최적화 코드 모션을 위한 알고리즘

。신현덕, 심손권, 안희학
관동대학교 전자계산공학과
e-mail:ubhd@mail.kwandong.ac.kr
sksim@mail.kwandong.ac.kr
hhahn@mail.kwandong.ac.kr

An Algorithm for Optimal Code Motions

Hyun-Deok Shin, Son-Kweon Sim, Heui-Hak Ahn
Department of Computer Science Kwandong University

요약

본 논문에서는 코드 최적화를 위하여 계산적으로나 수명적으로 제한이 없는 배정문 모션 알고리즘을 제안한다. 이 알고리즘은 지나친 레지스터의 사용을 막기 위하여 불필요한 코드 모션을 억제한다. 또한, 본 논문에서는 기존 알고리즘의 술어의 의미가 명확하지 않은 것을 개선하였고 노드 단위 분석과 명령어 단위 분석을 혼용했기 때문에 발생하는 모호함도 개선하였다. 따라서, 제안한 알고리즘은 불필요하게 중복된 수식이나 배정문의 수행을 피하게 함으로써, 프로그램의 불필요한 재계산이나 재실행을 하지 않게 하여 프로그램의 능률 및 실행시간을 향상시킨다.

1. 서론

코드 최적화는 실행시간에 불필요한 값의 재 계산을 피하기 위해 프로그램을 계산적으로나 수명적으로 최적인 상태로 변환하여 프로그램의 능률을 개선하는 기술이다[1].

수명 최적화(lifetime optimal)는 프로그램의 계산적 최적성이 유지되는 범위 내에서 식을 가능한 한 늦게 두는 것으로서 이 전략에 의해 계산적 최적화 과정에서 도입된 임시변수의 수명을 최소로 줄일 수 있다[2].

프로그램을 계산적으로나 수명적으로 최적화하는 기법에는 수식 모션(EM : Expression Motion) 변환[2, 3, 4, 5, 6]과 수식 모션을 포함하는 배정문 모션(AM : Assignment Motion) 변환[7, 8, 9, 10]이 있다. 제안된 알고리즘은 수식모션을 포함하는 배정문 모션 변환 알고리즘으로서 모든 배정문에 임시 변수를 도입하고 최대 상위(as-early-as-possible) 재 배치 전략으로 가능한 모든 배정문을 끌어올려서 중복된 배정문을 제거한다. 이 단계는 계산적 최적화 단계이며 이 단계에서 불필요한 코드 모션이 일어날 수 있다. 따라서, 최대 하위(as-late-as-possible) 재 배치 전략을 이용하여 불필요한 코드 모션을 억제한다.

2. 수식 모션 변환

수식 모션은 안전한 삽입위치 중에서 가장 이른 삽입위치에 t를 삽입한다. 따라서 ∀n∈N에 대한 Earliest(n)은 (정의 2-1)과 같다[2, 11, 12].

(정의 2-1) Earliest

$$Earliest(n) = Safe(n) \wedge \begin{cases} true & \text{if } n = s \\ \bigvee_{m \in pred(n)} \neg Transparent(m) \wedge \neg Safe(m) & otherwise \end{cases}$$

수식 모션에서의 초기화는 계산적 최적성이 유지되는 한 시작 노드에서 마지막 노드까지의 경로상에서 지연될 수 있다는 특징으로 Delayability를 정의하고, 불필요한 코드 모션을 억제하기 위해서 실행

시간을 향상시키지 못하는 코드 모션은 억제되어야 한다는 특징으로 Latest를 정의한다[2, 11, 12].

(정의 2-2) Delayability

$\forall n \in N,\ Delayed(n) \Leftrightarrow_{df}$
$\forall p \in P[s, n]\ \exists i \le \lambda_p.\ Earliest(p_i) \wedge$
$\neg Computation^{\exists}(p[i, \lambda_p[)$

(정의 2-3) Latest

$\forall n \in N, Latest(n) = {}_{df}\ Delayed(n) \wedge$
$\left(Comp(n) \vee \bigvee_{m \in succ(n)} \neg Delayed(m)\right)$

프로그램의 실행 시간을 향상시키는 코드 모션일지라도 불필요한 임시변수 초기화를 실행할 수 있다는 특징으로 Isolated를 정의한다.[2, 11, 12].

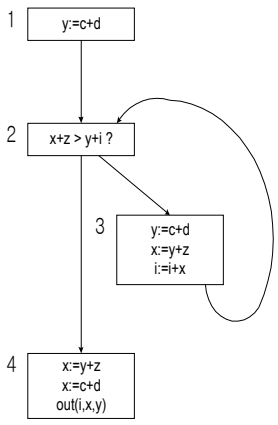
(정의 2-4) Isolation

$\forall CM \in cm\ \forall n \in N,$
$Isolated_{CM}(n) \Leftrightarrow_{df}$
$\forall p \in P[n, e]\ \forall 1 < i \le \lambda_p,\ Replace_{CM}(p_i)$
$\Rightarrow Insert^{\exists}{}_{CM}(p]1, i])$

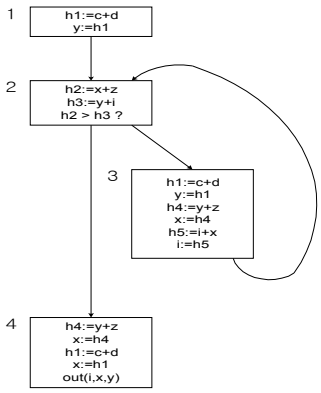
3. 코드 최적화 알고리즘

3.1 초기 설정 단계

초기 설정 단계에서는 임시 기억장소를 도입해서 프로그램 내의 모든 배정문을 분해한다. 모든 배정문 x:=t를 배정문 h_t:=t; x:=h_t로 대체한다. h_t는 t항을 보관하는 유일한 임시 기억장소이다. 초기 설정 단계에서 배정문 모션은 수식 모션을 포함하게 된다. [그림 3-1]에 초기 설정 단계를 적용하면 [그림 3-2]와 같다.



[그림 3-1] 예제 흐름그래프



[그림 3-2] 초기 설정 단계 수행 결과

3.2 배정문 모션 단계

배정문 모션 단계에서는 LOC_HOISTABLE, LOC_BLOCKED, EXECUTED, ASS_TRANSP라는 지역 술어들을 이용하여 배정문 모션 후보를 프로그램의 안전한 위치로 끌어올린다.

노드 n에 대해서 배정문 패턴 α가 α≡ v:=t라 할 때, LOC_HOISTABLE은 n의 끌어올리기 후보 α를 의미한다. 또한 LOC_BLOCKED는 α의 끌어올리기가 노드 n의 다른 문장에 의해 블록 되었다는 것을 의미하고 EXECUTED는 현재 문장이 α 패턴의 배정문이라는 것을 의미하며 ASS_TRANSP는 배정문 패턴 α의 v뿐만 아니라 t의 어떤 피연산자도 현재 문장에 의해 수정되지 않는다는 것을 의미한다.

배정문 끌어올리기 알고리즘은 프로그램의 의미를 유지하면서 배정문을 본래의 위치로부터 프로그램의 가장 앞부분으로 끌어올리는 것이다.

```
procedure Find_HOISTABLE( )
 begin
  for i := 0 to FlowG_node_MAX do
        if (FlowG_node[i] == E_node) then
                X_HOISTABLE[i] := FALSE;
  for i := FlowG_node_MAX to 1 do
    begin
     for m := HOIST_SUCC_START(i) to
                                HOIST_SUCC_END(i)
        do Hoist_Succ_Sum := Hoist_Succ_Sum &&
                N_HOISTABLE[m];
          N_HOISTABLE[i] :=
          FlowG_node[i].LOC_HOISTABLE ||
          X_HOISTABLE[i] &&
          !FlowG_node[i].LOC-BLOCKED;
          X_HOISTABLE[i] := Hoist_Succ_Sum
    end
 end;
```

[알고리즘 1] 배정문 끌어올리기 알고리즘

[알고리즘 1]에서 끌어올리기 후보를 나타내는 술어 N_HOISTABLE과 X_HOISTABLE은 α의 끌어올리기 후보들을 기본 블록 n의 입력이나 출력 부분까지 각각 이동시킬 수 있다는 것을 의미한다.

```
procedure Find_ELIMINATION( )
 begin
  for i := 0 to FlowG_node_MAX do
    begin
        N_ELIMINATION[i] := FALSE;
        X_ELIMINATION[i] := FALSE
    end;
  for i :=  0 to FlowG_node_MAX do
    begin
        if (N_REDUNDANT[i]) then
          N_ELIMINATION[i] := N_REDUNDANT[i]
                && FlowG_node[i].EXECUTED;
        if (X_REDUNDANT[i]) then
           X_ELIMINATION[i] := X_REDUNDANT[i]
                && FlowG_node[i].EXECUTED
    end
 end;
```

[알고리즘 2] 중복 배정문 제거 알고리즘

[알고리즘 2]에서 N_ELIMINATION과 X_ELIMINATION은 기본 블록 n의 입력이나 출력 부분에서 중복인 배정문 α를 제거한다.

배정문 h4 := y+ z에 대한 ELIMINATION은 [알고리즘 2]에 의해 다음과 같이 계산된다.

$X\_ELIMINATION_3 = T \wedge T = T$

$X\_ELIMINATION_4 = T \wedge T = T$

3.3 불필요한 코드 모션 재구성 단계

불필요한 코드 모션 재구성 단계에서는 프로그램의 의미를 유지하면서 h_ε:=ε(ε는 수식 패턴) 형태의 모든 배정문을 프로그램의 가장 뒷부분으로 옮기고, h_ε가 배정문의 실제 값 이후에 맞아야 한번 이용되는 모든 실제 값을 제거한다.

```
procedure Find_RECONSTRUCT( )
 begin
  for i := 0 to FlowG_node_MAX do
    RECONSTRUCT[i] := FALSE;
  for i :=  0 to FlowG_node_MAX do
    begin
    if (N_INIT[i]) then
       RECONSTRUCT[i] :=
            FlowG_node[i].USED&&N_INIT[i] &&
                                  !X_USABLE[i];
    if (INITELIM[i]) then RECONSTRUCT[i] :=
                                  INITELIM[i]
    end
 end;
```

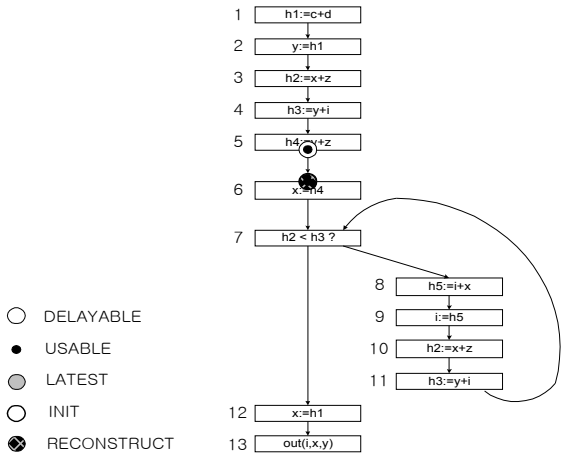
[알고리즘 3] 재구성 알고리즘

[알고리즘 3]에서 RECONSTRUCT는 v := h_ε 형태의 배정문에서 h_ε의 값을 원래의 식으로 재구성할 위치를 결정한다.

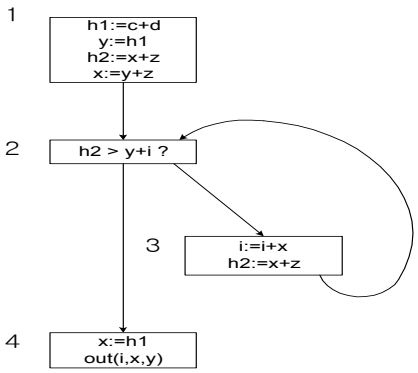
배정문 h4 := y+ z에 대한 RECONSTRUCT는 [알고리즘 3]에 의해 다음과 같이 계산된다.

$RECONSTRUCT_6 = T \wedge T \wedge \neg F = T$

배정문 h4 := y+ z에 대한 DELAYABLE과 USABLE, LATEST, INIT, RECONSTRUCT를 [알고리즘 3]으로 계산한 결과는 [그림 3-3]과 같고 불필요한 코드 모션 재구성 단계를 적용한 결과는 [그림 3-4]와 같다.



[그림 3-3] DELAYABLE, USABLE, LATEST, INIT, RECONSTRUCT의 계산 결과



[그림 3-4] 재구성 단계 수행 결과

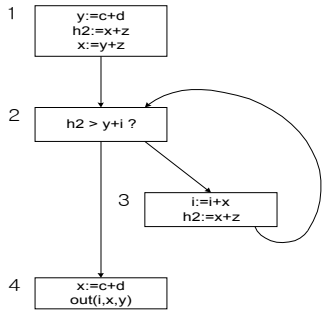
3.4 최종 최적화 단계

v:=h_ε 형태의 사용을 사용횟수에 포함시키는 것은 불필요한 코드 모션이며 레지스터 압박을 초래할 수 있다. 따라서 v:=h_ε의 형태로 사용된 h_ε은 사용횟수에서 제외하는 알고리즘을 제안한다.

```
procedure Find_INITELIM( )
 begin
   for i := 0 to FlowG_node_MAX do
    if (FlowG_node[i].ASSIGNMENT == INI_ASS)
    then INTELIM[i] := TRUE
end;
```

[알고리즘 4] 최종 최적화 단계 알고리즘

[그림 3-4]에 최종 최적화 단계를 적용하면 [그림 3-5]와 같다.

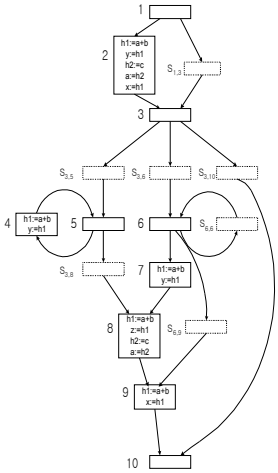


[그림 3-5] 최종 최적화 단계 수행 결과

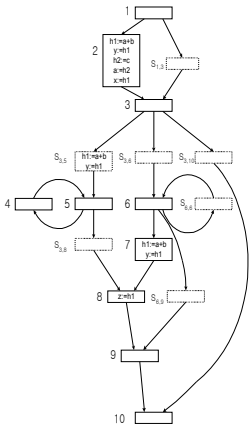
3.5 성능 분석

본 논문에서 제안한 배정문 모션 알고리즘의 성능 분석 결과는 다음과 같다.

[그림 3-6]은 예제 흐름그래프의 초기설정단계 수행결과를 나타내며 [그림 3-7]은 알고리즘을 적용한 결과를 나타낸다.



[그림 3-6] 예제 흐름그래프의 초기설정단계

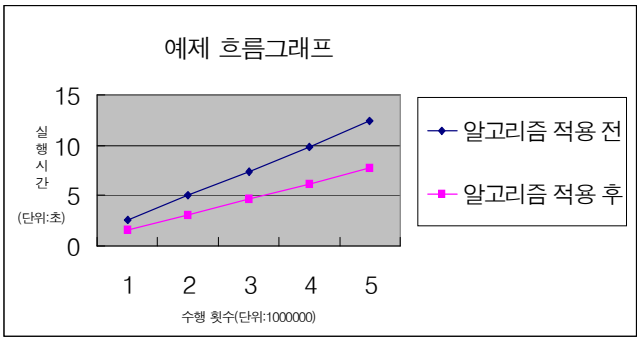


[그림 3-7] 예제 흐름그래프의 알고리즘 적용 결과

<표 3-1>과 [그림 3-8]은 예제 흐름그래프에 알고리즘을 적용시킨 결과를 나타낸다.

<표 3-1> 예제 흐름그래프의 알고리즘 수행 결과

구 분 / 수행 횟수	알고리즘 적용 전	알고리즘 적용 후
1000000	2.527473	1.593407
2000000	5.000000	3.076923
3000000	7.417582	4.615385
4000000	9.890110	6.208791
5000000	12.362637	7.692308



[그림 3-8] 예제 흐름그래프의 알고리즘 수행 결과 분석

성능 평가 결과, 프로그램 내에 복잡한 반복문이 많고 반복문 내의 중복 코드가 많을수록 제안한 알고리즘을 적용한 경우의 효과가 크다는 것을 알 수 있다. 따라서, 제안한 알고리즘은 프로그램의 구조가 복잡하고 루프의 횟수가 많을 수록 적용 효과가 높다는 것을 알 수 있다.

4. 결론

본 논문에서는 코드 최적화를 위하여 계산적, 수명적으로 제한이 없는 배정문 모션 알고리즘을 제안했다. 이 알고리즘은 배정문 모션 변환 알고리즘으로서 모든 배정문에 임시 변수를 도입하고 최대 상위 재 배치 전략으로 가능한 모든 배정문을 끌어올려서 중복된 배정문을 제거한다. 이 단계에서 불필요한 코드 모션이 일어날 수 있기 때문에 최대 하위 재 배치 전략을 이용하여 불필요한 코드 모션을 억제한다.

본 논문에서 제안한 알고리즘은 모든 불필요한 코드 모션을 억제시키기 때문에 계산적으로나 수명적으로 최적인 알고리즘이다.

본 논문에서 제안한 알고리즘을 병렬 프로그램의 코드 모션에 적용하여 재 구성하는 것이 향후 연구 과제이다.

참 고 문 헌

[1] Aho, A. V., Sethi, R., and Ullman, J. D., "*Compilers Principles, Techniques, and Tools*", Addison-wesley publishing Co., 1986.

[2] Knoop, J., Rüthing, O. and Steffen, B., "Optimal code motion: theory and practice", *ACM Transactions on Programming Languages and Systems*, Vol. 16, No. 4, pp. 1117-1155, 1994.

[3] Dhamdhere, D. M., "A fast algorithm for code movement optimization", *ACM SIGPLAN Notices*, Vol. 23, No. 10, pp. 172-180, 1998.

[4] Dhamdhere, D. M., Rosen, B. K. and Zadeck, F. K., "How to analyze large programs efficiently and informatively", In *Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation'92, of ACM SIGPLAN Notices*, Vol. 27, No. 7, pp. 212-223, San Francisco, CA, June 1992.

[5] Drechsler, K. H. and Stadel, M. P., "A Varia-tion of Knoop, Rüthing and Steffen's lazy code motion", *ACM SIGPLAN Notices*, Vol. 28, No. 5, pp 29-38, 1993.

[6] Morel, E. and Renvoise, C., "Global optimization by suppression of partial redundancies", *Communications of the ACM,* Vol. 22, No. 2, pp 96-103, 1979.

[7] Dhamdhere, D. M., "Register assignment using code placement techniques", *Journal of Computer Languages*, Vol. 13, No. 2, pp. 75-180, 1988.

[8] Dhamdhere, D. M., "A usually linear algorithm for register assignment using edge placement of load and store instructions", *Journal of Computer Languages*, Vol. 15, No. 2, pp. 83-94, 1990.

[9] Dhamdhere, D. M., "Practical adaptation of the global optimization algorithm of Morel and Renvoise", *ACM Transactions on Programming Languages and Systems*, Vol. 13, No. 2, pp. 291-294, 1991.

[10] Knoop, J., Rüthing, O. and Steffen, B., "The Power of Assignment Motion", *Proceedings of the Conference on Programming Language Design and Implementation*, Vol. 30, No. 6, pp. 233-245, 1995.

[11] Knoop, J., Rüthing, O. and Steffen, B., "Lazy code motion", In *Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation'92*, of *ACM SIGPLAN Notices*, Vol. 27, No. 7, pp. 224-234, San-Francisco. CA, June 1992.

[12] Knoop, J., Rüthing, O. and Steffen, B., "Partial dead code elimination", In Proc. *ACM SIGPLAN Conference on Programming Language Design and Implementation'94*, of *ACM SIGPLAN Notices*, Vol. 29, No. 6, pp. 147-158, Orlando, FL, June 1994.