트리패턴매칭기법의 재목적 가능한 중간코드 최적화 시스템

김 정 숙† · 오 세 만††

요 약

ACK에서는 패턴 테이블 생성기와 펩홀 최적화기에서 스트링 패턴 매칭 기법을 이용하여 EM 중간 코드에 대한 최적화 코드를 생성한다. 하지만 이 스트링 패턴 매칭 방법은 패턴 선정 시에 반복적으로 많은 비교 동작이 이루어지므로 비효율적이다. 본 논문은 ACK의 중간 코드 최적화기를 개선하기 위해 EM 트리 생성기, 최적화 패턴 테이블 생성기, 트리 패턴 매칭기로 구성된 트리 패턴 매칭 알고리즘을 이용한 EM 중간코드 최적화 시스템을 설계하고 구현하였다. 이러한 트리 패턴 매칭 알고리즘은 EM 트리를 하향식으로 순회하면서 트리 구조를 가진 패턴 테이블을 참조하여 루트 노드를 중심으로 패턴 매칭을 수행한다. 트리 패턴 매칭 동작은 궁극적으로 ACK의 스트링 패턴 매칭에 비해 최적화 패턴을 찾는데 걸리는 시간을 평균 10.8% 감소시킬 수 있는 효과를 보였다.

Retargetable Intermediate Code Optimization System Using Tree Pattern Matching Techniques

Jung-Sook Kim† · Se-Man Oh††

ABSTRACT

ACK generates optimized code using the string pattern matching technique in pattern table generator and peephole optimizer. But string pattern matching method is not effective due to the many comparative actions in pattern selection. We designed and implemented the EM intermediate code optimizer using tree pattern matching algorithm composed of EM tree generator, optimization pattern table generator and tree pattern matcher. Tree pattern matching algorithm practices the pattern matching that centering around root node with refer to the pattern table, with traversing the EM tree by top-down method. As a result, compare to ACK string pattern matching methods, we found that the optimized code effected to pattern selection time, and contributed to improved the pattern selection time by about 10.8%.

1. 서 론

최근 다양한 고급 언어 및 성능이 향상된 목적 기계가 개발됨에 따라 언어 설계자의 관심은 프로그래밍 언어와 목적 기계가 분리된 형태의 컴파일러를 작성하는 것이 효율적이라는 인식이 확산되어 있다. 따라서 한 개의 목적 기계에 적합한 것 대신에 목적 기계의 다양성에 쉽게 적용할 수 있는 재목적 가능한 컴파일러 자동화 도구에 대한 연구가 진행되고 있다.

이러한 기반을 중심으로 설계되는 재목적 컴파일러의 모델 중 전단부는 정형화된 표현 방법으로 자동화 방법이 잘 정립된 반면 후단부 작성을 도와주는 도구는 최적화나 코드 생성에 대한 정형화의 어려움으로

† 종신회원 : 광운대학교 교수
†† 종신회원 : 동국대학교 컴퓨터공학과 교수
논문접수 : 1998년 12월 30일, 심사완료 : 1999년 7월 26일

인해 1970년대부터 연구되어 PQCC, ACK를 중심으로 발표되기 시작하였다.

더구나 CPU의 성능이 아무리 높아졌다 하더라도 프로그램에 대한 보다 빠른 실행과 최적의 코드는 여전히 중요하기 때문에 최적화기 시스템 구축은 반드시 필요하다. 원시 언어와 목적 기계에 독립적인 중간 코드에서의 최적화가 가장 적절한 위치가 될 수 있다[8].

목적 기계 독립적인 최적화의 선행 연구가 되는 ACK 최적화기에서는 코드 최적화 알고리즘으로 스트링 패턴 매칭 방법을 사용하고 있으나 지나친 반복 검색 동작을 수행하므로 패턴을 검색하는데 많은 시간이 걸린다. 따라서 본 연구에서는 보다 빠른 시간에 최적화 패턴을 검색할 수 있도록 트리 패턴 매칭 알고리즘을 고안하였다.

트리 패턴 매칭은 프로그래밍 과제의 많은 분야인, 비절차적인 프로그래밍 언어에 대한 인터프리터의 설계, 추상자료형의 자동적인 구현, 컴파일러의 코드 최적화, symbolic computation, 구조적 편집기에서의 문맥 검색, 자동화 이론 증명 등에서 결정적인 역할로 활용되어 왔다[15]. 본 연구에서는 최적화 패턴 검색을 위해 트리 패턴 매칭 알고리즘을 활용한다.

최적화 패턴 검색용의 트리 패턴 매칭 알고리즘을 활용하여 본 연구에서는 EM 트리 생성기, 최적화 패턴 테이블 생성기, 트리 패턴 매칭기로 구성된 중간 코드 최적화 시스템을 설계하고 구현하였다.

또한 본 연구에서 사용하는 중간코드인 EM(Experimental Machine)은 ACK에서 알골 형태(Algol-like)의 원시언어와 바이트 기계(byte machine) 형태의 목적기계에 적합한 EM-기계라는 추상적인 스택의 개념에 근거를 둔 기계 구조의 중간언어이다. 이 EM 코드는 의사명령어(pseudo code)와 EM 명령어로 구분되며 각 명령어는 상수, 명칭 등과 같은 피연산자를 가질 수 있다. 그리고 EM 명령어는 원시 언어의 기본 연산에 대응되며, 기본 연산 동작은 스택에 기반하여 이루어지며 각 명령어의 실행정보(+:push, -:pop, 0:no effect)를 가지고 있어서 트리의 구성이 용이하다.

EM 트리 생성기는 EM 스트림을 입력으로 받아 트리 패턴 매칭을 위해 EM 트리를 생성한다. 이때 EM 트리의 생성 알고리즘은 각 EM 명령어의 Fake 스택에 대한 운영 정보를 이용한다. 최적화 패턴 테이블 생성기는 645개의 최적화 패턴들을 트리 형태의 최적화 패턴 테이블로 재구성한다. EM 트리 생성기에서 생성된 EM 트리와 더불어 트리 패턴 매칭기에 의해 트리 패턴 매칭을 수행할 수 있도록 하기 위함이다. 트리 패턴 매칭기는 EM 트리를 중위운행법의 방식으로 운행하면서 최적화 패턴 테이블에 존재하는 하위 EM 트리와 매핑하고 이에 부합하는 패턴에 대해 EM 트리를 최적화된 EM 트리로 재구성한다.

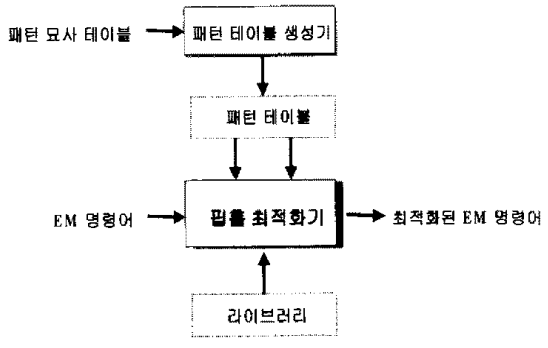
본 논문의 구성은 2장에서 본 연구의 기반이 되는 ACK 중간 코드 최적화 시스템의 구성과 ACK 최적화 기법에 대해 고찰한다. 제3장에서는 본 연구에서 설계하고 구현한 트리 패턴 매칭 기법을 이용한 EM 중간 코드 최적화 시스템과 트리 패턴 기법을 설명한다. 제4장에서는 트리 패턴 매칭 최적화 알고리즘의 성능을 평가하며 마지막으로 제5장에서는 결론과 향후 연구 방향에 대해 기술한다.

2. ACK에서의 중간코드 최적화

2.1 코드 최적화 시스템

전단부에서 생성된 EM 코드는 핍홀 최적화기에 입력되는데 여기서는 몇개의 명령어에 대한 윈도(window)를 가지고 입력을 검색하여 일련의 비효율적인 코드를 좀더 효율적인 코드로 대치하는 스트링 패턴 매칭 알고리즘을 사용한다.

ACK의 핍홀 최적화기는 크게 최적화 테이블, 파서, 패턴 테이블, 라이브러리, 최적화기 생성을 위해 테이블과 함께 컴파일되는 C 프로그램으로 구성되며 (그림 1)과 같은 구조를 가지고 있다.



(그림 1) ACK 핍홀 최적화기의 구성

패턴 묘사 테이블은 (패턴부:대치부)의 쌍으로 구성되는데 패턴부는 최적화기에 의해 인식될 수 있는 최적화되지 않은 EM 명령어의 열을 나타내며 대치부는 패턴에 대해 최적화된 EM 명령어로 변환되는 부분을

니다[14].

패턴 테이블 생성기는 패턴 묘사 테이블의 내용을 파싱하고 (패턴:대치) 쌍으로 구성된 입력을 받아들여 패턴 테이블을 출력한다. 패턴 테이블은 최적화기가 효율적으로 이용할 수 있도록 패턴 묘사 테이블을 바이트 배열(array of byte) 형태로 갖고 있는 테이블이다[14].

라이브러리는 핍홀 최적화기에서 이용되는 대부분의 자료 구조를 관리하며, 출력 큐, 패턴 큐, 보조 큐로 이루어져 있다.

핍홀 최적화기는 구성된 패턴 테이블을 참조하여 입력으로 들어오는 EM 명령어에 대해 스트링 패턴 매칭을 통해 최적화된 EM 명령어로 생성한다.

2.2 최적화 패턴

핍홀 최적화기에서 수행되는 최적화기의 성능은 패턴 내용에 크게 좌우되므로 패턴 작성은 매우 중요하며 최대한의 최적화 효과를 반영할 수 있도록 구성되어야 한다. 여기서 사용하는 최적화 패턴은 ACK에서 개발한 643개와 Kees Visser의 요청에 의해 추가된 2개의 패턴(lol asp $2>=w : asp $2-w, ldl asp $2>=2*w : asp $2-2*w)을 포함하여 모두 645개의 패턴을 사용한다[14].

2.3 ACK 최적화 알고리즘

ACK의 중간 코드 최적화기는 중간 코드에 대한 기본 블록에 대해 최적화 패턴으로 대치하는 핍홀 최적화 동작을 수행한다. 또한 중간 코드에 대응하는 최적의 패턴을 찾기 위해 스트링 패턴 매칭 기법을 적용하여 동등한 의미를 가지면서 좀더 효율적인 코드를 생성하는 방법으로 (알고리즘 1)과 같이 6단계의 과정을 거쳐 수행된다.

```
peep_process() {
    getline()          /* ① 원시코드를 라인 단위로 입력 */
    do {
        peephole_Match(pattern, peephole);
        relabel();
        flow();
    } while (madeopt && ++npasses<5000);
}
peephole_Match(pattern, peephole)
    Pattern_Descriptor pattern[];
    Queue peephole;
{
    basicblock()              /* ② hash value compute */
        for_each(p.pattern) {
            if(peephole.match_pattern(p) &&
                              /* ③ op_code check        */
            peephole.match_operand(p) &&
                              /* ④ operand check        */
            check_constraints(p);
                              /* ⑤ 제약조건 check       */
            peephole.tryrepl();
                              /* ⑥ 명령어 대체          */
                    }
    }
}
```

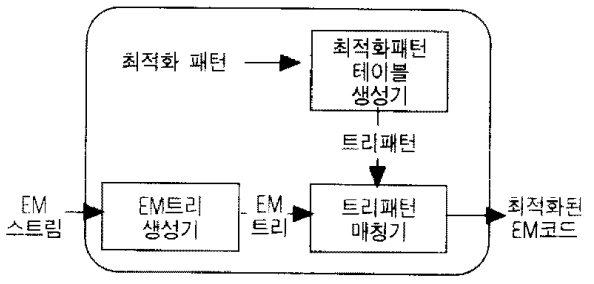
(알고리즘 1) 스트링 패턴 매칭 알고리즘

ACK 중간 코드 최적화는 최적화 패턴과 EM 입력 스트림을 입력으로 받아 스트링 패턴 매칭 방법으로 최적화를 수행한다. 스트링 패턴 매칭 방법은 패턴 결정 시에 반복적으로 많은 비교 동작이 이루어지므로 최적화 패턴을 찾는데 걸리는 시간이 많아 비효율적이다[1].

3. 트리 패턴 매칭 최적화 시스템

3.1 시스템 개요

본 논문의 트리 패턴 매칭 최적화 시스템은 (그림 2)와 같이 EM 트리 생성기, 최적화 패턴 테이블 생성기, 트리 패턴 매칭기로 구성되어 있다.



(그림 2) 트리 패턴 매칭 최적화 시스템

EM 트리 생성기는 EM 스트림을 입력으로 EM 트리를 생성한다. EM 트리는 각 EM 명령어와 Fake 스택에 대한 운영 정보를 이용하여 구성된다. 물론 피연산자와 연산자를 사전에 구분하지는 않는다.

최적화 패턴 테이블 생성기는 ACK에서 제공한 최적화 패턴들을 입력으로 받아 최적화 패턴 테이블로 재구성한다. 각 패턴은 트리 형태로 재구성되며 EM 트리 생성기에서 생성된 EM 트리와 더불어 트리 패턴 매칭기에 의해 트리 패턴 매칭을 수행할 수 있도록 하기 위함이다.

트리 패턴 매칭기는 EM 트리를 중위운행법 방식으로 운행하면서 최적화 패턴 테이블에 존재하는 하위 EM 트리와 매핑을 시도하여 이에 부합하는 패턴에 대하여 EM 트리를 최적화된 EM 트리로 재구성한다. 최적화된 EM 트리는 코드 변환기에 의해 최적화된 EM 코드로 출력된다.

3.3 EM 트리 생성기

EM 트리 생성기는 각 EM 명령어의 스택 운영 정보를 활용하여 EM 코드를 트리 패턴 매칭에 적합하도록 EM 트리로 변환한다.

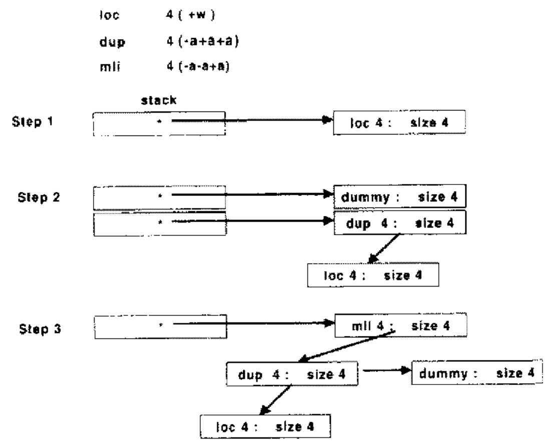
EM 트리 생성은 먼저 각 EM 명령어 중 연산 명령어에 대해서는 스택 운영 정보를 활용하여 (알고리즘 2)와 같이 구성한다.

```
process_mnem (emnode_ptr pnode)
  {
    while (action)
      fake 스택에 대한 push/pop/no effect 동작 결정.
      EM 명령어의 크기 설정.
    if (ACT_POP)
      크기 정보를 이용하여 자노드를 스택에서 얻음.
      만약, 먼저 pop된 자노드가 존재하면 자노드들 brother로.
    else if (ACT_PUSH)
      만약, 이미 push되었다면 보조노드를 push.
    fi
    elihw
      만약, ACT_PUSH가 없는 EM 명령어는 자노드를 자노드 포인터에 연결한 후 push. size는 0.
  }
```

(알고리즘 2) EM 트리 생성 알고리즘

그리고 EM 의사 명령어나 레이블의 처리는 스택에 존재하는 모든 노드를 제노드로 연결하고, 마지막 제노드로 해당 노드를 연결하여 구성한다. 이렇게 생성된 EM 트리는 중위운행법으로 운행시 EM 스트림 입력과 동일함을 유지한다.

(알고리즘 2)를 이용하여 구성한 EM 트리의 예는 (그림 3)과 같다.



(그림 3) EM 트리 구성

3.4 최적화 패턴 테이블 생성기

최적화 패턴 테이블 생성기는 최적화 패턴의 입력에 대해 EM 트리와 트리 패턴 매칭에 적합하도록 트리 패턴을 생성한다. 트리 패턴은 EM 트리에서 발견되는 패턴 부분과 패턴 부분에 대응되는 대치 부분으로 구성된다.

각 패턴의 구성 형식을 EBNF로 표시하면 다음과 같다.

{ EM 명령어 } + [expression] :
{ EM 명령어 [expression] }*

여기서 {EM 명령어} 부분은 EM 트리로 구성되며 [expression]은 결합 법칙과 우선 순위를 적용한 조건식 트리로 구성된다.

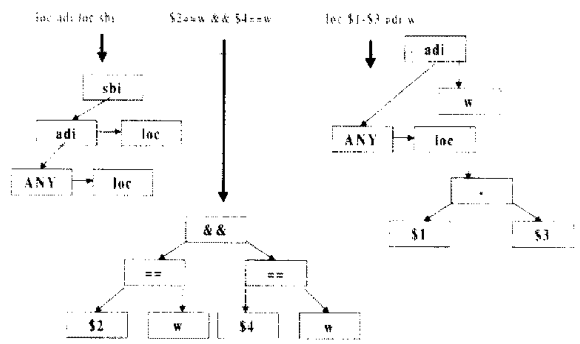
패턴에 대한 트리 생성은 EM 트리 생성 과정에서 fake 스택에 대한 단어 크기 정보를 이용하지 않고 연산자와 피연산자의 관계를 이용한다는 차이점을 제외하고는 EM 트리와 동일한 과정을 거쳐 구성된다.

최적화 패턴 트리 구성은 피연산자와 연산자 관계를 이용하여 하나의 최적화 패턴에 대해 트리 패턴 매칭을 위한 원시 EM 코드에 대한 트리, 조건식에 대한 트리, 대치부에 대한 트리 등 해당하는 3개의 EM 트리로 구성된다.

(그림 4)는 다음 loc adi loc sbi $2==w && $4==w : loc $1-$3 adi w 최적화 패턴 트리의 실제적인 구성 과정 예이다.

패턴부의 원시 EM 코드에 대한 트리와 대치부의 패턴 EM 트리에는 각각 ANY가 포함되어 있다. 이 ANY는 임의의 EM 연산 명령어와 매핑될 수 있는데,

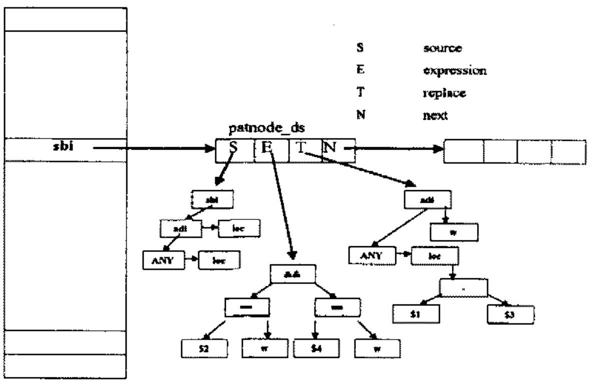
패턴 트리의 구성 과정에서 피연산자가 존재하지 않을 경우에 ANY를 추가하여 구성한 것이다. 조건식에 대한 트리의 구성은 내부함수나 연산자가 트리의 부모노드가 되고 각 인수가 자노드가 되도록 조건식에 대한 패턴 트리를 구성한다. 대치 부분에 대한 EM 트리는 expr에 대한 포인터를 개별적으로 가지고 있고 전체적인 최적화 패턴 테이블의 구조를 살펴보면, 테이블은 패턴부의 원시 EM 코드 부분의 EM 명령어에 대한 입력 순서가 아니라 구성된 패턴 EM 트리의 부모를 기준으로 테이블이 구성된다.



(그림 4) 최적화 패턴 트리의 구성

최적화 패턴 테이블은 총 EM 연산 명령어의 개수에 레이블, LLP(lol, ldl), LEP(loe, lde), SLP(stl, sdl), SEP(ste, sde), CBO(adi, adu, and, ior, xor)의 수를 더한 크기 만큼의 배열로 구성되어 있고 EM 트리의 opcode를 직접적으로 해당 테이블의 인덱스로 이용할 수 있도록 하였다.

최적화 패턴 테이블 구성에 대해 sbi를 기준으로 테이블에 대한 등록 사례는 (그림 5)와 같다.



(그림 5) 최적화 패턴 트리의 테이블 구성

3.5 트리 패턴 매칭기

입력된 EM 스트림에 대한 EM 트리와 최적화 트리 패턴이 구성된 후 트리 패턴 매칭을 시작한다. 트리 패턴 매칭은 EM 트리를 중위운행법으로 운행하면서 최적화 패턴 테이블에서 매칭되는 하위 EM 트리가 존재하는지의 여부를 확인하고 존재할 경우 조건식의 값이 참인지를 확인한다. 이런 조건을 만족한 경우에는 최적화된 트리로 재구성한다. 트리 패턴 매칭을 위한 알고리즘은 (알고리즘 3)과 같다.

```
match_treepattern(emnode_ptrppar,emnode_ptr
pnode, bool flag)
/*
 * ppar   parent emnode 포인터
 * pnode  매칭할 emnode 포인터
 * flag   pnode가 ppar의 자노드/세노드 여부
 */
{
  if (pnode != NULL)
    if (매칭되는 하위 tree 존재)
      트리를 재구성.
      pnode를 해당 위치로 변경.
      match_treepattern(pnode, pnode->son, true);
      match_treepattern(pnode,pnode->brother, false);
    fi
}
```

(알고리즘 3) 트리 패턴 매칭 알고리즘

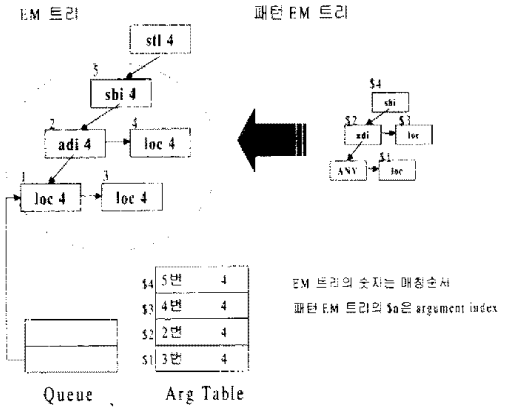
트리 패턴 매칭 알고리즘에 대한 실제적인 패턴 매칭 사례는 다음 4단계를 통해 확인할 수 있다.

먼저 제 1단계에서는 (그림 6)과 같이 EM 스트림 (loc 4, loc 4, adi 4, loc 4, sbi 4, stl 4)에 대한 EM 트리에 대해 매칭되는 최적화 패턴(loc adi loc sbi $2==w && $4==w : loc $1-$3 adi w)의 패턴 EM 트리가 존재하는지를 검색한다. 이 과정에서 EM 트리와 최적화 패턴 EM 트리에 대해 같은 깊이로 중위운행법으로 운행하면서 최적화 패턴 EM 트리의 각 노드와 매칭되는 EM 트리의 인수값은 임시 배열인 Arg Table에 저장해 두었다가 다음 단계에서 조건식을 계산할 때 활용한다.

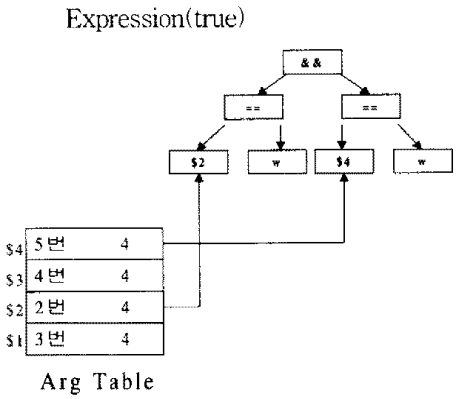
그리고 패턴 EM 트리에서의 ANY는 제 3단계의 대치 부분이나 제 4단계의 트리 재구성 과정에서 실제 값을 읽어야 할 노드이므로 내용되는 값을 큐에 포인터로 저장해 둔다.

제 2단계에서는 (그림 7)에서와 같이 매칭된 패턴에

대해 조건식을 체크한다. 이 과정에서 (그림 6)에서 Arg Table에 저장해 두었던 인수 값을 활용한다.

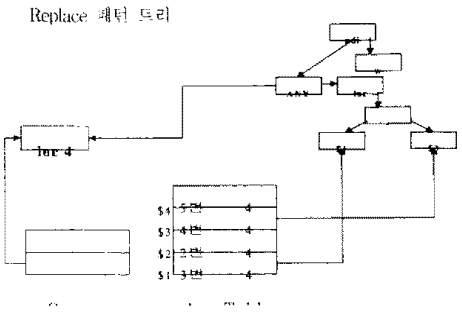


(그림 6) 트리 패턴 매칭 제 1단계 (패턴 트리 검색)



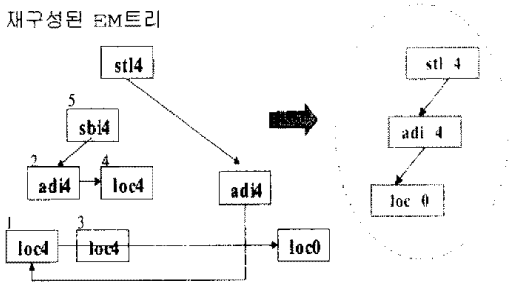
(그림 7) 트리 패턴 매칭 제 2단계 (조건식 체크)

제 3단계에서는 (그림 8)에서와 같이 1, 2단계에서 확인된 매칭의 결과를 토대로 의미적으로 동등하면서 훨씬 효율적인 명령어로 입력 EM 스트림을 대치한다. 이 과정에서 제 1단계의 큐에 저장해 두었던 포인터를 참조하여 ANY에 해당하는 노드를 찾아내어 표현한다.



(그림 8) 트리 패턴 매칭 제 3단계 (대치)

마지막 제 4단계에서는 (그림 9)에서와 같이 입력 EM 스트림의 트리와 트리로 재구성된 최적화 트리 패턴이 트리 패턴 매칭으로 최적화되어 최적화된 EM 트리로 재구성된 EM 트리를 출력으로 얻는다.

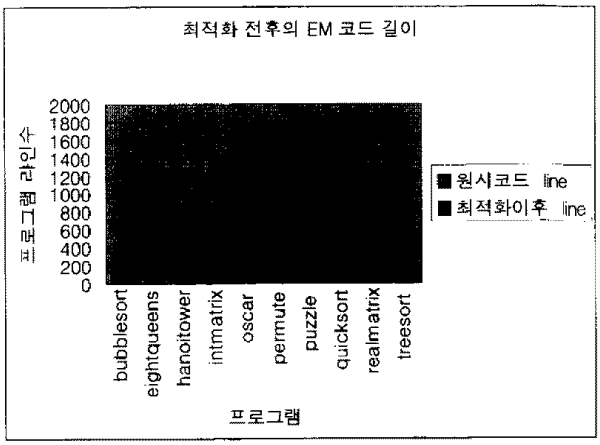


(그림 9) 트리 패턴 매칭 제 4단계(최적화 트리 구성)

4. 성능평가 및 분석

본 절에서는 본 논문에서 제안한 트리 패턴 매칭 알고리즘을 적용한 중간 코드 최적화 시스템의 성능을 평가하기 위하여 ACK에서 사용한 스트링 패턴 매칭 알고리즘과 패턴 매칭 시간을 비교 측정하였다. 성능 측정을 위한 실험 환경은 SUN 워크스테이션의 SUN OS 4.1.3 환경에서 역시 스텐포드 벤치마크의 10개 프로그램을 사용하였다.

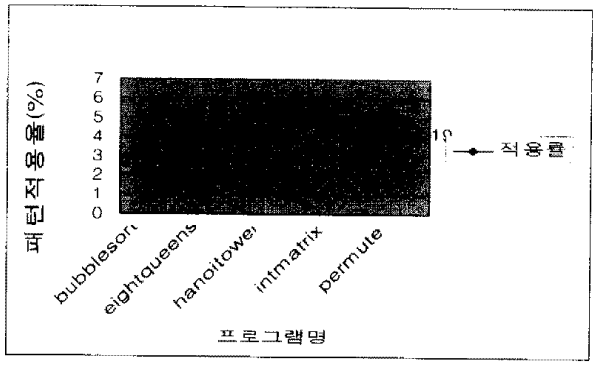
본 논문에서 구현한 최적화기에 스텐포드 벤치마크의 10개 프로그램을 수행시켜 실험한 결과, 원시 EM 코드에 대해 최적화 이후의 EM 코드 라인 수로 비교한 최적화율은 (그림 10)과 같다.



(그림 10) 최적화 전·후의 코드 길이 비교

그리고 최적화를 위해 실제로 사용된 패턴의 적용율은 (그림 11)과 같다. 실험에 적용한 패턴은 본 논문에

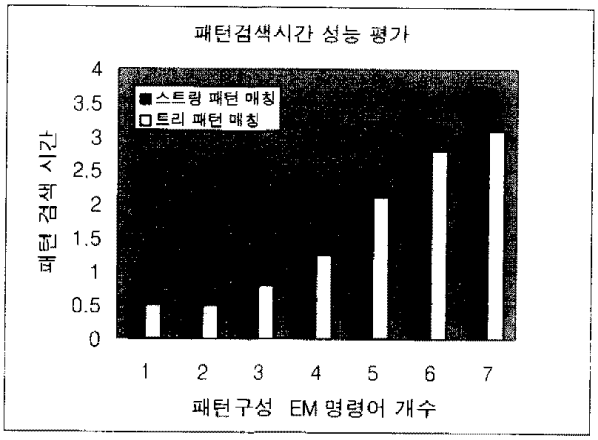
서 사용한 ACK 제공의 최적화 패턴 643개와 Kees Visser가 추가한 2개의 패턴을 포함한 645개의 최적화 패턴에 대한 적용율이다.



(그림 11) 최적화 패턴 적용율

최적화된 코드의 질은 최적화 패턴의 추가와 더불어 높아질 것이기 때문에 최적화율을 높이기 위한 패턴의 추가 및 EM 코드의 특성을 고려한 패턴을 보완할 필요가 있다고 생각한다. 또한 최적화율의 향상은 실행 시간과 기억장소 활용율 개선에도 상당히 긍정적인 영향을 미칠 것이기 때문에 다양한 최적화 내용을 반영할 수 있는 지속적인 패턴 추가에 대한 연구가 요구된다.

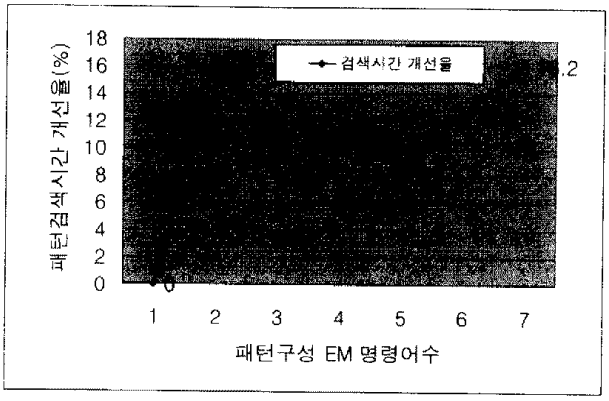
두 번째의 성능 평가로 본 논문에서 생성한 트리 형태의 최적화 패턴 테이블을 적용하여 매칭될 패턴에 대한 패턴 검색 시간을 비교한 결과는 (그림 12)와 같다.



(그림 12) 패턴 검색 시간 성능 평가

본 논문에 적용한 최적화 패턴에서 패턴을 구성하는 명령어의 개수는 2개에서부터 최대 7개 까지로 구성되어 있다. 물론 대부분이 3개 정도이며 실질적으로 5개 이상의 명령어를 가진 패턴이 적용되는 확률은 상당히 낮았다.

스트링 패턴 매칭에 대한 트리 패턴 매칭 알고리즘 적용시 패턴 검색 시간을 상대 비교한 결과는 (그림 13)과 같으며 평균 10.8%의 패턴 검색 시간 개선으로 나타났다.



(그림 13) 패턴 검색 시간 상대 비교

적용된 패턴의 검색 시간 비교는 ACK의 스트링 패턴 매칭 시간에 대한 트리 패턴 매칭 시간을 상대 평가하는 방식을 취하였다. 또한 패턴 매칭 시간을 보다 명확하게 확인하기 위해 실질적인 패턴 검색 시간에 대한 배율을 정하여 비교하였다. 패턴 검색 시간의 비교에 있어서 트리 패턴 매칭을 위해 수행되는 EM 트리의 구성 시간은 진단부의 출력으로 EM 트리를 생성할 수 있기 때문에 제외하였으며, EM 트리 구성 시에 소비되는 메모리 양은 전체적으로 스트링 패턴 매칭 기법에 비해 많이 사용되지만 본 논문에서는 이를 고려하지 않고 있다.

5. 결 론

ACK에서는 패턴 테이블 생성기와 핍홀 최적화기에서 스트링 패턴 매칭 기법을 이용하여 EM 중간 코드에 대한 최적화 코드를 생성한다. 패턴 테이블 생성기는 패턴 묘사 테이블의 내용을 파싱하고 (패턴:대치)쌍으로 구성된 입력을 받아들여 패턴 테이블을 출력한다. 패턴 테이블은 최적화기가 효율적으로 이용할 수 있도록 패턴 묘사 테이블을 바이트 배열 형태로 갖고 있는 테이블이다. 핍홀 최적화기에서는 구성된 패턴 테이블을 참조하여 입력으로 들어오는 EM 명령어에 대해 스트링 패턴 매칭 방법을 통해 최적화된 EM 명

명어로 생성한다. 하지만 이 스트링 패턴 매칭 방법은 패턴 길정시에 반복적으로 많은 비교 동작이 이루어지므로 비효율적이다. 본 논문은 ACK의 중간 코드 최적화기를 개선하기 위해 EM 트리 생성기, 최적화 패턴 테이블 생성기, 트리 패턴 매칭기로 구성된 트리 패턴 매칭 알고리즘을 이용한 EM 중간 코드 최적화 시스템을 설계하고 구현하였다.

EM 트리 생성기는 EM 스트림을 입력으로 EM 트리를 생성한다. EM 트리는 각 EM 명령어의 Fake 스택에 대한 운영 정보를 이용하여 구성된다. 물론 피연산자와 연산자를 사전에 구분하지는 않는다. 최적화 패턴 테이블 생성기는 최적화 패턴들을 최적화 패턴 테이블로 재구성한다. 각 패턴은 트리 형태로 재구성되며 EM 트리 생성기에서 생성된 EM 트리와 더불어 트리 패턴 매칭기에 의해 트리 패턴 매칭을 수행할 수 있도록 하기 위함이다. 트리 패턴 매칭기는 EM 트리를 중위운행법 방식으로 운행하며 최적화 패턴 테이블에 존재하는 하위 EM 트리와 매핑하고 이에 부합하는 패턴에 대해 EM 트리를 최적화된 EM 트리로 재구성한다.

본 논문에서 고안한 트리 패턴 매칭 알고리즘은 EM 트리를 하향식으로 순회하면서 트리 구조를 가진 패턴 테이블을 참조하여 루트 노드를 중심으로 패턴 매칭을 수행한다. 패턴 매칭 동작은 정적적으로 ACK의 스트링 패턴 매칭에 비해 최적화 패턴을 찾는데 걸리는 시간을 감소시킬 수 있는 효과를 가진다.

앞으로 보다 양질의 최적화된 코드를 생성하기 위해 최적화 패턴 테이블에 보다 효율적인 최적화 기법이 반영된 최적화 패턴을 개발하여 추가할 예정이다. 또한 목적기계 종속적인 코드의 최적화를 위해 목적 기계 의존적인 코드와 관련된 패턴 테이블의 개발 및 구동 루틴을 추가할 예정이다.

참 고 문 헌

[1] B. J. Mckenzie, "Fast Peephole Optimization techniques," Software Practice and Experience, Vol.19, No.2, pp.1151-1162, Dec., 1989.

[2] H. E. Bal, Vrije Universitcit Wiskundig Seinarium, Amsterdan, "The ACK Target Optimizer," 1989.

[3] Andrew S. Tanenbaum, Hans van Staveren, E.G.Keizer, Johan W. Stevenson, Mathematics Dept. Vrije Universiteit Amsterdan, The Netherlands. "A Practical Tool Kit for Making Portable Compilers," 1989.

[4] Jack W. Davidson and B. Whalley, "Quick Compilers Using Peephole Optimization," Software Practice and Experience., Vol.19, No.1, pp. 79-97, Jan., 1989.

[5] Jack W. Davidson and Christopher W. Fraser, "Automatic Generation of Peephole Optimizations," Preceeding of the ACM SIGPLAN '84 Symposiim on Compiler Construction SIGPLAN Notice, Vol.19, No.6, pp.111-116, June, 1984.

[6] Andrew S. Tanenbaum, Hans van Staveren and Johan W.Stevenson, "A Practical Tool Kit for Making Portable Compilers," CACM, Vol.26, No.9, Sep., 1983.

[7] Christoph M.Hoffmann and Michael. O'Donnell, "Pattern Matching in Trees," Journal of the Association for Computing Machinery, Vol.29, No.1, pp.68-95, Jan., 1982.

[8] Andrew S. Tanenbaum, Hans van Staveren and Johan W.Stevenson, "Using Peephole Optimization on Intermediate Code," ACM Trans. on Prog. and Systems, Vol.4, No.1, pp.21-36, Jan., 1982.

[9] William, A. Wulf, "An Overview of the Production Quality Compiler Compiler Project," IEEE Computer, Vol.13, No.8, pp.38-49, Aug., 1980.

[10] W. M. Mckeeman, "Peephole Optimization," CACM, Vol.8, No.7, pp.443-444, July, 1965.

[11] 김정숙, 오세만, "트리 패턴 매칭을 이용한 EM 코드 최적화 알고리즘", '96 추계학술발표대회논문집, 5권, 2호, pp.734-737, 한국정보처리학회, 1996. 10.

[12] 황순명, 김정숙, 오세만, "지연 슬롯을 제거하기 위한 SPARC 코드 최적화에 관한 연구", '96 춘계학술발표대회논문 집, 5권 1호, pp.129-132, 한국정보처리회, 1996. 4.

[13] 고광만, 김정숙, 오세만, "재목적 코드 생성에 관한 연구", '94 가을 학술발표논문집, 21권 2호, pp. 81-84, 한국정보과학회, 1994. 10.

[14] Has Van Staveren, Dept. of Mathematics and Computer Science Vrije Universitcit Amsterdam, The Netherlands, "Internal Documentation on the peephole optimizer from the Amsterdam Compiler Kit," 1991.

[15] CHRISTOPH M. HOFFMANN AND MICHAEL J. O'DONNELL, purdue University, west Lafavette, Indiana, "Pattern Matching in Trees," Journal of the Association for Computing Machinery, Vol.29, No.1, January 1982, pp.68-95.

김 정 숙

e-mail : jskim@elim.net

1984년 광운대학교 전산학과 졸업(학사)

1988년 동국대학교 교육대학원 전산학과 졸업(석사)

1999년 동국대학교 대학원 컴퓨터공학과 졸업(박사)

1995년~1999년 한국방송통신대학교 연구원

1999년 현재 광운대학교 강사

관심분야 : 프로그래밍 언어론, 컴파일러 구성론, 멀티미디어임

오 세 만

e-mail : semanoh@cakra.dongguk.ac.kr

1977년 서울대학교 사범대학 수학과 졸업(학사)

1979년 한국과학기술원 전산학과 졸업(석사)

1985년 한국과학기술원 전산학과 졸업(박사)

1985년~1988년 동국대학교 전산학과 조교수

1988년~1989년 미국 USL 대학 교환교수

1994년~현재 동국대학교 컴퓨터공학과 교수

관심분야 : 프로그래밍 언어론, 컴파일러 구성론임