình 1.2b); chúng tôi gọi đây là một cuộc truy quét. Vì cây là độ sâu $\lceil \lg n \rceil$ và cần một bộ xử lý cho mọi cặp phần tử, thuật toán yêu cầu thời gian O (lg n) và n / 2 bộ xử lý.

Nếu chúng ta giả sử một số bộ xử lý p cố định, với n> p, thì mỗi bộ xử lý có thể tính tổng một phần n / p của vectơ để tạo ra tổng bộ xử lý; các

(a) Thực hiện + -reduce trên cây.

```
cho d từ 0 đến (lg n) - 1
   song song với i từ 0 đến n - 1 bởi 2 d + 1
      a [i + 2 d + 1 - 1] ← a [i + 2 d - 1] + a [i + 2 d + 1 - 1]
```

Bươc	Vector trong bộ nhớ							
0	[3	1	7	0	4	1	6	3]
1	[3	4	7	7	4	5	6	9]
2	[3	4	7	11	4	5	6	14]
3	[3	4	7	11	4	5	6	25]

(b) Thực hiện một +-suy giảm trên một PRAM.

HÌNH 1.2
Một ví dụ về hoạt động giảm khi $\oplus$ là phép cộng số nguyên. Những cái hộp trong (b) hiển thị các vị trí được sửa đổi trên mỗi bước. Chiều dài của vectơ là n và phải là một lũy thừa của hai. Kết quả cuối cùng sẽ nằm trong a [n − 1].

Kỹ thuật cây sau đó có thể được sử dụng để giảm tổng của bộ xử lý (xem Hình 1.3). Thời gian cần thiết để tạo tổng bộ xử lý là $\lceil n / p \rceil$, vì vậy tổng thời gian yêu cầu trên EREW PRAM là:

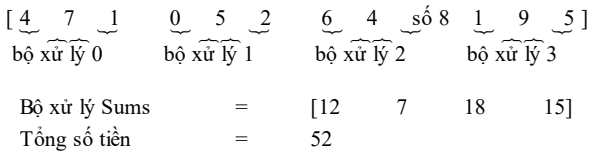
$$T_R(n, p) = \lceil n / p \rceil + \lceil \lg p \rceil = O(n / p + \lg p). \quad (1.1)$$

Khi $n / p \geq \lg p$ thì độ phức tạp là $O(n / p)$. Thời gian này là một tốc độ tối ưu qua thuật toán tuần tự được đưa ra trong Hình 1.1.

Bây giờ chúng ta quay lại thao tác quét. Chúng tôi thực sự chỉ ra cách áp dụng-

Trang 7


```
song song cho mỗi bộ xử lý tôi
   sum [i] ← a [(n / p) i]
   cho j từ 1 đến n / p
      sum [i] ← sum [i] + a [(n / p) i + j]
kết quả ← + -reduce (tổng)
```



HÌNH 1.3
Thao tác + -giảm nhận với nhiều phần tử hơn bộ xử lý. Chúng tôi cho rằng
n / p là một số nguyên.

đề cập đến hoạt động quét trước; quá trình quét sau đó được xác định bằng cách thay đổi kết quả và đặt tổng ở cuối. Nếu chúng ta nhìn vào cái cây được tạo ra bởi sự giảm hoạt động, nó chứa nhiều tổng từng phần trên các vùng của vectơ. Nó rẽ ra rằng những tổng tiền từng phần này có thể được sử dụng để tạo ra tất cả các tổng tiền tố. Điều này yêu cầu thực hiện một lần quét cây khác với một bước mỗi cấp, nhưng điều này thời gian bắt đầu từ gốc và đi đến lá (quét xuống). Ban đầu, phần tử nhận dạng được chèn vào gốc của cây. Trên mỗi bước, mỗi đỉnh ở mức hiện tại chuyển sang con bên trái của nó giá trị riêng của nó và nó chuyển cho con bên phải của nó, ⊕ được áp dụng cho giá trị từ con bên trái từ quá trình quét lên và giá trị riêng của nó (xem Hình 1.4a).

Hãy để chúng tôi xem xét lý do tại sao quét xuống hoạt động. Ta nói rằng đỉnh x trước nhường đỉnh y nếu x xuất hiện trước y trong đường đi ngang của cây được đặt hàng trước (độ sâu đầu tiên, từ trái sang phải).

LÝ THUYẾT 1.1

Sau khi quét xuống hoàn toàn, mỗi đỉnh của cây chứa tổng của tất cả các giá trị lá đứng trước nó.

BẰNG CHỨNG

Bằng chứng là quy nạp từ gốc: chúng tôi chỉ ra rằng nếu cha mẹ có tổng đúng thì cả hai con phải có tổng đúng. Gốc không có các phần tử đứng trước nó, vì vậy giá trị của nó chính xác là phần tử nhận dạng.

(a) Thực hiện quét +-quét trên cây.

```
quy trình quét xuống (A)
   a [n - 1] ← 0                                   % Đặt danh tính
   cho d từ (lg n) - 1 xuống 0
      song song với i từ 0 đến n - 1 bởi 2 d + 1
         t ← a [i + 2 d - 1]                       % Tiết kiệm tạm thời
         a [i + 2 d - 1] ← a [i + 2 d + 1 - 1]     % Đặt con bên trái
         a [i + 2 d + 1 - 1] ← t + a [i + 2 d + 1 - 1]% Đặt đúng con
```

	Bước	Vector trong bộ nhớ							
	0	[3	1	7	0	4	1	6	3]
lên	1	[3	4	7	7	4	5	6	9]
	2	[3	4	7	11	4	5	6	14]
	3	[3	4	7	11	4	5	6	25]
thông thoáng	4	[3	4	7	11	4	5	6	0]
xuống	5	[3	4	7	0	4	5	6	11]
	6	[3	0	7	4	4	11	6	16]
	7	[0	3	4	11	11	15	16	22]

(b) Thực hiện quét quét + trên PRAM.

HÌNH 1.4
Quét trước song song trên cây bằng cách sử dụng phép cộng số nguyên làm toán tử kết hợp $\oplus$, và 0 là danh tính.

HÌNH 1.5
Minh họa cho Định lý 1.1.

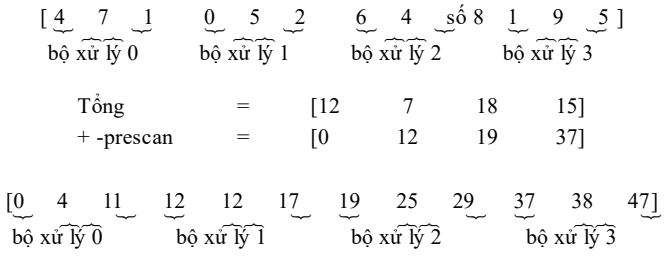
Xem xét Hình 1.5. Con bên trái của bất kỳ đỉnh nào có cùng một lá trước nó như là đỉnh của chính nó (các lá ở vùng A trong nhân vật). Điều này là do truyền tải đặt hàng trước luôn truy cập phần con bên trái của một đỉnh ngay sau đỉnh. Theo giả thuyết quy nạp, cha mẹ có tổng chính xác, vì vậy nó chỉ cần sao chép tổng này sang bên trái đứa trẻ.

Con bên phải của đỉnh bất kỳ có hai bộ lá đứng trước nó, các lá đứng trước vùng cha mẹ (vùng A) và để lại ở hoặc phía dưới vùng con bên trái (vùng B). Do đó, bằng cách thêm giá trị quét xuống của cha mẹ, giá trị này là đúng bằng giả thuyết quy nạp và giá trị quét lên của trẻ bên trái, con phải sẽ chứa tổng của tất cả các lá đứng trước nó.

Vì các giá trị của lá đứng trước bất kỳ lá nào là các giá trị ở bên trái của nó theo thứ tự quét, các giá trị ở các lá là kết quả của từ trái sang phải quét trước. Để thực hiện quét trước trên EREW PRAM, tổng một phần ở mỗi đỉnh phải được giữ trong quá trình quét lên để chúng có thể được sử dụng trong quét xuống. Do đó, chúng ta phải cẩn thận để không ghi đè chúng. Trong thực tế, đây là động lực để đưa các khoản tiền vào bên phải trong quá trình giảm trong Hình 1.2b. Hình 1.4b cho thấy mã PRAM cho quá trình quét xuống. Mỗi bước có thể thực hiện song song, do đó thời gian chạy là 2 ⌈lg n⌉.

Nếu giả sử một số bộ xử lý p cố định, với n> p, chúng ta có thể sử dụng một phương pháp tương tự như trong hoạt động giảm thiểu để tạo ra một

Trang 10



HÌNH 1.6
A + -prescan với nhiều phần tử hơn bộ xử lý.

thuật toán. Mỗi bộ xử lý đầu tiên tính tổng một phần n / p của vectơ để tạo

tổng bộ xử lý, kỹ thuật cây sau đó được sử dụng để quét trước bộ xử lý số tiền. Kết quả quét trước các tổng của bộ xử lý được sử dụng như một phần bù để mỗi bộ xử lý quét trước trong phần n / p của nó (xem Hình 1.6). Thời gian độ phức tạp của thuật toán là:

$$T S (n, p) = 2 (\lceil n / p \rceil + \lceil \lg p \rceil) = O (n / p + \lg n) \quad (1.2)$$

là thứ tự tương tự như hoạt động giảm và cũng là một tốc độ tối ưu qua phiên bản tuần tự khi $n / p \geq \lg p$.

Phần này mô tả cách triển khai thao tác quét (quét trước). Phần còn lại của chương thảo luận về các ứng dụng của nó.

1,3 Line-of-Sight và Radix-Sort

Ví dụ về việc sử dụng thao tác quét, hãy xem xét một dòng đơn giản vấn đề về thị giác. Vấn đề về tầm nhìn là: đưa ra một bản đồ địa hình ở dạng lưới độ cao và điểm quan sát X trên lưới, tìm điểm nào có thể nhìn thấy dọc theo một tia xuất phát tại điểm quan sát (xem Hình 1.7).

Một điểm trên tia có thể nhìn thấy được nếu và chỉ khi không có điểm nào khác giữa nó và điểm quan sát có góc thẳng đứng lớn hơn. Để tìm xem có điểm nào trước đó có một góc lớn hơn, độ cao của mỗi điểm dọc theo tia được đặt trong một vectơ (vectơ độ cao). Các độ cao này sau đó được chuyển đổi thành các góc và đặt

Trang 11


```
quy trình đường ngắm (độ cao)
   song song cho mỗi chỉ số i
      angle [i] ← arctan (tỷ lệ × (độ cao [i] - độ cao [0]) / i)
   max-trước-góc ← max-prescan (góc)
   song song cho mỗi chỉ số i
      if (angle [i]> max-before-angle [i])
          kết quả [i] ← "hiển thị"
      khác
          kết quả [i] ← không "hiển thị"
```

HÌNH 1.7
Thuật toán đường ngắm cho một tia duy nhất. Dấu X đánh dấu quan sát
điểm. Các điểm nhìn thấy được tô bóng. Một điểm trên tia có thể nhìn thấy nếu không
điểm trước đó có góc lớn hơn.

trong vectơ góc (xem Hình 1.7). Quét trước sử dụng toán tử tối đa
(max-prescan) sau đó được thực thi trên vectơ góc, vectơ này trở về mỗi điểm
góc trước lớn nhất. Để kiểm tra khả năng hiển thị, mỗi điểm chỉ cần
so sánh góc của nó với kết quả của phép quét đặt trước tối đa. Điều này có thể được khái quát thành
tìm tất cả các điểm có thể nhìn thấy trên lưới. Đối với n điểm trên một tia, độ phức tạp của
thuật toán là độ phức tạp của quá trình quét, T S (n, p) = O (n / p + lg n) trên một
EREW PRAM.

Bây giờ chúng ta xem xét một ví dụ khác, một thuật toán sắp xếp cơ số. Thuật toán
lặp lại các bit của khóa, bắt đầu từ bit thấp nhất, thực hiện phân tách

Trang 12


```
thủ tục split-radix-sort (A, number-of-bits)
  cho tôi từ 0 đến (số bit - 1)
    A ← tách (A, A ⟨i⟩ )
```

A	=	[5	7	3	1	4	2	7	2]
A ⟨0⟩	=	[1	1	1	1	0	0	1	0]
A ← tách (A, A ⟨0⟩)	=	[4	2	2	5	7	3	1	7]
A ⟨1⟩	=	[0	1	1	0	1	1	0	1]
A ← tách (A, A ⟨1⟩)	=	[4	5	1	2	2	7	3	7]
A ⟨2⟩	=	[1	1	0	0	0	1	0	1]
A ← tách (A, A ⟨2⟩)	=	[1	2	2	3	4	5	7	7]

HÌNH 1.8
Một ví dụ về sắp xếp cơ số tách trên một vectơ có chứa ba giá trị bit. Các
Kí hiệu ⟨n⟩ biểu thị việc trích xuất bit thứ n của mỗi phần tử của vectơ A.
Phép toán tách đóng gói các phần tử có cờ 0 ở dưới cùng và với 1
cờ lên đầu.

hoạt động trên mỗi lần lặp (giả sử tất cả các khóa có cùng số bit).
Phép toán tách đóng gói các khóa bằng 0 trong bit tương ứng với

dưới cùng của một vectơ và đóng gói các khóa bằng 1 bit ở đầu
cùng một vectơ. Nó duy trì trật tự trong cả hai nhóm. Sự sắp xếp hoạt động
bởi vì mỗi hoạt động phân tách sẽ sắp xếp các khóa liên quan đến bit hiện tại
(0 xuống, 1 lên) và duy trì thứ tự đã sắp xếp của tất cả các bit thấp hơn vì chúng tôi
lặp từ bit dưới lên. Hình 1.8 cho thấy một ví dụ về cách sắp xếp.

Bây giờ chúng tôi xem xét cách hoạt động phân tách có thể được thực hiện bằng cách sử dụng
quét. Ý tưởng cơ bản là xác định một chỉ mục mới cho mỗi phần tử và sau đó
hoán vị các phần tử thành các chỉ số mới này bằng cách sử dụng một bản ghi độc quyền. Để chết-
xác định các chỉ số mới cho các phần tử có bit 0, chúng tôi đảo ngược các cờ
và thực hiện quét trước với phép cộng số nguyên. Để xác định các chỉ số mới của
các phần tử có bit 1, chúng tôi thực thi dấu + -scan theo thứ tự ngược lại (bắt đầu từ
đỉnh của vectơ) và trừ các kết quả từ độ dài của vectơ
n. Hình 1.9 cho thấy một ví dụ về hoạt động phân tách cùng với mã để
thực hiện nó.

Vì hoạt động phân tách chỉ yêu cầu hai hoạt động quét, một vài bước
truy cập bộ nhớ độc quyền và một số phép toán số học song song, nó có
cùng độ phức tạp tiệm cận như quét: O (n / p + lg p) trên EREW

Trang 13


```
phân chia thủ tục (A, Cờ)
    I-down ← + -prescan (not (Flags))
    I-up       ← n - + -scan (thứ tự ngược lại (Cờ))
    song song cho mỗi chỉ số i
        nếu (Cờ [i])
            Chỉ mục [i] ← I-up [i]
        khác
            Chỉ mục [i] ← I-down [i]
    kết quả ← hoán vị (A, Chỉ mục)
```

A	=	[5	7	3	1	4	2	7	2]
Cờ	=	[1	1	1	1	0	0	1	0]
Tôi xuống	=	[0	0	0	0	0	1	2	2]
I-up	=	[3	4	5	6	6	6	7	7]
Mục lục	=	[3	4	5	6	0	1	7	2]
hoán vị (A, Chỉ mục)	=	[4	2	2	5	7	3	1	7]

HÌNH 1.9
Phép toán tách gói các phần tử có số 0 trong cờ tương ứng
vị trí ở dưới cùng của vectơ và đóng gói các phần tử bằng 1 vào
đỉnh của cùng một vectơ. Hoán vị ghi mỗi phần tử của A vào chỉ mục
được chỉ định bởi vị trí tương ứng trong Chỉ mục.

PRAM. 2 Nếu chúng ta giả sử rằng n khóa dài mỗi O (lg n) bit, thì tổng thể

thuật toán chạy trong thời gian:

$$O\left(\frac{n}{p} + \lg p\right) \lg n) = O\left(\frac{n}{p} \lg n + \lg n \lg p\right).$$

1,4 Phương trình lặp lại

Phần này cho thấy cách các phương trình lặp lại khác nhau có thể được giải quyết bằng cách sử dụng hoạt động quét. Sự lặp lại là một tập hợp các phương trình có dạng

$$x \text{ tôi} = f\, i\,(x\, i - 1\,,\, x\, i - 2\,,\, \cdots,\, x\, i - m\,),\quad m \le i < n \qquad (1.3)$$

2 Trên CREW PRAM, chúng ta có thể sử dụng cách quét được mô tả trong Chương 4 để có được thời gian là O (n / p + lg p / lg lg p).

Trang 14

cùng với một bộ giá trị ban đầu $x\ 0\ ,\ \cdots,\ x\ m - 1\ .$

Thao tác quét là trường hợp đặc biệt của biểu mẫu lặp lại

$$x \text{ tôi} = \begin{cases} \text{một } 0 & i = 0 \\ x\, i - 1 \oplus a\, i & 0 < i < n, \end{cases} \qquad (1.4)$$

trong đó $\oplus$ là bất kỳ toán tử kết hợp nhị phân nào. Phần này chỉ ra cách giảm một lớp định kỳ tổng quát hơn cho phương trình (1.4), và do đó làm thế nào để sử dụng thuật toán quét được thảo luận trong Phần 1.2 để giải quyết các lần lặp lại này trong song song, tương đông.

1.4.1 Số lần lặp lại của đơn hàng đầu tiên

Ban đầu, chúng tôi xem xét các lần lặp lại bậc nhất của biểu mẫu sau

$$x \text{ tôi} = \begin{cases} b\ 0 & i = 0 \\ (x\, i - 1 \otimes a\, i\,) \oplus b\, i & 0 < i < n, \end{cases} \qquad (1.5)$$

trong đó $a\ i$ và $b\ i$'s là tập hợp n hằng số tùy ý (không nhất thiết là vô hướng) và $\oplus$ và $\otimes$ là các toán tử nhị phân tùy ý đáp ứng ba hạn chế:

1. $\oplus$ là liên kết (tức là $(a \oplus b) \oplus c = a \oplus (b \oplus c)$).
2. $\otimes$ là bán tương đối (nghĩa là tồn tại một toán tử kết hợp nhị phân $\odot$ sao cho $(a \otimes b) \otimes c = a \otimes (b \odot c)$).
3. $\otimes$ phân phối trên $\oplus$ (tức là $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$).

Toán tử $\odot$ được gọi là toán tử đồng hành của $\otimes$. Nếu $\otimes$ là hoàn toàn liên kết, thì $\odot$ và $\otimes$ là tương đương.

Bây giờ chúng tôi chỉ ra cách (1.5) có thể được giảm xuống (1.4). Hãy xem xét tập hợp của cặp

$$c\ i = [a\ i\ ,\ b\ i\ ] \qquad (1.6)$$

và xác định một toán tử nhị phân mới • như sau:

$$c_i \bullet c_j \equiv [c_{i,a} \odot c_{j,a}, (c_{i,b} \otimes c_{j,a}) \oplus c_{j,b}] \quad (1.7)$$

trong đó $c_{i,a}$ và $c_{i,b}$ lần lượt là phần tử thứ nhất và thứ hai của c_i.

Với các điều kiện trên các toán tử $\oplus$ và $\otimes$, toán tử $\bullet$ là associative như chúng tôi hiển thị bên dưới:

$$\begin{aligned}
&(c_i \bullet c_j) \bullet c_k \\
&\quad = [c_{i,a} \odot c_{j,a}, (c_{i,b} \otimes c_{j,a}) \oplus c_{j,b}] \bullet c_k \\
&\quad = [(c_{i,a} \odot c_{j,a}) \odot c_{k,a}, (((c_{i,b} \otimes c_{j,a}) \oplus c_{j,b}) \otimes c_{k,a}) \oplus c_{k,b}]
\end{aligned}$$

Trang 15

$$\begin{aligned}
&= [c_{i,a} \odot (c_{j,a} \odot c_{k,a}), ((c_{i,b} \otimes c_{j,a}) \otimes c_{k,a}) \oplus ((c_{j,b} \otimes c_{k,a}) \oplus c_{k,b})] \\
&= [c_{i,a} \odot (c_{j,a} \odot c_{k,a}), (c_{i,b} \otimes (c_{j,a} \odot c_{k,a})) \oplus ((c_{j,b} \otimes c_{k,a}) \oplus c_{k,b})] \\
&= c_i \bullet [c_{j,a} \odot c_{k,a}, (c_{j,b} \otimes c_{k,a}) \oplus c_{k,b}] \\
&= c_i \bullet (c_j \bullet c_k)
\end{aligned}$$

Bây giờ chúng ta xác định tập có thứ tự $s_i = [y_i, x_i]$, trong đó y_i tuân theo định kỳ-rence

$$y_i = \begin{cases} a_0 & i = 0 \\ y_{i-1} \odot a_i & 0 < i < n, \end{cases} \quad (1.8)$$

và x_i là từ (1.5). Sử dụng (1.5), (1.6) và (1.8), chúng tôi nhận được:

$$\begin{aligned}
s_0 &= [y_0, x_0] \\
&= [a_0, b_0] \\
&= c_0
\end{aligned}$$

$$\begin{aligned}
s_i &= [y_i, x_i] \qquad 0 < i < n \\
&= [y_{i-1} \odot a_i, (x_{i-1} \otimes a_i) \oplus b_i] \\
&= [y_{i-1} \odot c_{i,a}, (x_{i-1} \otimes c_{i,a}) \oplus c_{i,b}] \\
&= [y_{i-1}, x_{i-1}] \bullet c_i \\
&= s_{i-1} \bullet c_i .
\end{aligned}$$

Vì $\bullet$ là liên kết, chúng tôi đã giảm (1,5) xuống (1,4). Kết quả x_i chỉ là các giá trị thứ hai của s_i (các $s_{i,b}$). Điều này cho phép chúng tôi sử dụng thuật toán quét của Phần 1.2 với toán tử $\bullet$ để giải quyết bất kỳ sự lặp lại nào của biểu mẫu (1.5) trên một EREW PRAM trong thời gian:

$$T_S(n,p) = 2(T_\odot + T_\otimes + T_\oplus)(n/p + \lg(p)) \quad (1.9)$$

trong đó $T_\odot$, $T_\otimes$ và $T_\oplus$ là thời gian được thực hiện bởi $\odot$, $\otimes$ và $\oplus$ ($\bullet$ thực hiện một cuộc gọi đến mỗi). Nếu tất cả những gì cần thiết là giá trị cuối cùng x_{n-1}, thì chúng ta có thể sử dụng giảm thay vì quét bằng toán tử $\bullet$, và thời gian chạy là:

$$(T_{\odot} + T_{\otimes} + T_{\oplus})\, T_R(n, p) = (T_{\odot} + T_{\otimes} + T_{\oplus})\,(n / p + \lg p) \quad (1.10)$$

tiệm cận là hệ số nhanh hơn 2 so với (1,9).

Các ứng dụng của định kỳ tuyến tính bậc nhất bao gồm mô phỏng các hệ thống tuyến tính thay đổi theo thời gian khác nhau, giai đoạn phục hồi ngược của tam giác bộ giải hệ thống tuyến tính và đánh giá đa thức.

Trang 16

1.4.2 Số lần lặp lại đơn hàng cao hơn

Bây giờ chúng ta xem xét các lần lặp lại m đơn hàng tổng quát hơn của biểu mẫu:

$$x_i = \begin{cases} b_i & 0 \le i < m \\ (x_{i-1} \otimes a_{i,1}) \oplus \cdots \oplus (x_{i-m} \otimes a_{i,m}) \oplus b_i & m \le i < n \end{cases} \quad (1.11)$$

trong đó $\oplus$ và $\otimes$ là toán tử nhị phân có cùng ba hạn chế như trong (1.5): $\oplus$ là liên kết, $\otimes$ là bán liên kết, và $\otimes$ phân phối trên $\oplus$.

Để chuyển phương trình này thành dạng (1.5), chúng ta xác định như sau vectơ của các biến:

$$s_i = [x_i \quad \cdots x_{i-m+1}]. \quad (1.12)$$

Sử dụng (1.11), chúng ta có thể viết (1.12) là:

$$s_i = [x_{i-1} \quad \cdots x_{i-m}] \otimes (v) \begin{bmatrix} a_{i,1} & 1 & 0 & \cdots & 0 \\ \cdots & & & & \cdots \\ & & 0 & 1 & \\ \cdots & & \cdots & & \cdots\ 0 \\ & & & & \cdots \\ a_{i,m} & 0 & \cdots & 0 & 0 \end{bmatrix} \oplus (v)\, [b_i \quad 0 \quad \cdots \quad 0]$$

$$= (s_{i-1} \otimes (v)\, A_i) \oplus (v)\, B_i \quad (1.13)$$

trong đó $\otimes$ (v) là phép nhân vectơ-ma trận và $\oplus$ (v) là phép cộng vectơ. Nếu chúng ta sử dụng nhân ma trận-ma trận với tư cách là toán tử đồng hành của $\otimes$ (v) , thì (1.13) ở trong biểu mẫu (1.5). Thời gian thực hiện để giải phương trình dạng (1.11) trên một EREW PRAM do đó:

$$(T_{m\otimes m}(m) + T_{v\otimes m}(m) + T_{v\oplus v}(m))\, T_S(n, p) = O((n / p + \lg p)\, T_{m\otimes m}(m)) \quad (1.14)$$

trong đó $T_{m\otimes m}(m)$ là thời gian nhân ma trận $m\otimes m$. Trình tự-độ phức tạp tial để giải phương trình là O (nm), vì vậy độ phức tạp song song là tối ưu theo n khi $n / p \ge \lg p$, nhưng không tối ưu theo m — thuật toán song song-rithm thực hiện một hệ số $O(T_{M\otimes M}(m) / m)$ nhiều hơn so với tuần tự thuật toán.

Các ứng dụng của việc lặp lại (1.11) bao gồm việc giải quyết các lần lặp lại của

dạng x i = a i + b i / x i − 1 (xem bài toán 10) và tạo n Fibonacci đầu tiên
các số x 0 = x 1 = 1, x i = x i − 1 + x i − 2 (xem bài 11).

Trang 17

a	=	[5	1	3	4	3	9	2	6]
f	=	[1	0	1	0	0	0	1	0]
phân đoạn + -scan	=	[5	6	3	7	10	19	2	số 8]
quét tối đa được phân đoạn	=	[5	5	3	4	4	9	2	6]

HÌNH 1.10
Các hoạt động quét được phân đoạn khởi động lại ở đầu mỗi phân đoạn. Các vectơ f chứa các cờ đánh dấu điểm đầu của các đoạn.

1,5 Quét được phân đoạn

Phần này cho biết cách vectơ được vận hành bởi quá trình quét có thể được chia thành phân đoạn có cờ để quá trình quét bắt đầu lại ở mỗi ranh giới phân đoạn (xem Hình 1.10). Mỗi lần quét này nhận hai vectơ giá trị: một vectơ dữ liệu và một vector cờ. Các hoạt động quét được phân đoạn trình bày một cách thuận tiện để thực hiện quét một cách độc lập trên nhiều bộ giá trị. Phần tiếp theo cho thấy cách quét được phân đoạn có thể được sử dụng để thực hiện một nhanh song song, bằng cách giữ mỗi cuộc gọi đệ quy trong một phân đoạn riêng biệt và sử dụng một + -scan để thực hiện phân tách trong mỗi phân đoạn.

Các bản quét được phân đoạn đáp ứng sự lặp lại:

$$x_i = \begin{cases} a_0 & i = 0 \\ a_i & f_i = 1 \\ (x_{i-1} \oplus a_i) & f_i = 0 \end{cases} \qquad 0 < i < n \tag{1.15}$$

trong đó $\oplus$ là toán tử quét kết hợp ban đầu. Nếu $\oplus$ có một danh tính $i_\oplus$, thì (1.15) có thể được viết là:

$$x_i = \begin{cases} a_0 & i = 0 \\ (x_{i-1} \times_s f_i) \oplus a_i & 0 < i < n \end{cases} \tag{1.16}$$

trong đó $\times_s$ được định nghĩa là:

$$x \times_s f = \begin{cases} i_\oplus & f = 1 \\ x & f = 0. \end{cases} \tag{1.17}$$

Đây là dạng (1,5) và $\times_s$ là bán liên kết với lôgic hoặc dưới dạng toán tử đồng hành (xem Vấn đề 9). Vì chúng tôi đã giảm (1,15) xuống

biểu mẫu (1.5), chúng ta có thể sử dụng kỹ thuật được mô tả trong Phần 1 để thực thi quét phân đoạn trong thời gian

$$(T_{\text{hoặc}} + T_{\times} s + T_{\oplus}) T_S(n, p). \tag{1.18}$$

Độ phức tạp thời gian này chỉ là một yếu tố không đổi nhỏ lớn hơn phiên bản không phân đoạn vì hoặc và × s là các toán tử tầm thường.

1.5.1 Ví dụ: Quicksort

Để minh họa việc sử dụng quét phân đoạn, chúng tôi xem xét một phiên bản song song của quicksort. Tương tự như phiên bản tuần tự tiêu chuẩn, phiên bản song song chọn một trong các khóa làm giá trị tổng hợp, chia các khóa thành ba bộ — khóa nhỏ hơn, bằng và lớn hơn pivot — và đệ quy trên mỗi tập hợp. 3 Các thuật toán song song có độ phức tạp thời gian dự kiến là $O(T_S(n, p) \lg n) = O(\frac{n}{p} \lg n + \lg^2 n)$.

Trực giác cơ bản của phiên bản song song là giữ cho mỗi tập hợp con trong phân đoạn riêng và để chọn các giá trị tổng hợp và tách các khóa một cách độc lập trong từng đoạn. Hình 1.11 cho thấy mã giả cho quicksort song song và đưa ra một ví dụ. Các bước sắp xếp được trình bày như sau:

1. Kiểm tra xem các phím đã được sắp xếp hay chưa và thoát khỏi quy trình nếu có.
   Mỗi bộ xử lý sẽ kiểm tra xem liệu bộ xử lý trước có thấp hơn hoặc giá trị tương đương. Chúng tôi thực hiện giảm với lôgic và để kiểm tra xem tất cả các phần tử theo thứ tự.
2. Trong mỗi phân đoạn, chọn một trục và phân phối nó cho phần kia các yếu tố.
   Nếu chúng tôi chọn phần tử đầu tiên làm trục, chúng tôi có thể sử dụng quét phân đoạn với bản sao toán tử nhị phân, trả về giá trị đầu tiên trong số hai tranh luận:

   $$a \leftarrow \text{bản sao}(a, b).$$

   Điều này có tác dụng sao chép phần tử đầu tiên của mỗi đoạn trên toàn phân khúc. Thuật toán cũng có thể chọn một số ngẫu nhiên trong mỗi phân đoạn (xem Vấn đề 15).
3. Trong mỗi phân đoạn, hãy so sánh từng phần tử với trục xoay và phân chia dựa trên kết quả của phép so sánh.
   Đối với việc phân tách, chúng tôi có thể sử dụng một phiên bản của hoạt động phân tách được mô tả trong Phần 1.3 chia thành ba bộ thay vì hai, và

3 Chúng ta không cần phải sắp xếp đệ quy các khóa bằng trục xoay, nhưng thuật toán như được mô tả dưới đây không.

```
thủ tục quicksort (các phím)
    seg-flags [0] ← 1
    trong khi không được sắp xếp (khóa)
        trụ          ← seg-copy (phím, seg-flags)
        f            Phím ← pivots <=>
        chìa khóa    ← seg-split (phím, f, seg-flags)
        seg-flags ← new-seg-flags (key, pivots, seg-flags)
```

```
Chìa khóa                = [6,4 9,2 3,4 1,6 8,7 4,1 9,2 3,4]
Seg-Flags                = [1   0   0   0   0   0   0   0]

Pivots                   = [6,4 6,4 6,4 6,4 6,4 6,4 6,4 6,4]
F                        = [=   >   <   <   >   <   >   <]
Phím ← tách (Phím, F) =    [3,4 1,6 4,1 3,4 6,4 9,2 8,7 9,2]
Seg-Flags                = [1   0   0   0   1   1   0   0]

Pivots                   = [3,4 3,4 3,4 3,4 6,4 9,2 9,2 9,2]
F                        = [=   <   >   =   =   =   <   =]
Phím ← tách (Phím, F) =    [1,6 3,4 3,4 4,1 6,4 8,7 9,2 9,2]
Seg-Flags                = [1   1   0   1   1   1   1   0]
```

HÌNH 1.11
Một ví dụ về quicksort song song. Trên mỗi bước, trong mỗi phân đoạn, chúng tôi cống nạp vào trục, kiểm tra xem mỗi phần tử là bằng, nhỏ hơn hay lớn hơn- so với trục xoay, chia thành ba nhóm và tạo một tập hợp phân đoạn mới cờ. Phép toán <=> trả về một trong ba giá trị tùy thuộc vào việc đối số đầu tiên nhỏ hơn, bằng hoặc lớn hơn đối số thứ hai.

được phân đoạn. Để thực hiện một sự phân chia có phân đoạn như vậy, chúng ta có thể sử dụng một seg- phiên bản được đề cập của thao tác + -scan để tạo các chỉ số tương đối ở đầu mỗi phân đoạn và chúng ta có thể sử dụng một bản sao được phân đoạn- quét để sao chép phần bù của phần đầu của mỗi đoạn qua bộ phận. Sau đó, chúng tôi thêm phần bù vào các chỉ số phân khúc để tạo vị trí mà chúng tôi hoán vị từng phần tử.

4. Trong mỗi phân đoạn, hãy chèn các cờ phân đoạn bổ sung để phân tách các giá trị phân chia. Biết giá trị tổng hợp, mỗi phần tử có thể xác định xem nó có ở đầu phân đoạn bằng cách xem phần tử trước đó.
5. Quay lại bước 1.

Mỗi lần lặp lại kiểu này yêu cầu một số lần gọi quét liên tục
và đối với các cơ sở ban đầu của PRAM. Nếu chúng tôi chọn các trục một cách ngẫu nhiên trong mỗi
phân đoạn, nhanh chóng được dự kiến sẽ hoàn thành trong O (lg n) lần lặp, và do đó
có thời gian chạy dự kiến là O (lg n · T S (n, p)).

Kỹ thuật chia đoạn đệ quy thành các đoạn nhỏ và
hoạt động độc lập trong mỗi phân đoạn có thể được sử dụng cho nhiều
các thuật toán chia để trị, chẳng hạn như hợp nhất.

1,6 Bộ xử lý phân bổ

Hãy xem xét vấn đề sau: đưa ra một tập hợp các bộ xử lý, mỗi bộ chứa-
nhập một số nguyên, phân bổ số nguyên đó của bộ xử lý mới cho mỗi bộ xử lý ban đầu
bộ xử lý. Việc phân bổ như vậy là cần thiết trong quy trình vẽ đường thẳng song song
được mô tả trong Phần 1. Trong quy trình vẽ đường thẳng này, mỗi bộ xử lý calcu-
tính toán số lượng pixel trong dòng và phân bổ động một bộ xử lý
cho mỗi pixel. Phân bổ các phần tử mới cũng hữu ích cho phần phân nhánh
của nhiều thuật toán rẽ nhánh và ràng buộc. Ví dụ, hãy xem xét một lực lượng vũ phu
thuật toán chơi cờ vua thực hiện tìm kiếm độ sâu cố định các nước đi có thể có
để xác định nước đi tiếp theo tốt nhất. Chúng ta có thể kiểm tra hoặc tìm kiếm các bước di chuyển song song
bằng cách đặt từng động thái có thể vào một bộ xử lý riêng biệt. Kể từ khi thuật toán
tự động quyết định có bao nhiêu bước di chuyển tiếp theo để tạo ra (tùy thuộc vào
vị trí), chúng ta cần cấp phát động các phần tử xử lý mới.

Chính thức hơn, cho trước một vectơ A có độ dài l với các phần tử nguyên a i , allo-
cation là nhiệm vụ tạo ra một vectơ B mới có độ dài

$$L = \sum_{i=0}^{l-1} \text{một tôi} \qquad (1.19)$$

với i phần tử của B được gán cho mỗi vị trí i của A. Bằng cách gán cho, chúng tôi
nghĩa là phải có một số phương pháp để phân phối giá trị ở vị trí i của
một vectơ cho các phần tử a i được gán cho vị trí đó. Vì có
sự tương ứng 1-1 giữa các phần tử của vectơ và bộ xử lý,
vectơ gốc yêu cầu l bộ xử lý và vectơ mới yêu cầu L bộ xử lý.
Thông thường, một thuật toán không hoạt động trên hai vectơ cùng một lúc,
để chúng ta có thể sử dụng cùng một bộ xử lý cho cả hai.

Việc phân bổ có thể được thực hiện bằng cách chỉ định một phân đoạn liền kề của
các phần tử cho mỗi vị trí i của A. Để phân bổ các phân đoạn, chúng tôi thực hiện quét + -prescan

V = [v 1 v 2 câu 3]
A = [4 1 3]
Hpointers ← + -prescan (A) = [0 4 5]

% c c

Cờ phân đoạn = [1 0 0 0 1 1 0 0]
phân phối (V, Hpointers) = [v 1 v 1 v 1 v 1 v 2 v 3 v 3 câu 3]
chỉ mục (Hpointers) = [0 1 2 3 0 0 1 2]

HÌNH 1.12
Một ví dụ về phân bổ bộ xử lý. Vectơ A chỉ định có bao nhiêu
các yếu tố mà mỗi vị trí cần. Chúng tôi có thể phân bổ một phân đoạn cho mỗi vị trí bằng cách
áp dụng + -prescan cho A và sử dụng kết quả làm con trỏ đến đầu
của mỗi phân khúc. Sau đó, chúng ta có thể phân phối các giá trị của V cho các phần tử mới
với hoán vị đầu đoạn và quét sao chép phân đoạn
trên toàn phân khúc.

trên vectơ A trả về một con trỏ đến đầu mỗi đoạn (xem Hình-
urê 1,12). Sau đó, chúng tôi có thể tạo các cờ phân đoạn thích hợp bằng cách viết
gắn cờ cho chỉ mục được chỉ định bởi con trỏ. Để phân phối các giá trị từ mỗi vị trí-
tương ứng với phân đoạn của nó, chúng tôi ghi các giá trị vào đầu các phân đoạn và
sử dụng thao tác sao chép-quét được phân đoạn để sao chép các giá trị trên phân đoạn.
Phân bổ và phân phối mỗi yêu cầu một lần gọi để quét và do đó có
độ phức tạp T S (l, p) và T S (L, p) tương ứng.

Khi một phân đoạn đã được phân bổ cho mỗi phần tử ban đầu, nó thường
cần thiết để tạo các chỉ số trong mỗi phân đoạn. Chúng tôi gọi đây là chỉ mục
và nó có thể được triển khai với + -prescan được phân đoạn.

1.6.1 Ví dụ: Vẽ đường

Để làm ví dụ về cách phân bổ được sử dụng, hãy xem xét việc vẽ đường. Các
bài toán vẽ đoạn thẳng là: cho trước một tập hợp các cặp điểm

$$\langle (X\,0\,,\,y\,0\,):(x\,0\,,\,y\,0\,)\rangle\,,\ \ldots,\ \langle (x\,n-1\,,\,y\,n-1\,):(x\,n-1\,,\,y\,n-1\,)\rangle\,,$$

tạo ra tất cả các vị trí của pixel nằm giữa các cặp điểm.
Hình 1.13 minh họa một ví dụ. Quy trình chúng ta thảo luận trả về một vectơ của

```
thủ tục vẽ dòng (x, y)
    song song cho mỗi dòng tôi
        % xác định độ dài của dòng
        length [i] ← max (| p 2 [i] .x - p 1 [i] .x |, | p 2 [i] .y - p 1 [i] .y |)

        % xác định gia số x và y
        Δ [i] .x ← (p 2 [i] .x - p 1 [i] .x) / length [i]
        Δ [i] .y ← (p 2 [i] .y - p 1 [i] .y) / length [i]

    % phân phối các giá trị và tạo chỉ mục
    p′1 ← phân phối (p 1 , độ dài)
    Δ′ ← phân phối (Δ, độ dài)
    chỉ mục ← chỉ mục (độ dài)

    song song cho mỗi pixel j
        % xác định vị trí cuối cùng
        kết quả [j] .x ← p′ 1 [j] .x + vòng (chỉ số [j] × Δ′ [j] .x)
        kết quả [j] .y ← p′ 1 [j] .y + vòng (chỉ số [j] × Δ′ [j] .y)
```

HÌNH 1.13
Các pixel được tạo ra bởi một thói quen vẽ đường thẳng. Trong ví dụ này, các điểm cuối là ⟨(11, 2): (23, 14)⟩ , ⟨(2, 13): (13, 8)⟩ , và ⟨(16, 4): (31, 4)⟩ . Thuật toán phân bổ 12, 11 và 16 pixel tương ứng cho ba dòng.

(x, y) các cặp xác định vị trí của mỗi pixel dọc theo mỗi dòng. Nếu một pixel xuất hiện trong nhiều dòng, nó sẽ xuất hiện nhiều hơn một lần trong vectơ. Quy trình tạo ra cùng một tập hợp các pixel như được tạo bởi kỹ thuật số đơn giản kỹ thuật tuần tự máy phân tích vi phân.

Ý tưởng cơ bản của quy trình là mỗi dòng phân bổ một bộ xử lý cho mỗi pixel trong dòng và sau đó cho mỗi pixel được phân bổ để xác định, trong song song, vị trí cuối cùng của nó trong lưới. Hình 1.13 cho thấy mã. Để phân bổ một bộ xử lý cho mỗi pixel, mỗi dòng trước tiên phải xác định số lượng pixel trong dòng. Con số này có thể được tính bằng cách lấy giá trị lớn nhất của x và y sự khác biệt của các điểm cuối của dòng. Mỗi dòng bây giờ phân bổ một phân đoạn của bộ xử lý cho các pixel của nó và phân phối một điểm cuối cùng với mỗi pixel gia số x và y trên toàn đoạn. Chúng tôi hiện có một bộ xử lý cho mỗi pixel và một đoạn cho mỗi dòng. Chúng ta có thể xem vị trí của một bộ xử lý trong phân đoạn của nó như là vị trí của một pixel trong dòng của nó. Dựa trên điểm cuối, độ dốc và vị trí trong dòng (được xác định bằng thao tác lập chỉ mục), mỗi pixel có thể xác định vị trí cuối cùng (x, y) của nó trong lưới.

Quy trình này có cùng độ phức tạp như quét T S (m, p), trong đó m là tổng số pixel. Để thực sự đặt các điểm trên một lưới, thay vì chỉ cần tạo ra vị trí của họ, chúng tôi sẽ cần hoán vị cờ thành một vị trí dựa trên vị trí của điểm. Nói chung, điều này sẽ yêu cầu đơn giản nhất dạng ghi đồng thời (một trong các giá trị được ghi), vì pixel có thể xuất hiện trong nhiều dòng.

1,7 Bài tập

1.1 Sửa đổi thuật toán trong Hình 1.4 để thực hiện quét thay vì quét trước.

1,2 Sử dụng thao tác quét để so sánh hai chuỗi có độ dài n trong O (n / p + lg p) thời gian trên EREW PRAM.

1,3 Cho hai vectơ bit đại diện cho các số nguyên không âm, hãy chỉ ra cách Có thể sử dụng tính năng quét trước để thêm hai số (trả về một vectơ gồm các bit biểu diễn tổng của hai số).

1,4 Theo dõi các bước của sắp xếp cơ số tách trên vectơ

[2 11 4 5 9 6 15 3].

1,5 Chỉ ra rằng phép trừ là bán liên kết và tìm toán tử đồng hành của nó.

1,6 Viết phương trình truy hồi dạng (1.5) đánh giá một đa thức

$$y = b_1 x_{n-1} + b_2 x_{n-2} + \cdots + b_{n-1} x + b_n$$

cho một x cho trước.

1,7 Chứng tỏ rằng nếu ⊗ có nghịch đảo, thì sự lặp lại của dạng (1.5) có thể được giải quyết với một số hoạt động cục bộ (không liên quan đến giao tiếp giữa các bộ xử lý) và hai hoạt động quét (sử dụng ⊗ và ⊕ làm toán tử).

1,8 Chứng minh rằng phép nhân vectơ-ma trận là nửa bán nghĩa.

1,9 Chứng minh rằng toán tử × s được xác định trong (1.17) là bán tương đối.

1.10 Hiển thị cách lặp lại x (i) = a (i) + b (i) / x (i - 1), trong đó + là số phép cộng và / là phép chia, có thể được chuyển thành dạng (1.11) với hai số hạng (m = 2).

1.11 Sử dụng quá trình quét để tạo n số Fibonacci đầu tiên.

1.12 Chỉ ra cách giải một hệ thống tuyến tính tam giác bằng cách sử dụng các lần lặp lại trong Phần-tion 1.4. Thuật toán có tối ưu về mặt tiệm cận không?

1.13 Trong ngôn ngữ Common Lisp, ký tự% có nghĩa là những gì theo sau ký tự đến cuối dòng là chú thích. Sử dụng thao tác quét để đánh dấu tất cả các ký tự nhận xét (mọi thứ từ% đến cuối-hàng).

1,14 Theo dõi các bước của đường nhanh song song trên vectơ

[27 11 51 5 49 36 15 23].

1,15 Mô tả cách thay đổi quicksort để nó chọn một phần tử ngẫu nhiên bên trong mỗi phân đoạn cho một trục.

1.16 Thiết kế một thuật toán cung cấp bán kính và số cạnh của một mặt phẳng chính quy polygon, xác định tất cả các pixel phác thảo đa giác.

Ghi chú và Tài liệu tham khảo

Phép toán tất cả tiền tố-tổng đã tồn tại trong nhiều thế kỷ như rence $x_i = a_i + x_{i-1}$. Đầu tiên là một mạch song song để thực hiện thao tác quét được đề xuất bởi Ofman (1963) để bổ sung các số nhị phân. Một song song việc thực hiện quét trên một mạng ngẫu nhiên hoàn hảo sau đó đã được đề xuất bởi Stone (1971) để đánh giá đa thức. Thuật toán tối ưu được thảo luận trong Phần 1.2 là một biến thể nhỏ của các thuật toán do Kogge và Stone đề xuất (1973) và của Stone (1975) trong bối cảnh của các phương trình lặp lại.

Ladner và Fischer (1980) lần đầu tiên cho thấy một mục đích chung hiệu quả cuit để thực hiện thao tác quét. Brent và Kung (1980), trong

ngữ cảnh bổ sung nhị phân, lần đầu tiên cho thấy bố cục VLSI hiệu quả để quét mạch điện. Các công việc gần đây hơn về việc triển khai các hoạt động quét song song bao gồm

công việc của Fich (1983) và của Lakshmivarahan, Yang và Dhall (1987), đưa ra những cải tiến đối với mạch của Ladner và Fischer, và của Lubachevsky và Greenberg (1987), chứng minh việc thực hiện quét hoạt động trên máy không đồng bộ. Blelloch (1987) cho rằng chắc chắn các hoạt động quét được bao gồm trong mô hình PRAM như là các hoạt động ban đầu và chỉ ra cách điều này ảnh hưởng đến độ phức tạp của các thuật toán khác nhau. Làm việc trên danh sách liên kết-hoạt động dựa trên tất cả tiền tố-tổng được xem xét và tham chiếu trong Chương 2, 3 và 4.

Các thuật toán sắp xếp theo đường nhìn và cơ số được Blelloch thảo luận (1988, 1990). Giải pháp song song của các vấn đề lặp lại lần đầu tiên được thảo luận của Karp, Miller và Winograd (1967), và các thuật toán song song để giải quyết chúng được đưa ra bởi Kogge và Stone (1973), Stone (1973, 1975) và Chen và Kuck (1975). Hyafil và Kung (1977) chỉ ra rằng độ phức tạp (1.10) thấp hơn ràng buộc.

Schwartz (1980) và Mago (1979) lần đầu tiên đề xuất các phiên bản quét được phân đoạn. Blelloch (1990) đề xuất nhiều cách sử dụng những lần quét này bao gồm cả thuật toán nhanh chóng và thuật toán vẽ đường được trình bày trong Phần 1 và 1.

Tôi muốn cảm ơn Siddhartha Chatterjee, Jonathan Hardwick và Jay Sipelstein vì đã đọc bản nháp của chương này.

Thư mục

Blelloch, GE, Quét dưới dạng Hoạt động Song song Nguyên thủy. Giao dịch IEEE trên Computers, C-38 (11): 1526–1538, tháng 11 năm 1989.

Blelloch, GE, Mô hình vectơ cho Máy tính Song song Dữ liệu. MIT Press, Cambridge, MA, 1990.

Blelloch, GE và Little, JJ, Giải pháp song song cho các vấn đề hình học trên máy quét Mô hình tính toán. Trong Kỷ yếu Hội nghị Quốc tế về Song song Đang xử lý, trang Vol 3: 218–222, tháng 8 năm 1988.

Brent, RP và Kung, HT, Độ phức tạp chip của số học nhị phân. Trong Proceedings ACM Symposium on Theory of Computing, trang 190-200, 1980.

Chen, S. và Kuck, DJ, Giới hạn thời gian và bộ xử lý song song để lặp lại tuyến tính Hệ thống. Giao dịch IEEE trên Máy tính, C-24 (7), tháng 7 năm 1975.

Fich, FE, New Bounds cho các mạch tiền tố song song. Trong Kỷ yếu Hội nghị chuyên đề ACM về Lý thuyết Máy tính, trang 100–109, tháng 4 năm 1983.

Hyafil, L., và Kung, HT, Sự phức tạp của đánh giá song song của tuần hoàn tuyến tính-rences. Tạp chí của Hiệp hội Máy tính, 24 (3): 513–521, Tháng 7 năm 1977.

Karp, RH, Miller, RE, và Winograd S., Tổ chức tính toán cho Phương trình lặp lại đồng nhất. Tạp chí của Hiệp hội Máy tính Máy móc, 14: 563–590, 1967.

Kogge, PM và Stone, HS, Một thuật toán song song cho giải pháp hiệu quả của một Loại chung của phương trình lặp lại. Giao dịch IEEE trên Máy tính, C-22 (8): 786–793, tháng 8 năm 1973.
Ladner, RE và Fischer, MJ, Tính toán tiền tố song song. Tạp chí Association for Computing Machinery, 27 (4): 831–838, tháng 10 năm 1980.
Lakshmivarahan, S., Yang, CM và Dhall, SK, Mạch tiền tố song song tối ưu với (kích thước + độ sâu) = 2n − n và ⌈log n⌉ ≤ độ sâu ≤ ⌈2 log n⌉ − 3. Trong Kỷ yếu Hội nghị Quốc tế về Xử lý Song song, trang 58–65, tháng 8 năm 1987.
Lubachevsky, BD và Greenberg, AG, Song song không đồng bộ đơn giản, hiệu quả Thuật toán tiền tố. Trong Kỷ yếu Hội thảo Quốc tế về Prollel Processing, trang 66–69, tháng 8 năm 1987.
Mago, GA, Một mạng máy tính để thực thi các ngôn ngữ rút gọn. Quốc tế Tạp chí Khoa học Máy tính và Thông tin, 1979.
Ofman, Y., Về độ phức tạp của thuật toán của các hàm rời rạc. Vật lý Xô viết Doklady, 7 (7): 589–591, tháng 1 năm 1963.
Schwartz, JT, Siêu máy tính. Giao dịch ACM trên ngôn ngữ lập trình và Hệ thống, 2 (4): 484–521, tháng 10 năm 1980.
Đá, HS, Xử lý song song với Perfect Shuffle. Giao dịch IEEE trên Máy tính, C-20 (2): 153–161, 1971.
Stone, HS, Một thuật toán song song hiệu quả cho giải pháp của một tuyến tính ba giác Hệ phương trình. Tạp chí của Hiệp hội Máy tính Máy tính, 20 (1): 27–38, tháng 1 năm 1973.
Đá, HS, Giải Phương trình Tam giác Song song. Giao dịch ACM trên Mathematical Software, 1 (4): 289–307, tháng 12 năm 1975.