추론적 부분 중복 제거의 최적화 예외 영역 문제 해결 알고리즘

신 현 덕[†] · 안 희 학[††]

요 약

본 논문에서는 Knoop 등이 2004년에 제안한 추론적 부분 중복 제거 알고리즘을 개선한다. 본 연구에서는 기존 추론적 부분 중복 제거에서 최적화가 적용되지 않는 영역이 발생될 수 있는 문제를 제기하고 이 문제에 대한 해법을 제안한다.

개선된 추론적 부분 중복 제거 알고리즘은 컴파일러의 프로필링 기법을 통해 얻어진 실행 빈도에 대한 정보를 통해 실행 속도 최적화를 수행하며 메모리 최적화도 수행한다.

키워드 : 추론적 부분 중복 제거, 최적화, 프로필링

An Algorithm of Solution for the Exceptional Field Problem in the Speculative Partial Redundancy Elimination(SPRE) Optimization

Hyun-Deok Shin[†] · Heui-Hak Ahn[††]

ABSTRACT

This paper improves the algorithms for Speculative Partial Redundancy Elimination(SPRE) proposed by Knoop et al. This research brings up an issue concerning a field to which SPRE cannot be applied, and suggests a solution to the problem.

The Improved SPRE algorithm performs the execution speed optimization based on the information on the execution frequency from profiling and the memory space optimization.

Key Words : SPRE, Optimization, Profiling

1. 서 론

추론 부분 중복 제거(SPRE : Speculative Partial Redundancy Elimination)는 부분 중복 제거[4-7]에 추론을 통한 부분 중복제거라는 새로운 접근법을 도입하여 수행하는 기법이다[2,3]. Bodik은 추론 부분 중복 제거의 이 문제에 대한 해법을 설명했으나 이 해법은 증명도 없고 실험 결과도 없다. 추론 부분 중복 제거는 Gupta, Bodik과 Soffa에 의해 연구되었고[1,3], 2004년에 Scholz, Horspool과 Knoop에 의해 메모리와 실행 속도 최적화를 위한 알고리즘[8]이 제안되었으나 이 알고리즘은 프로그램의 흐름 정보에 따라 최적화 예외 영역이 발생할 수 있다는 문제점을 갖고 있다.

본 논문에서 제안하는 개선된 추론 부분 중복 제거 기법은 네트워크로 구성된 제어 흐름 그래프를 실행 속도 최적화, 메모리 최적화, 실행 속도/메모리 최적화의 최적화 분야별로 분할하여 부분 중복 제거를 수행하고, 기존의 SPRE 기법에서 발생할 수 있는 최적화 예외 영역 문제를 제시하고 그 해결 방법을 제안한다.

2. 추론 부분 중복 제거의 이론적 배경

2.1 Correctness 정의

제어 흐름 그래프의 모든 노드 $u \in N$는 I_u로 표시하는 노드의 입구와 O_u로 표시하는 노드의 출구 부분으로 구분된다.

제안하는 알고리즘에 의해 입구와 출구를 네트워크로 구성하여 두 영역 *Available*과 $\neg$*Available*로 분할할 것이다. *Available*은 변수 t에 의해 e가 사용 가능하다는 것을 나타내며 $\neg$*Available*은 *Available*을 제외한 나머지 노드들이다.

SPRE에 의한 프로그램 변환은 정의 2-1을 만족해야 한다[8].

† 정 회 원 : 관동대학교 컴퓨터학과 겸임교수
†† 종신회원 : 관동대학교 컴퓨터학과 교수
논문접수 : 2006년 7월 2일, 심사완료 : 2006년 9월 8일

(정의 2-1) *Correctness*

$$\forall p = < u_1, \ldots, u_k > \in Path(s,f) : \forall i (1 \leq i \leq k) :$$

$$replace(u_i) \Rightarrow \; ((\exists j (1 \leq j \leq i) : store_N(u_j) : \forall k (j \leq k < i) :$$

$$COMP(u_k) \vee NULL(u_k))$$

$$\vee (\exists j (1 \leq j \leq i) : store_X(u_j) :$$

$$\forall k (j < k < i) : COMP(u_k) \vee NULL(u_k))$$

정의 2-1은 어떤 노드 u_i에 재배치가 수행되기 위해서는 노드 u_i 이전 노드 u_j에 $insert_N$이 수행되어야하고 u_j를 포함하는 이후 노드에서 u_i까지의 노드 u_k 중에 COMP 노드나 NULL 노드가 있거나, 노드 u_i 이전 노드 u_j에 $insert_X$가 수행되어야하고 u_j 이후 노드에서 u_i까지의 노드 u_k 중에 COMP 노드나 NULL 노드가 있어야 함을 의미한다.

2.2 SPRE 비용함수

SPRE에서 각 노드의 실행 비용을 계산하기 위한 비용함수는 다음과 같다.

$$f = \sum_{u} cst_u(i_u, o_u)(\alpha frq(u) + \beta spc(u))$$

frq(u)는 노드 u의 실행 빈도를 의미고 spc(u)는 메모리 공간 비용을 의미한다. α가 0인 경우는 메모리 공간을 위한 최적화를 의미하고 β가 0인 경우는 속도를 위한 최적화가 수행됨을 의미한다.

3. 네트워크 분할을 위한 알고리즘

3.1 네트워크 구성

본 논문에서 사용하는 제어 흐름 그래프의 각 노드는 식 e의 값을 수정하는 노드 MOD와 식 e를 계산하는 노드 COMP 그리고 식 e를 계산하지도 않고 값을 수정하지도 않는 노드 NULL의 3가지 경우로 구분한다.

알고리즘 3-1은 s-t 네트워크를 구성하는 알고리즘으로 제어 흐름 그래프를 입력받아 s-t 네트워크를 출력한다.

```
procedure construct_network()
 begin
    for i := 1 to |CFG_E_node| do
      begin  CFG_node = read(CFG);
        if (path(edge(CFG_node_i)) >= 2) then M_level_edge(edge(CFG_node_i));
        node_i.n = CFG_node.entry;        node_i.x = CFG_node.exit;        end;
    add_node(S);          add_node(T);   weight_edge(S, node_1.n) = ∞;
    for i := 1 to E_node do
      begin weight_edge(node_i.x, (succ(node_i)).n) = ∞;
            case modification : weight_edge(S, node_i.x) = node_i.frq;
            case computation : weight_edge(node_i.n, T) = node_i.frq;
            case null : weight_edge(node_i.n, node_i.x) = node_i.frq;
      end; end
```

[알고리즘 3-1] 네트워크 구성 알고리즘

```
function M_level_edge(m_edge)
begin
  for i := succ(node(m_edge(v))) to E_node do
   begin  if (nodei == PR_Code) then
            begin add_node(synthetic(m_edge));
              action_flag = HOISTABLE();
              if (action_flag) then
                    begin  INSERT();  delete(m_edge); end;
              else       return 0;   end;
          else      delete(m_edge);
  end; end
```

[알고리즘 3-2] 다중 레벨 가지 함수 알고리즘

알고리즘 3-2는 제어 흐름 그래프를 네트워크로 구성하기 전에 SPRE 최적화 예외 영역 문제를 일으킬 수 있는 가지의 존재 여부를 확인하여 다중 레벨 가지가 존재할 경우 이에 대한 처리를 수행하는 알고리즘이다. PR_Code는 부분 중복(partial redundancy) 코드를 나타낸다.

3.2 네트워크 분할

알고리즘 3-1에 의해 구성된 s-t 네트워크의 노드들의 가지 가중치를 이용하여 각 노드의 실행 비용을 계산하기 위한 비용 함수는 알고리즘 3-3과 같다.

알고리즘 3-4는 비용 함수의 결과를 이용하여 s-t 네트워크의 각 노드를 *Available*과 ¬*Available* 영역으로 분할한다. 이 알고리즘은 s-t 네트워크를 입력으로 받아들인 후 알고리즘 3-3을 이용하여 네트워크 가지의 가중치를 계산하여 min-cut 네트워크를 출력한다.

```
function cost_func(cut_edge_line)
 begin
    for i := 1 to |cut_edge_line| do
      begin  if (α> 0 && β> 0) then
        cost_value = cost_value + (cost(cut_edge_i) *
                    (frequency(cut_edge_i) + byte_instruction(exp)));
        else if (α== 0) then
        cost_value = cost_value + (cost(cut_edge_i) * byte_instruction(exp));
        else (β== 0) then
        cost_value = cost_value + (cost(cut_edge_i) * frequency(cut_edge_i));
 end; end
```

[알고리즘 3-3] 비용 함수 알고리즘

```
function partitioning()
 begin
    for i := 1 to E_node do net_node = read(s_t_network);
    while (net_node != S || net_node != T) do
      begin          s_t_node_i = net_node_j.n;        i++;
          s_t_node_i = net_node_j.x;     i++;  j++;  end;
    source_nodes = s_t_node_1;          Best_value = ∞;
    while (source_nodes != s_t_node) do
      begin          sink_nodes = s_t_node - source_nodes;
        cut_edge_line = s_t_cut(source_nodes, sink_nodes);
        Unit_value = cost_func(cut_edge_line);
        if (Unit_value < Best_value) then
          begin     min_cut_line = cut_edge_line;
                    source_node = s_t_node_next  end;
      end; end
```

[알고리즘 3-4] 네트워크 분할 알고리즘

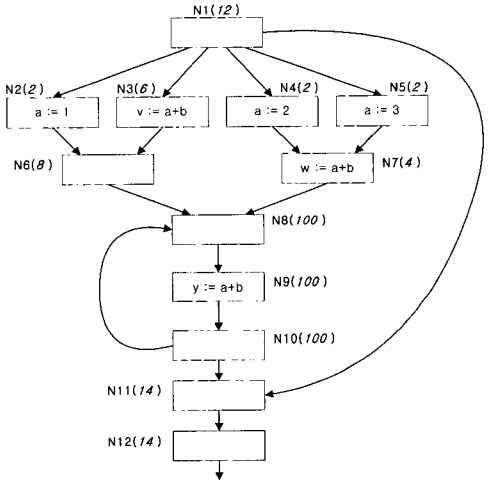
4. 최적화 예외 영역 문제와 해결방법

SPRE는 제어 흐름 그래프의 가지 정보를 이용하여 네트워크의 노드들을 연결한다. 이 경우 정의 2-1에 의해 제어 흐름 그래프의 모든 가지 (u,v)∈E에 대해 네트워크의 노드 O_u와 I_v사이의 가지에 무한대의 가중치가 부과된다. 그러나 경우에 따라 이 무한대의 가중치 부과가 SPRE 해법이 적용될 수 없는 영역을 만든다.

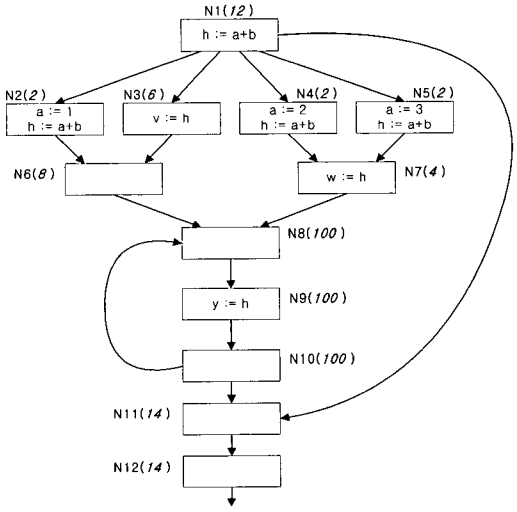
4.1 최적화 예외 영역과 해결방법1

4.1.1 최적화 예외 영역 1

(그림 4-1)의 제어 흐름 그래프는 식 a+b에 대한 3개의 계산과 110회의 수행 횟수를 갖는다. 이 제어 흐름 그래프가 네트워크로 구성될 경우 N1에서 N11의 가지는 무한대의 가중치를 갖게 되는데 이 가지의 범위 내에서는 SPRE 해법을 적용할 수 없다.



(그림 4-1) 제어 흐름 그래프 1



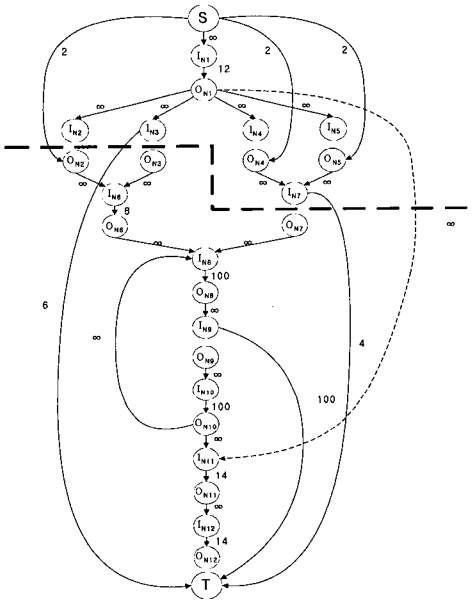
(그림 4-2) SPRE 적용 결과

(그림 4-1)은 하나의 min-cut만을 가질 수 있으며 SPRE 수행 결과는 (그림 4-2)와 같다. 이 흐름 그래프는 식 a+b에 대한 4개의 계산과 18회의 수행 횟수를 갖는다.

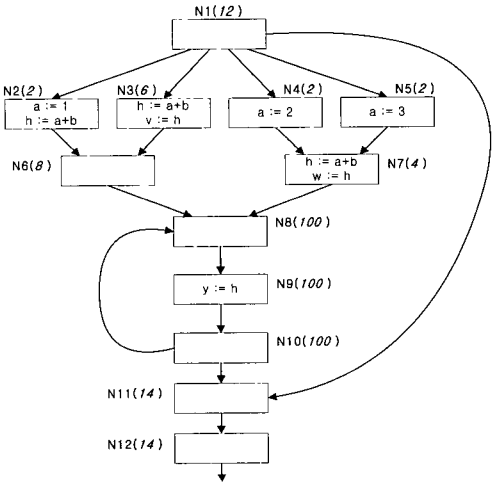
4.1.2 해결방법 1

(그림 4-1)의 N1에서 N11의 가지 범위 이후에 부분 중복 코드가 나타나지 않는다면 이 가지에 대한 가중치는 무시될 수 있다. 제안한 알고리즘에서는 이와 같은 범위의 가지 가중치를 무시할 수 있도록 구성하였다.

(그림 4-3)은 알고리즘 3-1과 알고리즘 3-2에 의해 구성된 min-cut 네트워크이다. 점선으로 나타낸 가지는 가중치가 무시된 가지를 의미한다. (그림 4-4)는 제안한 알고리즘을 적용한 최적화 결과다. 이 흐름 그래프는 식 a+b에 대한 3개의 계산과 12회의 수행 횟수를 갖는다.



(그림 4-3) 그림 4-1의 min-cut 네트워크



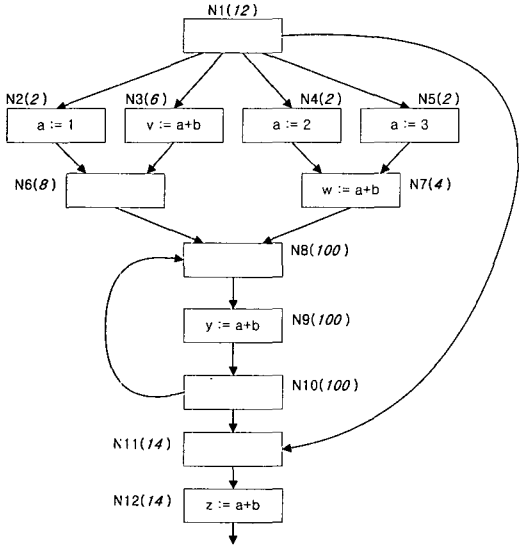
(그림 4-4) 제안한 알고리즘 적용결과

4.2 최적화 예외 영역과 해결방법 2

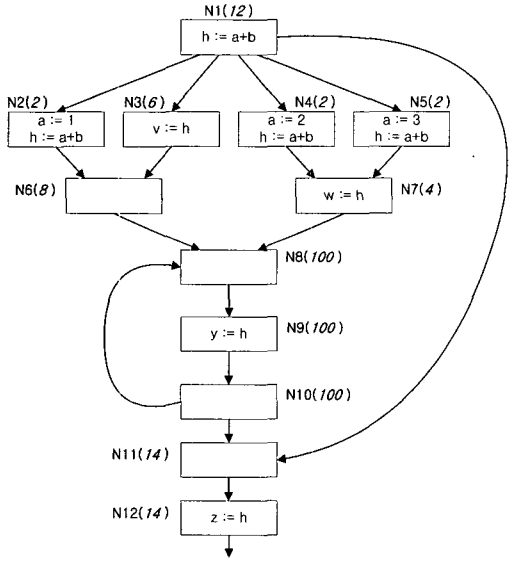
4.2.1 최적화 예외 영역 2

(그림 4-5)의 제어 흐름 그래프는 식 a+b에 대한 4개의 계산과 124회의 수행 횟수를 갖는다. 이 제어 흐름 그래프가 네트워크로 구성될 경우 N1에서 N11의 가지는 무한대의 가중치를 갖게 되는데 이 가지의 범위 내에서는 SPRE 해법을 적용할 수 없다. 또한, 이 가지의 범위 이후 노드 N12에 부분 중복 코드가 존재하기 때문에 이 가지는 무시될 수 없다.

(그림 4-5)는 두개의 min-cut을 가질 수 있지만 실행속도와 메모리, 실행속도/메모리 최적화의 결과는 모두 동일하다. SPRE 수행 결과는 (그림 4-6)과 같고 이 흐름 그래프는 식 a+b에 대한 4개의 계산과 18회의 수행 횟수를 갖는다.



(그림 4-5) 제어 흐름 그래프 2



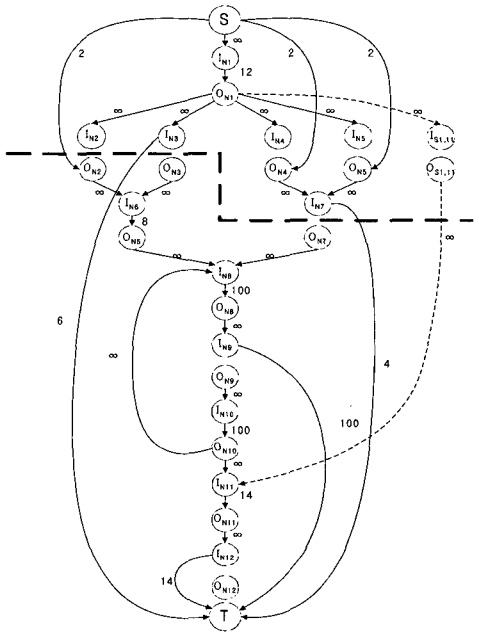
(그림 4-6) SPRE 적용 결과

4.2.2 해결방법 2

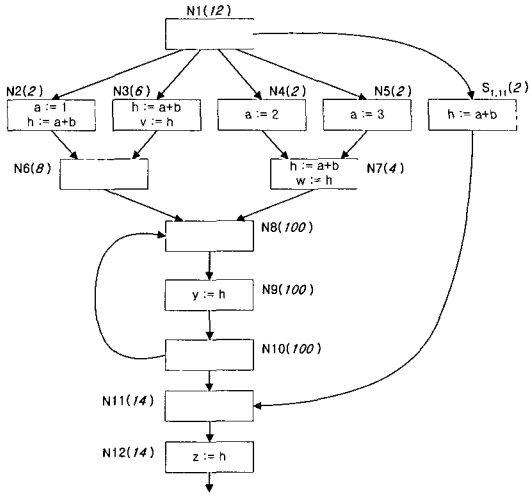
(그림 4-5)의 N1에서 N11의 가지는 네트워크에서 무한대의 가중치를 갖게 되는데 이 가지의 범위 이후 노드 N12에 부분 중복 코드가 존재하기 때문에 이 가지는 무시될 수 없다.

제안한 알고리즘에서는 이와 같은 범위 이후의 부분 중복 코드를 이 범위 안으로 끌어올리는 기법을 이용한다.

(그림 4-7)은 (그림 4-6)의 N1에서 N11의 가지에 합성 노드 S1,11을 삽입하여 N12의 식을 끌어올린 후 알고리즘 3-1과 알고리즘 3-2에 의해 구성된 min-cut 네트워크이다. (그림 4-7)에서 $O_{S1,11}$에서 I_{N11}의 가지에 대한 무한대 가중치는 무시될 수 있다. 점선으로 나타낸 가지는 가중치가 무시된 가지를 의미한다. (그림 4-8)은 제안한 알고리즘을 적용



(그림 4-7) 그림 4-6의 min-cut 네트워크



(그림 4-8) 제안한 알고리즘 적용 결과

한 최적화 결과다. 이 흐름 그래프는 식 a+b에 대한 4개의 계산과 14회의 수행 횟수를 갖는다.

5. 성능 분석

성능 분석을 위해 Windows 시스템에서 Visual Studio .NET 2003과 Visual C++ 6.0을 이용하였으며 요구 메모리 측정을 위해 개발한 프로그램을 이용했다.

5.1 네트워크 분할을 이용한 SPRE의 분야별 적용 분석

사용된 세 가지 유형의 프로그램은 본문에서 사용한 제어 흐름그래프의 원본 프로그램과 복잡한 반복문을 많이 포함하는 프로그램, 메모리 할당이 많이 일어나는 프로그램을 사용하였다.

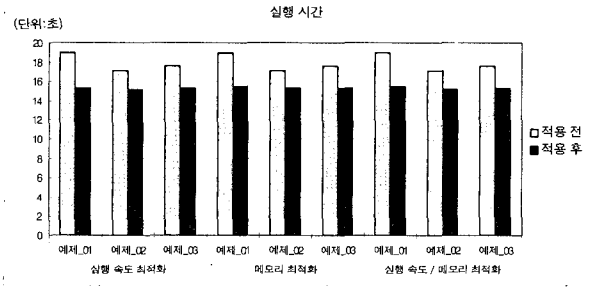
<표 5-1>에서 예제_01의 실행속도 최적화 경우, 알고리즘을 수행하기 전 3,644KB이던 요구 메모리가 48,672KB로 증가하였다. 속도만을 고려한 최적화는 메모리 요구가 크게 증가할 수 있기 때문에 메모리 감소에 대한 고려도 중요하다.

메모리 최적화를 수행한 경우는 상대적으로 실행시간에 대한 효율이 낮아 질 수 있다. 예제_01의 실행속도 최적화 비율은 0.887이지만 메모리 최적화 비율은 0.894로 높아졌다.

〈표 5-1〉 수행결과

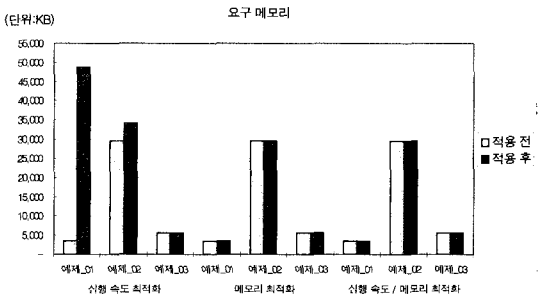
(단위 : 시간(초), 메모리(KB))

알고리즘 \ 예제			예제_01	예제_02	예제_03
실행 속도 최적화	실행 시간	적용 전	17.14	19.01	17.63
		적용 후	15.21	15.32	15.38
		비율	0.887	0.806	0.872
	요구 메모리	적용 전	3,644	29,524	5,576
		적용 후	48,672	34,316	5,576
		비율	13.357	1.162	1.000
메모리 최적화	실행 시간	적용 전	17.14	19.01	17.63
		적용 후	15.32	15.55	15.38
		비율	0.894	0.818	0.872
	요구 메모리	적용 전	3,644	29,524	5,576
		적용 후	3,644	29,524	5,576
		비율	1.000	1.000	1.000
실행 속도 /메모리 최적화	실행 시간	적용 전	17.14	19.01	17.63
		적용 후	15.21	15.55	15.38
		비율	0.887	0.818	0.872
	요구 메모리	적용 전	3,644	29,524	5,576
		적용 후	3,644	29,524	5,576
		비율	1.000	1.000	1.000



(그림 5-1) 실행 시간 분석 결과

실행속도/메모리 최적화의 경우 실행속도가 느릴 수 있지만 메모리 요구는 더 적다. 예제_02의 실행속도 최적화에서 실행시간은 15.32초이고 실행속도/메모리 최적화에서는 15.55초로 증가했지만 메모리 요구는 34,316KB에서 29,524KB로 낮아졌다.



(그림 5-2) 요구 메모리 분석 결과

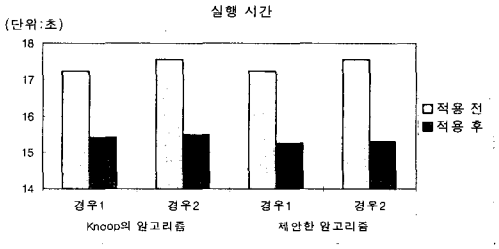
5.2 최적화 예외 영역 분석

<표 5-2>는 SPRE의 최적화 예외 영역 문제에 제안한 알고리즘을 적용한 결과를 나타낸다. 경우1의 경우에 Knoop의 SPRE 기법과 제안한 알고리즘의 실행 시간 비율은 0.894와 0.885로 큰 차이가 없지만, Knoop의 SPRE 기법의 요구 메모리 비율은 1.304로 최적화 전 프로그램보다 증가했다.

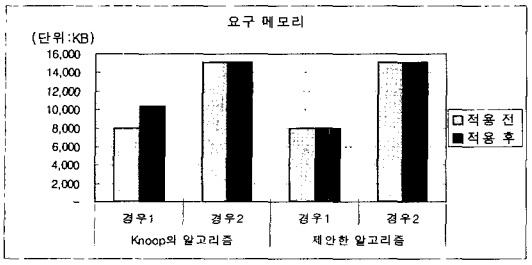
〈표 5-2〉 최적화 예외 영역 수행 결과

(단위 : 시간(초), 공간(KB))

알고리즘 \ 경우			경우1	경우2
Knoop의 알고리즘	실행 시간	적용 전	17.25	17.57
		적용 후	15.43	15.49
		비율	0.894	0.882
	요구 메모리	적용 전	7,948	15,148
		적용 후	10,364	15,148
		비율	1.304	1.000
제안한 알고리즘	실행 시간	적용 전	17.25	17.57
		적용 후	15.27	15.32
		비율	0.885	0.872
	요구 메모리	적용 전	7,948	15,148
		적용 후	7,948	15,148
		비율	1.000	1.000



(그림 5-3) 알고리즘 적용 결과 - 실행 시간



(그림 5-4) 알고리즘 적용 결과 - 요구 메모리

6. 결 론

본 논문에서는 임베디드 응용 프로그램의 최적화를 위해서 네트워크 분할 알고리즘을 이용한 추론 부분 중복 제거 알고리즘을 제안했다. 제안한 네트워크 분할 알고리즘은 네트워크로 구성된 제어 흐름 그래프를 실행 속도 최적화, 메모리 최적화, 실행 속도/메모리 최적화의 최적화 분야별로 분할하여 부분 중복 제거를 수행한다.

제안한 알고리즘의 첫 번째 목적은 프로그램 실행 시 요구되는 메모리의 감소이며 두 번째는 실행 시간을 감소시키는 것이다. 단지 프로그램의 실행 속도만을 고려하는 경우에는 메모리 요구가 크게 증가하기 때문에 메모리 감소에 대한 고려도 중요하다.

성능 분석을 해본 결과 실행속도/메모리 최적화의 경우 실행 속도 최적화에 비해 실행 속도가 느릴 수 있지만 메모리에 대한 요구는 더 적을 수 있다. 예제_02의 경우 실행 속도 최적화 적용 결과 15.32초인 실행 시간이 실행속도/메모리 최적화 적용에서는 15.55초로 증가했지만 메모리에 대한 요구는 34,316KB에서 29,524KB로 약 14% 낮아졌다. 이것은 프로그램을 실행하는데 요구되는 메모리의 크기가 실행 속도에 비해 더 중요한 임베디드 시스템에 적합한 최적화 기법이다.

본 논문에서는 네트워크 분할 알고리즘을 이용하여 Knoop 등이 제시한 SPRE 해법을 개선시키고 기존 SPRE의 최적화 예외 영역 문제를 제시하고 이에 대한 해결 방법을 제안했다. 제안한 기법을 이용하여 제어 흐름 그래프에 대한 SPRE 알고리즘 적용 결과를 각각 비교 분석하고 기존 SPRE 알고리즘과 제안한 알고리즘의 성능 분석 결과도 제시했다.

참 고 문 헌

[1] Bodik, R., Gupta, R. and Soffa, M. L., "Complete removal of redundant computations," *Proceed-ings of ACM Conference on Programming Language Design and Implementation*, Vol. 33, No. 5, pp. 1-14, New York, June 1998.

[2] Cai, Q. and Xue. J., "Optimal and efficient speculation-based partial redundancy elimination," *Proceedings of the 1st IEEE/ACM International Symposium on Code Generation and Optimization*, pp.91-102, 2003.

[3] Gupta, R., Berson, D. A. and Fang, J. Z., "Path profile guided partial redundancy elimination using speculation," *Proceedings of the 1998 International Conference on Computer Languages*, pp.230-239, 1998.

[4] Horspool, R. N. and Ho, H. C., "Partial redundancy elimination driven by a cost-benefit an-alysis," *Proceedings of 8th Israeli Conference on Computer Systems and Software Engineering*, pp. 111-118, June 1997.

[5] 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.

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

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

[8] Scholz, B., Horspool, N. and Knoop, J., "Optimizing for space and time usage with specula-tive partial redundancy elimination," *ACM SIGPLAN Notices, Proceedings of the 2004 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems*, Vol. 39, No. 7, pp. 221-230, June 2004.

신 현 덕

e-mail : ubhd@kd.ac.kr

1998년 관동대학교 전자계산공학과 (공학사)

2000년 관동대학교 대학원 전자계산공학과(공학석사)

2006년 관동대학교 대학원 전자계산공학과(공학박사)

관심분야 : 컴파일러, 병렬 컴파일러, 프로그래밍 언어, 코드 최적화

안 희 학

e-mail : hhahn@kd.ac.kr

1981년 숭실대학교 전자계산학과(공학사)

1983년 숭실대학교 대학원 전자계산학과 (공학석사)

1994년 숭실대학교 대학원 전자계산학과 (공학박사)

1994년~1996년 관동대학교 전자계산소 소장

2001년~2003년 관동대학교 전산정보원장

1984년~현재 관동대학교 공과대학 컴퓨터학과 교수

관심분야 : 컴파일러, 병렬컴파일러, 프로그래밍 언어, 함수언어, 오토마타 등