TỐI ƯU MÃ TRONG QUÁ TRÌNH BIÊN DỊCH

Phan Thị Tươi
Trường Đại học Bách khoa – Đại học Quốc Gia TP.HCM
(Bài nhận ngày 06 tháng 2 năm 2002, hoàn chỉnh sửa chữa ngày 11 tháng 3 năm 2002)

TÓM TẮT: Tối ưu mã trong quá trình biên dịch bao gồm việc: loại bỏ biểu thức con dùng chung, lan truyền bản sao chép, tối ưu vòng lặp và thay thế các phát biểu phức tạp bằng các phát biểu đơn giản. Bài báo sẽ trình bày các giải thuật để thực hiện tối ưu mã. Cuối bài báo sẽ là phần minh họa các kết quả nghiên cứu.

1. GIỚI THIỆU:

Chúng tôi đã có dịp trình bày giai đoạn một của bộ tối ưu mã, đó là phân tích dòng điều khiển và phân tích dòng dữ liệu. Quá trình phân tích này nhằm xác định cấu trúc điều khiển, các thông tin dữ liệu của chương trình. Trong bài báo này, tôi sẽ trình bày giai đoạn thứ hai của quá trình tối ưu mã, đó là quá trình chuyển mã. Giai đoạn này sẽ sử dụng thông tin của quá trình phân tích cấu trúc điều khiển và dữ liệu để thực hiện tối ưu mã.

2. TỐI ƯU CỤC BỘ TRONG KHỐI CƠ BẢN

Để tối ưu cục bộ cho khối cơ bản, ta dùng phương pháp xây dựng dag, sắp xếp lại khối cơ bản trên dag, loại bỏ biểu thức con dùng chung trên dag, xác định các biến được dùng trong khối cơ bản và các biến dùng bên ngoài phạm khối cơ bản [3, 4].

3. LOẠI BỎ DƯ THỪA

Trong phần này, chúng tôi sẽ nói đến việc loại bỏ dư thừa toàn cục trong chương trình. Loại bỏ dư thừa ở đây bao gồm: loại bỏ biểu thức con dùng chung, lan truyền bản sao chép, chuyển mã không đổi trong vòng lặp ra ngoài vòng lặp, thu giảm độ phức tạp.

3.1. Loại bỏ biểu thức con dùng chung

Quá trình phân tích biểu thức có sẵn cho ta tập AEin và AEout [1, 4] tại điểm vào và ra khỏi khối cơ bản, từ đó giúp ta loại bỏ biểu thức con dùng chung. Loại bỏ biểu thức con dùng chung là loại bỏ sự tính toán lại biểu thức đó (mà giá trị không đổi), thay thế nó bằng các sử dụng kết quả tính toán của biểu thức, giải thuật chi tiết được trình ở [4].

Ví dụ về loại bỏ biểu thức con dùng chung ở H.3.1

3.2. Lan truyền bản sao chép

Trước khi đi đến giải thuật lan truyền bản sao chép, chúng tôi xin trình bày một số thông tin cần thiết:

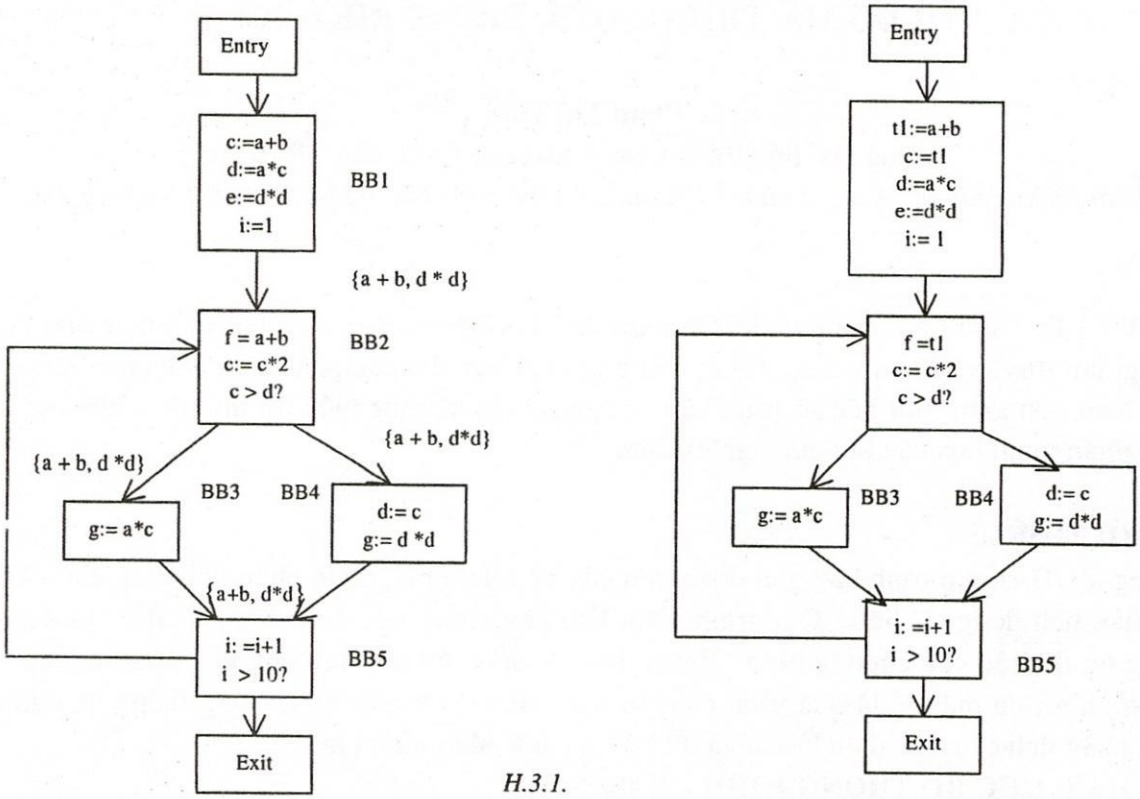
3.2.1. Dây xích sử dụng định nghĩa ud – chain (use –definition chain)

Tập các định nghĩa đến sự sử dụng a, a là một biến, được gọi là dây xích sử dụng định nghĩa, ký hiệu ud – chain. Ud – chain có thể được dẫn xuất từ đạt đến sự định nghĩa [1, 4] của giai đoạn phân tích dòng dữ liệu.

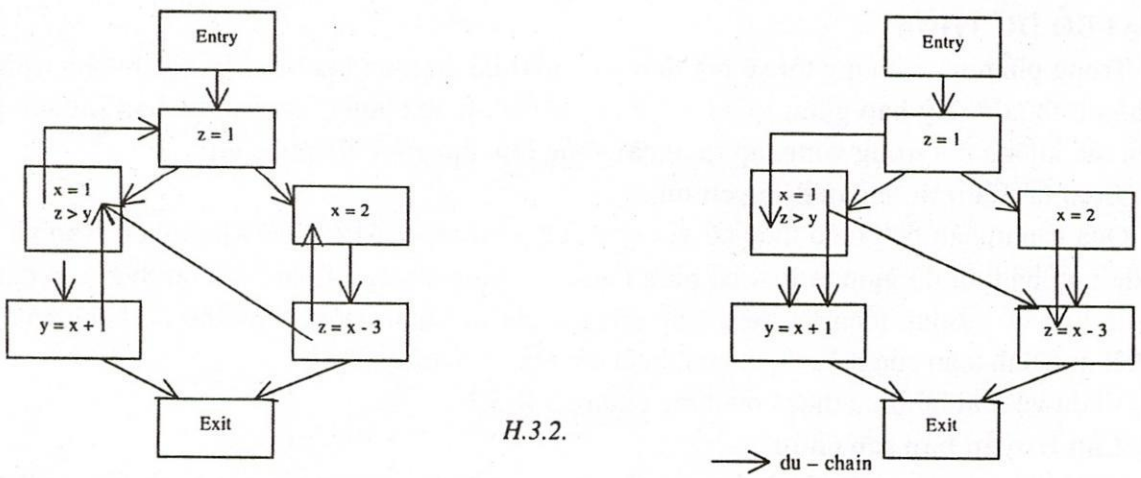
3.2.2. Dây xích định nghĩa sử dụng du – chain

Tập tất cả các sử dụng của một biến a, đặt tên bởi một định nghĩa được gọi là dây xích định nghĩa sử dụng du – chain.

Ví dụ về ud – chain và du – chain được trình bày ở H.3.2.



H.3.1.



H.3.2.

3.2.3. Biểu thức sao chép có sẵn (Available copy expression)

Biểu thức sao chép có sẵn tại một điểm vào của khối cơ bản có thể được xác định bằng việc phân tích dòng dữ liệu (giống như tìm biểu thức có sẵn của một khối cơ bản).

3.2.4. Tập sao chép

Tập các câu lệnh u: = v trong b, mà u và v không bị thay đổi giá trị trong b, đó là lệnh sao chép có sẵn tại điểm kết thúc của b.

3.2.5. Tập kill

Là tập các lệnh sao chép từ các khối cơ bản khác xuất hiện trong b, có toán hạng bị thay đổi giá trị trong b.

3.2.6. Sự cân bằng dòng dữ liệu

CopyIn: là tập lệnh sao chép có sẵn tại điểm vào khối cơ bản b.

CopyOut: là tập lệnh sao chép có sẵn tại điểm kết thúc của khối cơ bản b.

Công thức: $\text{CopyIn}(b) = \bigcap_{i \in pred(b)} \text{CopyOut}$

$$\text{CopyOut}(b) = \text{Copy}(b) \cup [\text{CopyIn}(b) - \text{Kill}(b)]$$

Giải thuật tìm bản sao chép có sẵn tại điểm vào khối cơ bản b: CopyIn (b) được trình bày chi tiết ở [4]

3.2.7. Lan truyền bản sao chép

Các lệnh sao chép có dạng x: = y

Lan truyền bản sao chép là sự thay thế việc sử dụng của x bằng y, mà không làm thay đổi trị x hoặc y.

Giải thuật 3.1: Lan truyền bản sao chép

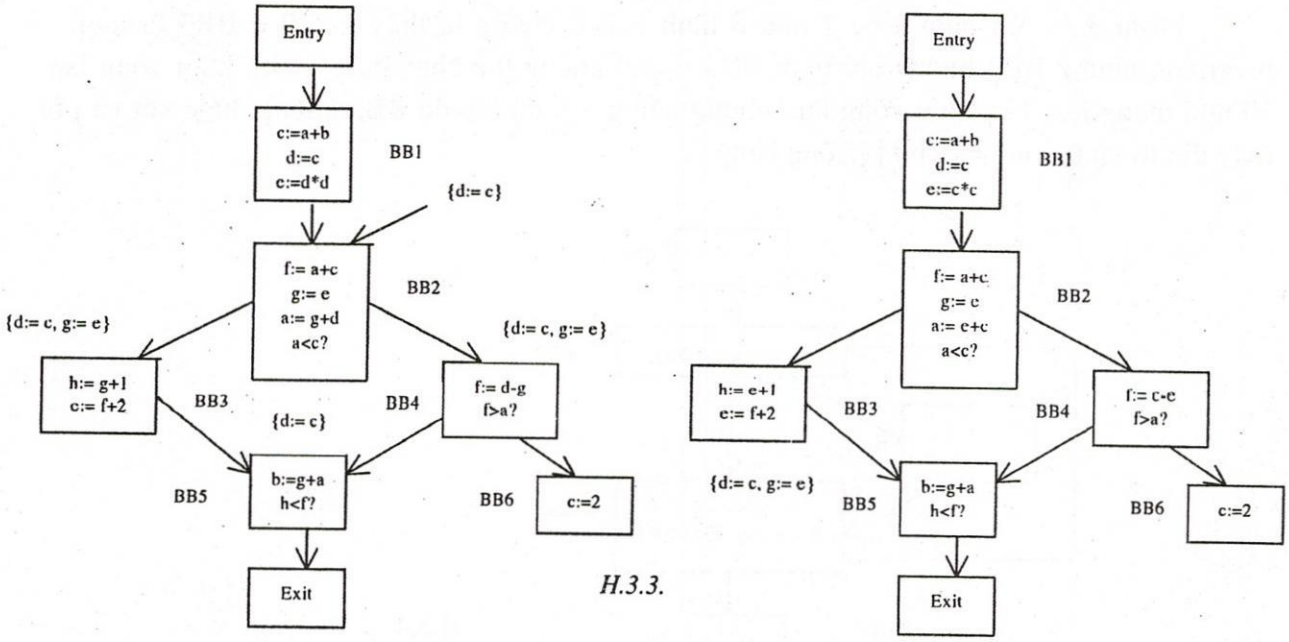
Nhập: đồ thị dòng điều khiển G với ud – chain và du –chain của các biến.

Xuất: đồ thị dòng điều khiển có sử dụng sự lan truyền bản sao chép

Giải thuật: với mỗi bản sao chép s, có dạng x: = y hãy thực hiện các bước sau:

1. Xác định các vị trí mà sự định nghĩa của x được sử dụng
2. Với mỗi sự sử dụng:
   a) s phải có sự định nghĩa của x, đạt đến u và
   b) mọi con đường từ s đến u không có tác vụ gán trị cho y.

Ví dụ về lan truyền bản sao chép ở H.3.3.



H.3.3.

3.3. Di chuyển mã không đổi của vòng lặp

Sự tính toán bên trong một vòng lặp được gọi là loop – invariant nếu sự tính toán đó luôn cho ra cùng một giá trị.

Di chuyển mã không đổi ra khỏi vòng lặp là di chuyển các loop – invariant ra bên ngoài vòng lặp. Ví dụ ta có tác vụ x: = y + z ở trong vòng lặp, nhưng tất cả sự định nghĩa của y và z ở ngoài vòng lặp thì y + z được gọi là loop – invariant. Vì thế x: = y + z sẽ được mang ra ngoài vòng lặp.

3.3.1. Tính toán không đổi

Mỗi tác vụ trong vòng lặp được gọi là loop – invariant nếu với mỗi toán hạng trong tác vụ là hằng số hoặc tất cả các định nghĩa cho toán hạng đều ở bên ngoài vòng lặp hoặc chỉ có duy nhất một định nghĩa cho toán hạng nằm trong vòng lặp mà sự định nghĩa này là loop – invariant.

Giải thuật 3.2.: Xác định loop – invariant

Nhập: Vòng lặp L, với ud – chain đã được xác định

Xuất: Các loop – invariant

Giải thuật: 1. Đánh dấu các lệnh có các toán hạng là hằng số hoặc tất cả sự định nghĩa của chúng nằm ngoài vòng lặp.

2. Lặp lại bước 1 cho đến khi không có câu lệnh nào được đánh dấu là loop – invariant. Đánh dấu các câu lệnh mà tất cả các toán hạng được đánh dấu là loop – invariant, hoặc các toán hạng có sự định nghĩa duy nhất ở trong vòng lặp mà sự định nghĩa này là một câu lệnh đã được đánh dấu.

3.3.2. Thực hiện di chuyển mã ra khỏi vòng lặp

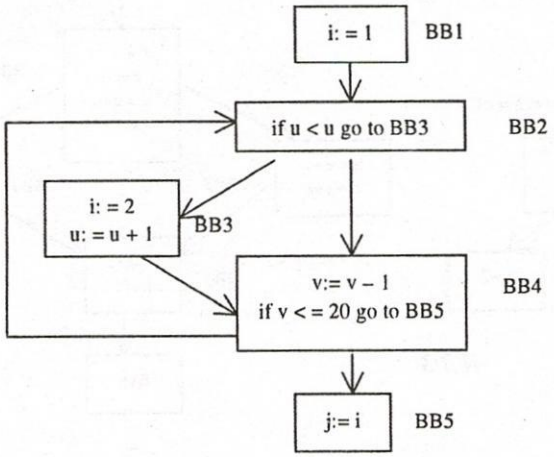
Sau khi xác định được tất cả các loop – invariant ta bắt đầu di chuyển mã ra khỏi vòng lặp. Sự di chuyển này phải thỏa đồng thời 3 điều kiện đối với phát biểu s: x = y+z

i) Khối chứa s chi phối các lối ra của vòng lặp

ii) Không có phát biểu gán trị cho x

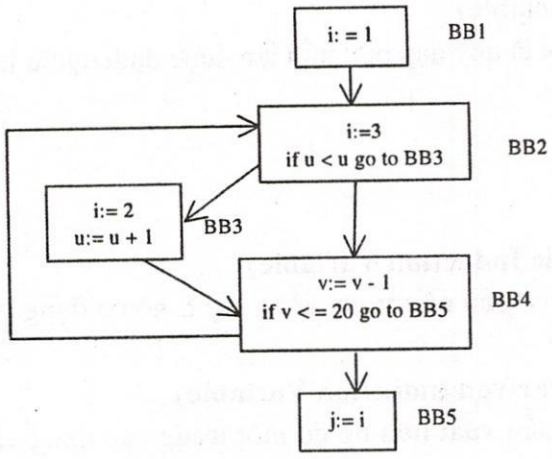
iii) Tất cả sự sử dụng của x phải được định nghĩa trong s ở trong khối cơ bản

Ví dụ 3.1.: Về điều kiện 1 như ở hình H.3.4. chúng ta thấy i: = 2 ở BB3 là loop – invariant, nhưng BB3 không chi phối BB4 vì thế không thể chuyển i: = 2 ra khỏi vòng lặp. Vì nếu mang i: = 2 ra khỏi vòng lặp, nhưng nếu u > v thì lúc đó BB3 không được xét và j bị thay đổi trị cụ thể là j =1 chứ j không bằng 2.



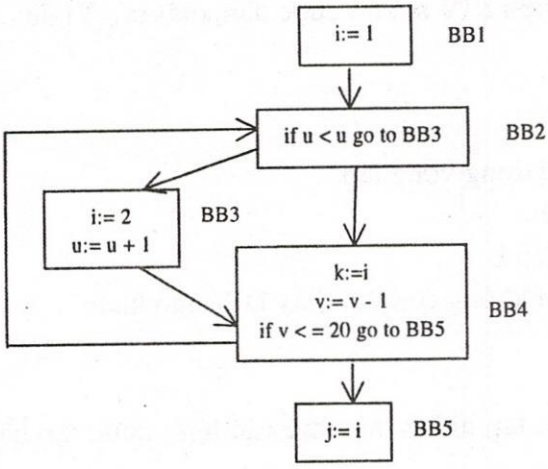
H.3.4.

Ví dụ 3.2.: Xét điều kiện 2 ở hình H.3.5. thì i := 3 ở BB2 là loop – invariant, BB2 chi phối BB4 là lối ra của vòng lặp nên thỏa điều kiện 1. Tuy nhiên i lại được định nghĩa lại trong BB3 nên không thể mang i : = 3 ra ngoài vòng lặp.



H.3.5.

Ví dụ 3.3.: Xét điều kiện 3 ở H.3.6., ta thấy k:= i trong BB4 được đạt đến bởi i = 1 trong BB2 và i = 2 trong BB3 vì thế không di chuyển k : = i ra khỏi vòng lặp



H.3.6.

Giải thuật 3.3.: Di chuyển mã ra khỏi vòng lặp
Nhập: vòng lặp L và loop – invariant trong L
Xuất: vòng lặp đã di chuyển mã

Giải thuật: *for* với mỗi câu lệnh s định nghĩa v *do*

if 1. s chi phối tất cả các lối ra của L
2. không được định nghĩa nơi nào khác trong L
3. tất cả sự sử dụng của v trong L chỉ được định nghĩa trong s *then* di chuyển câu lệnh loop – invariant ra pre – header của L.

4. TỐI ƯU VÒNG LẶP

Để tối ưu vòng lặp chúng ta sẽ dùng giải thuật thu giảm độ phức tạp "strength reduction". Giải thuật này thay thế các lệnh phức tạp bằng câu lệnh đơn giản hơn. Để đi đến giải thuật trước tiên ta hãy định nghĩa một số khái niệm sau:

4.1. Biến quy nạp (Introduction variable)

Trong vòng lặp L, biến x được gọi là quy nạp nếu mỗi lần được định nghĩa lại nó sẽ tăng hoặc giảm một hằng số.

Ví dụ: Do i = 1,100
a(i) : = 202 – 2 * i
Enddo

4.2. Biến quy nạp cơ bản BIV (Basic Induction Variable)

Một biến v được gọi là quy nạp cơ bản nếu trong vòng lặp L nó có dạng v: = v ± c với c là hằng số.

4.3. Biến quy nạp dẫn xuất DIV (Derived Induction Variable)

Biến j được gọi làbiến quy nạp dẫn xuất nếu nó có một trong các dạng sau, xuất hiện trong vòng lặp:

j:= a* i hoặc j:= i * a
j: = b ± i hoặc j: = i ± b
j:= i/a Với i là biến quy nạp cơ bản

Như vậy trị của j bị thay đổi tùy thuộc vào sự thay đổi trị của i. Từ đó ta biểu diễn j bằng bộ tam, là hàm thay đổi theo i, là biến BIV mà nó được dẫn xuất ra. Ví dụ:

j:= a * i → (i, a, 0)
j:= b + i → (i, 1, b)

Ta nói j thuộc họ i

Giải thuật 4.1.: Tìm biến thay đổi trong vòng lặp

1. Tìm tất cả các BIV trong L.
2. Tìm các DIV trong vòng lặp L.
3. Lặp lại bước 2 cho đến khi không còn tìm thấy DIV nào khác.

4.4. Thu giảm độ phức tạp

Giải thuật 4.2.: Thu giảm độ phức tạp nhằm thay thế các lệnh phức tạp bằng các lệnh đơn giản.

Nhập: Vòng lặp L và tập họ các biến quy nạp

Xuất: Đoạn mã đã thực hiện thay thế các câu lệnh phức tạp bằng các câu lệnh đơn giản hơn trong vòng lặp L.

Giai thuật: *for* với mỗi biến BIV_i *do begin*
for với mỗi biến j trong họ i là bộ tam (i, a, b) *do begin*

Tạo ra biến sj và khởi động câu lệnh sj: = a * i + b và đặt vào pre – header của L; thay thế câu lệnh gán cho j bằng j: = sj;

for sau mỗi phát biểu i: = i ± c trong L *do*
thêm sj: = sj ± c * a; end;
thêm sj vào họ của i; end;

Ví dụ 4.1.: đoạn chương trình ở hình H.4.1. sau khi được áp dụng giải thuật thu giảm độ phức tạp cho ra đoạn chương trình ở hình H.4.2.

(1) i : = 1
(2) if i > 100 go to (13)
(3) to : = 202
(4) t_1 : = i * 2 ← (i, 2, 0)
(5) t_2: = t_0 – t_1
(6) t_3: = add (a)
(7) t_4: = t_3 – 4
(8) t_5: = 4 * i ← (i, 4, 0)
(9) t_6: = t_4 + t_5
(10) [t_6] : = t_2
(11) i: = i + 1
(12) go to (2)

H.4.1. Đoạn mã ba địa chỉ

(1) i : = 1
(2) s_1 : = 2 * i
(3) s_2 : = 4 * i
(4) if i > 100 go to (17)
(5) t_0: = 202
(6) t_1: = s_1
(7) t_2: = t_0 – t_1
(8) t_3: = add (a)
(9) t_4: = t_3 – 4
(10) t_5 : = s_2
(11) t_6 : = t_4 + t_5
(12) [t_6] : = t_2
(13) i: = i + 1
(14) s_1: = s_2 + 2
(15) s_2: = s_2 + 4
(16) go to (4)

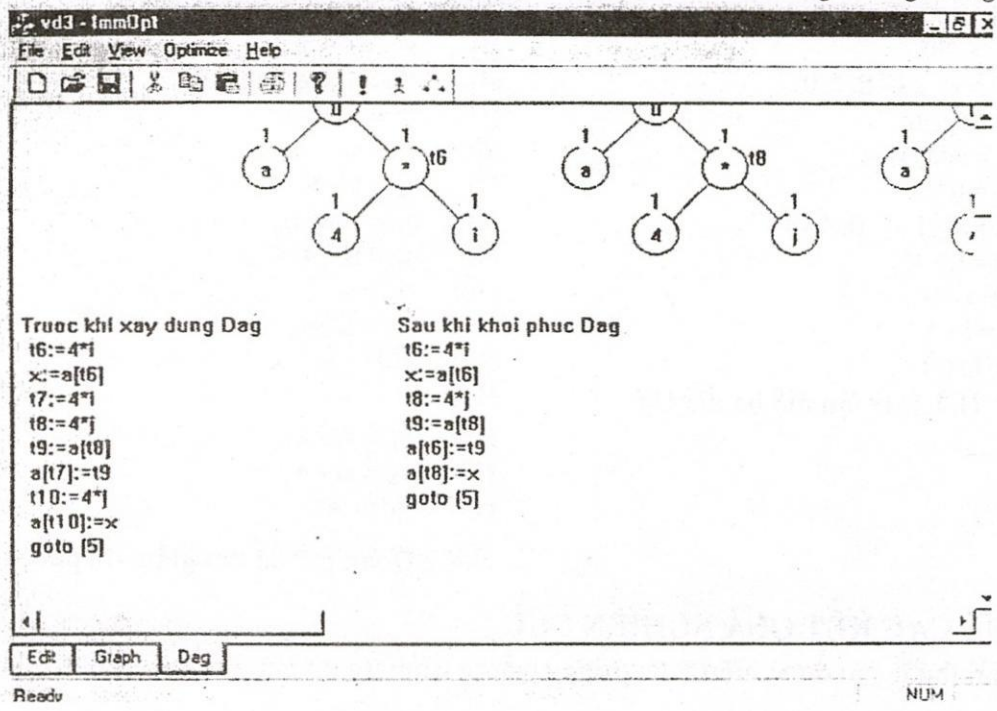
H.4.2 Đoạn mã đã thu giảm độ phức tạp

5. MINH HỌA CÁC KẾT QUẢ NGHIÊN CỨU

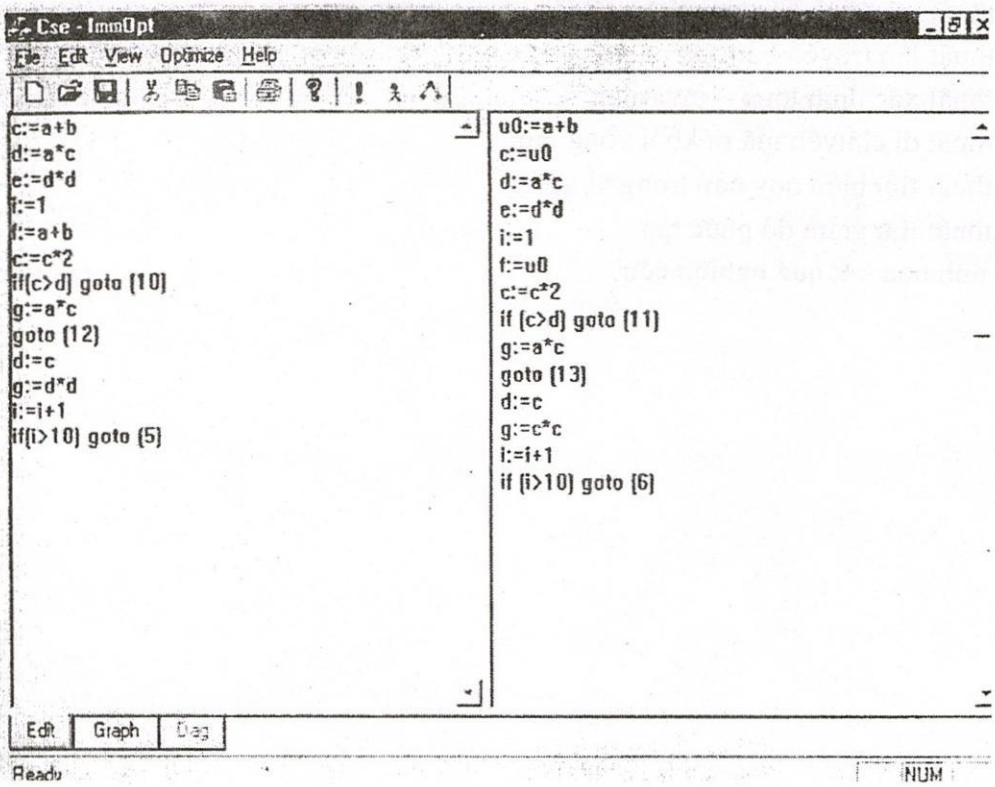
Các giải thuật sau được hiện thực trong chương trình tối ưu mã ở [4]

5.1. Giải thuật xây dựng Dag
5.2. Giải thuật loại bỏ biểu thức con dùng chung
5.3. Giải thuật xác định bản sao chép có sẵn
5.4. Giải thuật lan truyền bản sao chép
5.5. Giải thuật xác định loop – invariant
5.6. Giải thuật di chuyển mã ra khỏi vòng lặp
5.7. Giải thuật tìm biến quy nạp trong vòng lặp
5.8. Giải thuật thu giảm độ phức tạp
5.9. Các minh họa kết quả nghiên cứu

a. Xây dựng Dag cho khối cơ bản và loại bỏ biểu thức con dùng chung trong khối cơ bản

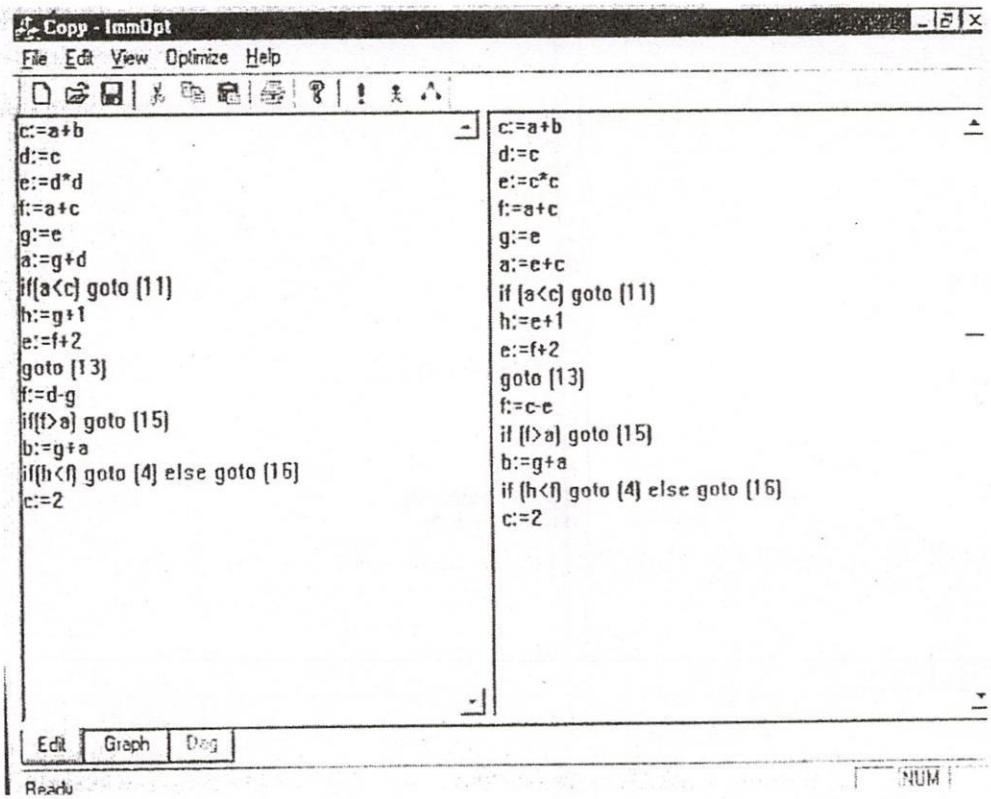


b. Loại bỏ biểu thức con dùng chung trong chương trình

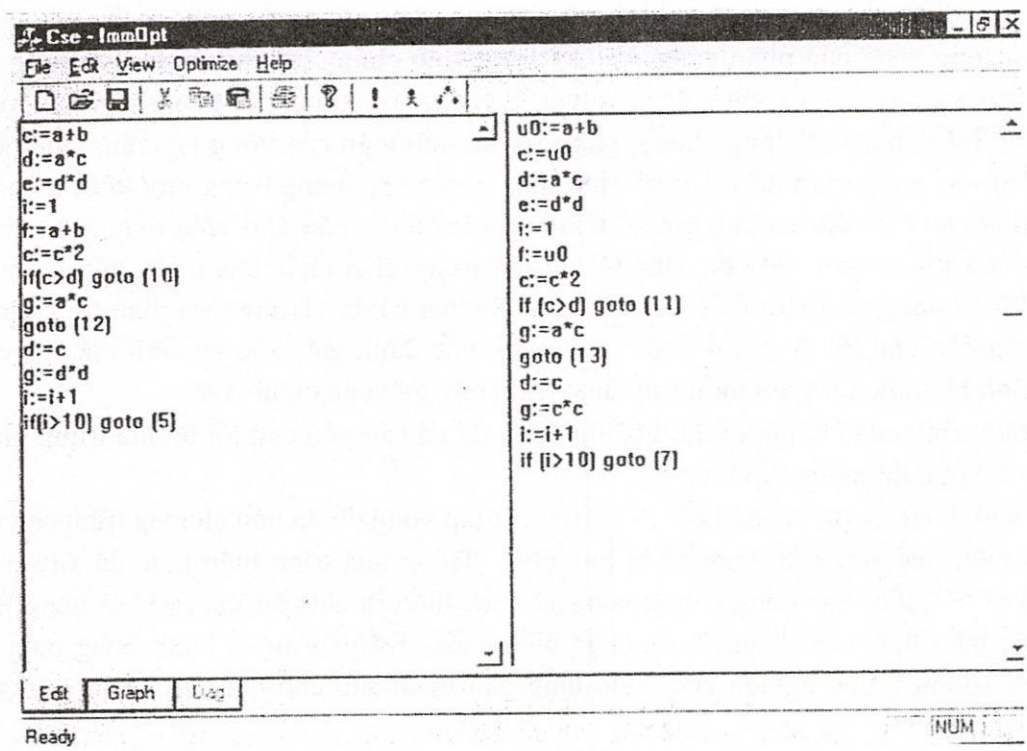


❖ *Lưu ý*: cửa sổ bên trái là chương trình trước khi loại bỏ biểu thức con dùng chung. Cửa sổ bên phải là sau khi thực hiện loại bỏ

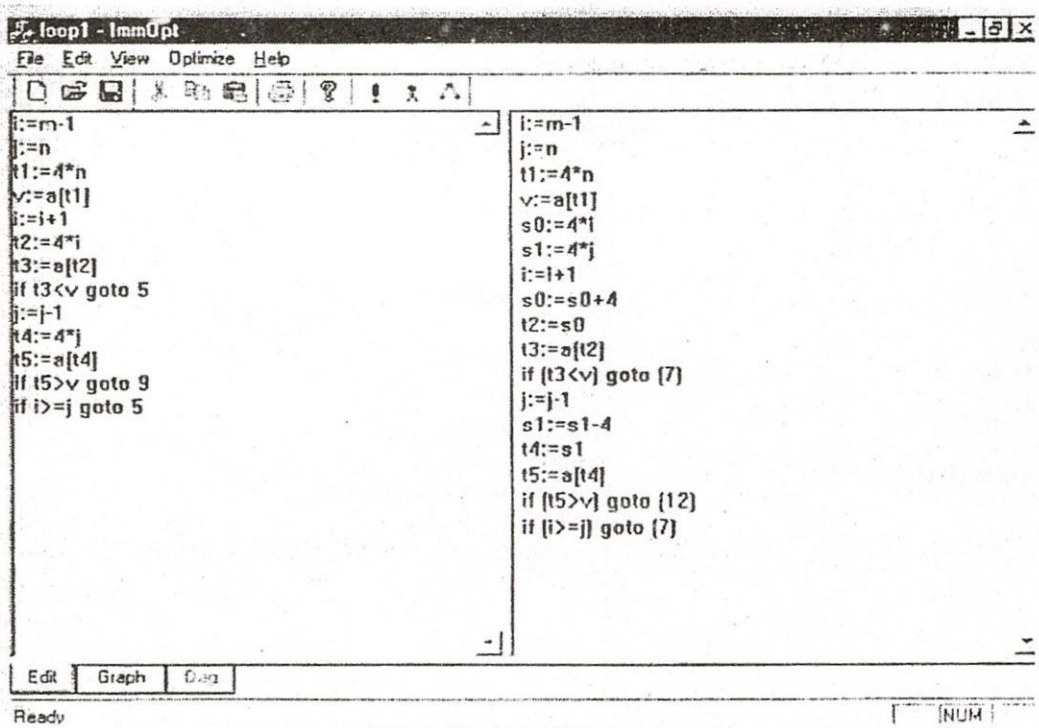
c. Lan truyền bản sao chép



d. Di chuyển mã không đổi ra khỏi vòng lặp



e. Thu giảm độ phức tạp



KẾT LUẬN

Đây là một nghiên cứu và thử nghiệm chương trình về tối ưu mã trung gian của trình biên dịch. Còn tối ưu mã đối tượng phải được xét trên một máy tính (dù là giả lập hay thật) vì nó liên quan đến việc sử dụng thanh ghi và tận dụng khả năng truy xuất cache của bộ nhớ và khả năng song song của phần cứng. Vì vậy trước tiên chúng tôi chỉ tối ưu mã trung gian. Tối ưu mã trung gian là một công việc sẽ quyết định đến tối ưu mã đối tượng vì nó đã loại bỏ sự dư thừa về biểu thức con dùng chung, giảm số lần tính toán của vòng lặp cũng như nó sắp xếp lại thứ tự mã trung gian để giảm số biểu thức con dùng chung trong một khối cơ bản và trong giai đoạn này nó cũng đánh giá số thanh ghi tối thiểu cần cho việc thực hiện chương trình của một khối cơ bản. Do đó việc tối ưu mã trung gian chắc chắn dẫn đến giảm kích thước bộ nhớ sử dụng và giảm thời gian thực thi chương trình. Trong thời gian tới chúng tôi sẽ tiếp tục nghiên cứu tối ưu mã đối tượng, lúc đó việc đánh giá tỷ lệ về sinh mã của chúng tôi so với trình biên dịch truyền thống sẽ được trình bày một các chính xác.

Chương trình của chúng tôi đã thực thi được tất cả các yêu cầu tối ưu mã trung gian ba địa chỉ, tốc độ thực thi tương đối tốt.

Các giải thuật tối ưu mã hầu hết là vét cạn và lặp vòng do đó nếu chương trình cần tối ưu lớn thì tính hiệu quả của giải thuật sẽ bị hạn chế. Trong quá trình hiện thực đã xảy ra hiện tượng khi loại bỏ biểu thức dùng chung xong và thực hiện di chuyển mã ra khỏi vòng lặp thì xuất hiện lại biểu thức con dùng chung hoặc biến chết. Để giải quyết hiện tượng này, trong thời gian tới chúng tôi sẽ nghiên cứu, xây dựng giải thuật sao cho hai vấn đề này được giải quyết trọn vẹn.

CODE OPTIMIZATION IN A COMPILE PROCESS

Phan Thi Tuoi
University of Technology – VNU-HCM
(Received 06 February 2002, Revised 11 March 2002)

ABSTRACT: Code optimization of a compile process consists of common subexpression elimination, copy propagation, loop optimization and strength reduction. In this paper, algorithms of these transformations are presented and some demonstrations of the study result are also shown at the end of the paper.

TÀI LIỆU THAM KHẢO

[1] Tarek S. Abdelrahman, ECE 1724 Special Topic in Software Engineering Optimizing Compilers.
[2] Alfred V. Aho, Jeffrey D. Ullman, Compolers principles, Techniques and Tools, Printice Hall Inc. 1986.
[3] Phan Thị Tươi, Trình biên dịch, NXB Đại học Quốc gia TPHCM, 2001,
[4] Lê Hồng Sơn, GVHD: Phan Thị Tươi, Luận văn tốt nghiệp, Trường Đại học Bách Khoa – ĐHQG TPHCM, 01/2002.
[5] Steven S.Muchnick, Advanced Compiler Design Implementation, Morgan Kaufman Publishers, 1997.