자료 종속성 제거 방법을 이용한 프로시저 변환

장 유 숙† · 박 두 순††

요 약

기존의 순차 프로그램에서 병렬성을 추출하는 연구들은 하나의 프로시저 내 변환에 치중되고 있다. 그러나 대부분의 프로그램들은 프로시저 간 잠재된 병렬성을 가지고 있다. 본 논문에서는 자료 종속성 제거방법을 이용하여 프로시저 호출을 가진 루프에서 병렬성 추출 방식을 제안한다. 프로시저 호출을 포함하는 루프의 병렬화는 대부분 자료종속거리가 uniform 형태의 코드에서만 연구되었다. 본 논문에서는 자료종속거리가 uniform 코드와 nonuniform 코드에 대해 모두 적용 가능한 프로시저 간 변환 방법을 제시하였으며, 제시된 알고리즘의 성능평가를 위하여 CRAY T3E에서 성능평가 하였고, 제시된 방법이 효과적임을 보였다.

The Procedure Transformation using Data Dependency Elimination Methods

Yusug Chang† · Doo Soon Park††

ABSTRACT

Most researches of transforming sequential programs into parallel programs have been based on the loop structure transformation method. However, most programs have implicit interprocedure parallelism. This paper suggests a way of extracting parallelism from the loops with procedure calls using the data dependency elimination method. Most parallelization of the loop with procedure calls have been conducted for extracting parallelism from the uniform code. In this paper, we propose interprocedural transformation, which can be apply to both uniform and nonuniform code. We show the examples of uniform, nonuniform, and complex code parallelization. We then evaluated the performance of the various transformation methods using the CRAY-T3E system. The comparison results show that the proposed algorithm out performs other conventional methods.

키워드 : 재구성 컴파일러(reconstruction compiler), 자료 종속성(data dependence), 프로시저 변환(procedure transformation)

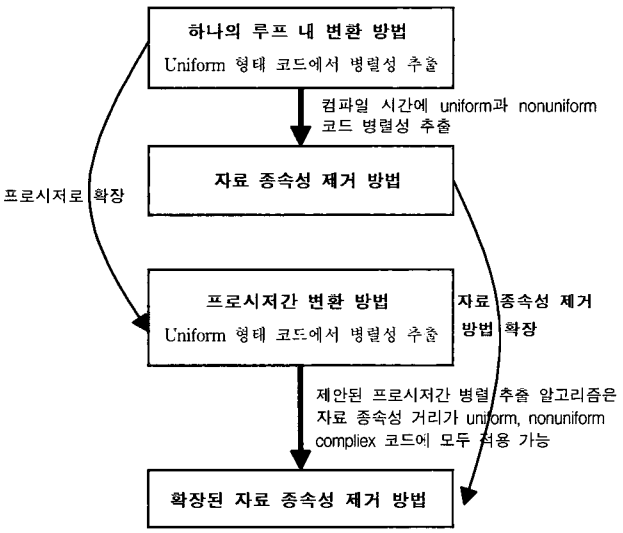
1. 서 론

현대는 정보화 사회이고, 정보화 사회에서는 수많은 정보들이 있다. 이렇게 많은 정보들을 빠른 시간안에 처리하기 위한 방법으로 병렬처리 시스템과 같은 더 빠른 컴퓨터를 만드는 노력들이 있었고, 최근에 상용화된 병렬처리 시스템들이 등장하였다.

그러나 더 많은 프로세서를 붙인다고 더 빠른 컴퓨터가 되는 것은 아니다. 병렬 컴퓨터의 능력은 그것을 탐지하기 위한 소프트웨어의 능력 없이는 불가능하다. 이러한 요구에 맞추어서 병렬화를 탐지하는 기술과 병렬 코드를 생성하는 기술들이 발전되어 왔다. 병렬화를 탐지하고, 병렬 코드를 생성하는 방법들은 주로 하나의 루프에서 연구되었다. 이러한 경우를 루프 변환이라 한다. 루프 변환 방법들은 자료종속성 거리가 uniform인 경우를 처리하는 방법과 nonuniform인 경우를 처리할 수 있는 방법으로 나누어 볼 수 있다. 자료종속성 거리가 uniform인 경우는 tiling[15], interchanging[3], skewing[3], unimodular[2], selective cycle shrinking[8], Hollander[8], Chen&Wang[4]등이 있으며, 자료종속성 거리가 nonuniform인 경우는 DCH[13]방법 그리고 IDCH[12]방법 등이 있다. 이러한 기존의 방법들은 전부 자료종속성을 분석해서 분할한 다음 스케줄링하는 방법들을 택하고 있으며 이 방법들로는 병렬성 추출에 한계를 가지고 있다. 따라서 본 논문에서는 자료종속성 거리가 uniform과 nonuniform에 모두 적용 가능한 것으로서 문장 수행에 의하여 매우 효율적으로 자료종속성을 제거하는 자료 종속성 제거 알고리즘[16]을 이용하여, 하나의 루프가 아닌 루프에 프로시저 호출을 포함하는 경우로 확장한다. 프로시저 간 변환 방법들도 루프 변환 방법들과 마찬가지로 자료종속성 거리가 uniform인 경우를 처리하는 방법과 nonuniform인 경우를 처리할 수 있는 방법으로 나누어 볼 수 있다. 자

† 준 회 원 : 순천향대학교 대학원 전산학과
†† 정 회 원 : 순천향대학교 정보기술공학부 교수
논문접수 : 2001년 4월 27일, 심사완료 : 2001년 11월 7일

료종속성 거리가 uniform인 경우는 loop extraction[7], loop embedding[6] 그리고 procedure cloning[11] 방법 등이 있으며, 자료종속성 거리가 nonuniform인 경우는 아직 제시되지 않고 있다. 본 논문에서는 자료 종속성 제거방법을 이용하여 프로시저 호출을 가진 루프에서 병렬성 추출 방법을 제안한다. 본 논문에서 제시한 방법은 자료 종속성 거리가 uniform인 경우와 nonuniform인 경우에 모두 적용이 가능하다. 제안된 프로시저간 변환 방법은 (그림 1.1)과 같다.



(그림 1.1) 프로시저 간 변환 방법

본 논문의 구성은 2장에서 프로시저 분석과 변환을 서술하였고 3장에서는 확장된 자료종속성 제거 방법을 이용한 병렬성 추출 알고리즘을 제시하였다. 4장에서는 성능분석을 하고 5장에서 결론을 서술한다.

2. 프로시저 분석과 변환

프로시저 분석은 프로시저 내 분석과 프로시저 간 분석으로 구분하고 프로시저 변환도 프로시저 내 변환과 프로시저 간 변환으로 구분한다[9, 14].

2.1 프로시저 분석

프로시저 내 분석이란 한 프로시저 내에서의 자료 흐름이나 제어 흐름에 관한 정보를 분석하는 것을 말한다. 프로시저 간 분석은 프로시저 사이의 호출/피 호출 관계에 나타나는 자료의 흐름에 관한 분석을 말한다. 프로시저 간 분석을 하기 위해서는 먼저 각 프로시저 사이의 호출/피 호출 관계를 나타내는 호출 그래프를 작성하고 이를 이용하여 프로시저간 자료의 흐름을 분석하게 된다. 각 프로시저 사이에 자료가 전달되는 경로는 형식 인자와 실 인자사이의 바인딩과 전역 변수에 의해 이루어지므로 인자들 사이의 자료 흐름을 나타낼 수 있는 바인딩 그래프를 사용하여 바인딩과 전역 변수에 의한 자료의 흐름을 분석한다. 그리고 이명 관계, 상수 전달, 참조 정보 등을 분석한다[1, 2, 4, 5].

2.2 프로시저 변환

프로시저 내에서 사용되는 변환 방법으로 루프 분리, 루프 융합, 루프 반전, 루프 왜곡, 루프 교환, 루프 타일링 등의 변환 방법이 있다[8, 11, 13]. 프로시저 내 변환 방법 중 자료종속성 제거 방법은 행렬의 계산을 이용하여 자료종속성을 계산하고 프로그램을 수행하는 기법에 의해 자료종속성을 제거하는 방법이다[16].

프로시저 간 변환 방법은 프로시저 사이에 적용되는 변환 방법으로 inlining 확장 변환 방법은 피 호출 프로시저의 문장들을 프로시저 정보에 의하여 프로시저 호출 위치에 모두 대체한다. Loop extraction변환 방법은 루프에서 호출을 가진 프로시저에서 피 호출 프로시저의 루프를 프로시저의 호출 위치 외부로 이동하는 방법이다. Loop embedding은 프로시저 호출을 포함하는 루프 헤더를 피 호출 프로시저로 이동하는 변환 방법이다. Procedure cloning은 프로시저가 여러 차례 호출될 때 프로시저를 최적화 한 프로시저로 복사하여 많은 프로시저가 호출하게 하는 변환 방법 등이 있다[3, 6, 7, 12].

3. 확장된 자료종속성 제거 방법을 이용한 병렬성 추출 알고리즘

프로시저를 포함한 코드의 병렬화와 병렬성 증가를 위한 최적화 방법으로 inlining후 최적화 방법과 최적화 후 inlining방법이 있다. Inlining후 최적화 방법은 호출 프로시저의 호출 위치들을 피 호출 프로시저의 코드로 대체하는 inlining을 수행한 후 inline된 코드를 프로시저 내 변환 방법을 사용하여 최적화하는 방법이다. 최적화 후 inlining방법은 각 프로시저 단위로 프로시저 내 변환 방법으로 최적화한 후 inlining을 제외한 프로시저 간 변환 방법으로 최적화한다. 본 논문에서는 프로시저를 포함한 코드의 병렬화와 병렬성 증가를 위해 호출 프로시저의 개수와 호출 매개변수 값에 따라 최적화 후 inlining 방법과 inlining후 최적화 방법을 제안한다. 최적화 방법으로는 하나의 프로시저 내 변환 방법인 자료종속성 제거 방법을 확장하여 프로시저 간 변환 방법으로 사용한다.

본 논문에서 제시한 알고리즘은 다음과 같다.

1. 호출 멀티그래프 작성
2. 확장된 호출 그래프(augmented call graph)로 확장
3. 프로시저간 정보 계산
4. 종속성 분석
5. IF (하나의 프로시저가 한번 이상 호출되고 호출 매개 변수 값이 변하지 않으면)
   THEN goto (알고리즘 3)

```
   ELSE goto (알고리즘 4)
6. 중간 코드에 자료 종속성 제거 알고리즘 (알고리즘 2) 대입 병
   렬 코드 생성
```

(알고리즘 1)

```
1. 초기화 과정
   S = 문장 수
   동적 기억장소 배열 DMA, DMB, DMC선언
   sw = 0
2. diophantine방식 계산 위해 GCD계산 함수 호출하여 pass = 1인
   S * S 크기의 2차원 종속 행렬 DMB구함
   IF (인덱스 변수가 같으면) THEN rename
3. 중첩 루프의 인덱스 변수를 i, j라하고 최대치 N1, N2인 경우
   Forall DMB행렬의 i, j에 대하여
   IF ((aij ⊕ uij) > N1 || (cij ⊕ wij) > N2)
   THEN 그 원소는 0으로 치환
   IF sw == 0 THEN DMA = DMB, sw = 1
4. DMB 행렬 이용하여 0이 아닌 모든 원소에 대해
   Forall i, j에 대하여 IF (uij & wij == 1)
   THEN uniform
   ELSE IF (uij & wij != 1) THEN nonuniform
        ELSE complex형태의 doall Si 문장으로 변환
5. IF 똑같은 원소가 존재한다면 THEN 하나만 남겨 놓고 제거
6. IF 모든 원소 = 0 THEN go to 8
   ELSE pass 증가하기 위해 행렬의 곱 이용하여
        S * S^(pass+1) 크기의 종속행렬
        DMC = DMA * DMB 계산
        pass++
        DMB = DMC
        free DMC
   ENDIF
7. Go to 3
8. 변형된 문장을 제외한 모든 문장에 대해 doall Si 문장으로 변환
```

(알고리즘 2)

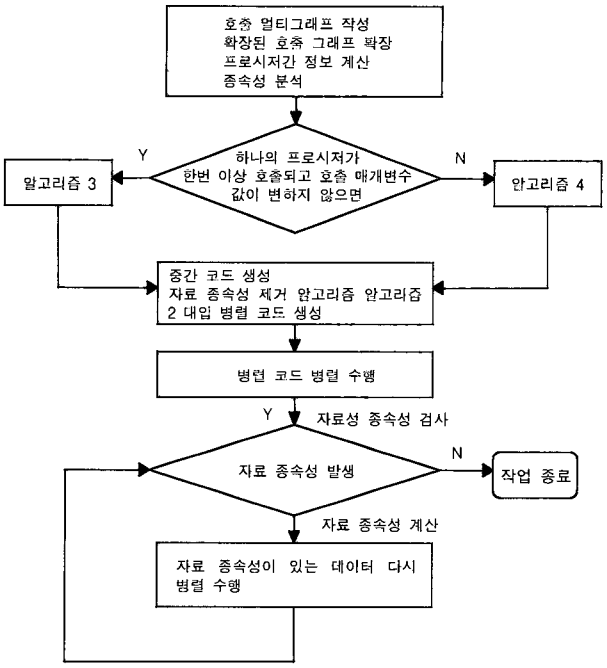
```
/* 프로시저 변환 적용 후 inlining 적용 */
{
  1. repeat(각 프로시저가 최적화될 때까지)
     { /* 프로시저 내 변환 적용 */ }
  2. repeat(프로시저 간 최적화될 때까지)
     { /* inlining을 제외한 프로시저 간 변환 적용 */ }
  3. reverse topological order로 inlining 적용
}
```

(알고리즘 3)

```
/* inlining 후 프로시저 내 변환 적용 */
{
  1. reverse topological order로 inlining 적용
  2. repeat(각 프로시저가 최적화될 때까지)
     { /* 프로시저 내 변환 적용 */ }
}
```

(알고리즘 4)

확장된 자료 종속성 제거 알고리즘의 순서도는 (그림 3.1)과 같다.



(그림 3.1) 확장된 자료 종속성 제거 알고리즘 순서도

4. 성능 평가

성능평가를 위하여 자료 종속성 거리가 uniform, nonuniform, complex형태에 대해서 기존의 다른 방법에서 사용한 두 개의 예제를 가지고 수행하였다. 첫 번째는 프로시저 호출을 가진 루프에서 사용되는 loop extraction변환 방법, loop embedding변환 방법 그리고 inlining변환 방법들을 비교 분석한다. 두 번째는 프로시저 호출을 가진 루프에서 사용되는 변환 방법 중 첫 번째 성능 평가에서 가장 좋은 성능을 가진 방법을 가지고 확장된 자료 종속성을 적용한 방법과 loop extraction변환 방법, loop embedding변환 방법 그리고 procedure cloning변환 방법 등과 비교 분석한다. 비교 분석을 위해서 CRAY T3E 시스템에서 수행하였다.

4.1 기존 방법들의 성능 평가

프로시저간의 변환 방법의 성능 평가하기 위하여 예제 코드를 uniform인 경우에는 하나의 루프 안에서 코드 변환 시 가장 많이 사용한 예를 선택 변형하였다. Nonuniform과 complex는 제시된 예제가 없어서 [16]에서 제시한 예제를 선택 변형하였다.

(그림 4.1)은 자료 종속성 거리가 일정한 uniform, 자료 종속성 거리가 일정치 않은 nonuniform, 자료 종속성 거리가 uniform과 nonuniform이 혼재된 complex의 예제 코드를 나타낸다.

```
Procedure P
real a(n1, n2), b(n1, n2), c(n1, n2)
integer i, j
```

```
do i = 1, n1
  do j = 1, n2
    call Q
  enddo
enddo

Procedure Q
real a(n1, n2), b(n1, n2), c(n1, n2)
integer i, j
  a(i, j-7) = b(i-1, j-3) + c(i-9, j+5)
  b(i+2, j+8) = a(i-5, j+4) + c(i+6, j-4)
  c(i+1, j-6) = a(i-6, j+3) + b(i-2, j+3)
```

(a) uniform

```
Procedure P
real a(n1, n2), b(n1, n2), c(n1, n2)
integer i, j
 do i = 1, n1
  do j = 1, n2
    call Q
  enddo
 enddo

Procedure Q
real a(n1, n2), b(n1, n2), c(n1, n2)
integer i, j
  a(4i, 5j) = b(5i, 2j) + c(4i, 6j)
  b(6i, 4j) = a(3i, 2j) + b(4i, 3j) + c(5i, 3j)
  c(7i, 5j) = a(4i, 3j) + b(5i, 3j)
```

(b) nonuniform

```
Procedure P
real a(n1, n2), b(n1, n2), c(n1, n2)
integer i, j
 do i = 1, n1
 do j = 1, n2
 call Q
 enddo
 enddo

Procedure Q
real a(n1, n2), b(n1, n2), c(n1, n2)
integer i, j
 a[i-2][j] = b[4*i][3*j] ;
 b[5*i][4*j] = a[3*i+3][j-4] + b[i-4][3*j] + c[3*i][4*j] ;
 c[7*i][5*j] = A[i-5][i-3] ;
```

(c) complex

(그림 4.1) 예제 코드 I

4.1.1 프로시저간의 변환 방법의 성능 평가

프로시저간 변환 방법인 inline substitution과 loop extraction, loop embedding변환 방법을 (그림 4.1) 예제 코드를 사용하여 성능평가 하였다. (그림 4.2)는 (그림 4.1)의 예제 코드(a)의 결과 코드를 보여준다.

```
Procedure P
real a(n1, n2), b(n1, n2), c(n1, n2)
integer i, j
do i = 1, n1
  do j = 1, n2
    a(i, j-7) = b(i-1, j-3) + c(i-9, j+5)
    b(i+2, j+8) = a(i-5, j+4) + c(i+6, j-4)
    c(i+1, j-6) = a(i-6, j+3) + b(i-2, j+3)
  enddo
enddo
```

(a) inline substitution

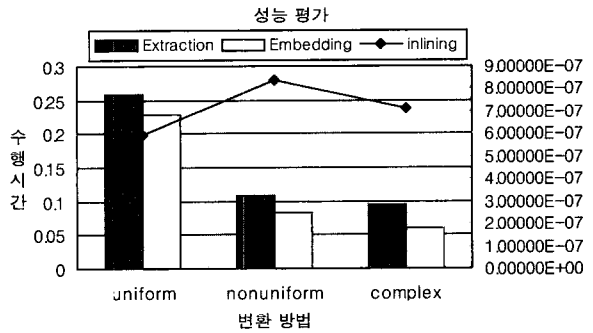
```
Procedure P
real a(n1, n2), b(n1, n2), c(n1, n2)
integer i, j
do i = 1, n1
  do j = 1, n2
    call Q
  enddo
enddo

Procedure Q
real a(n1,n2), b(n1,n2), c(n1,n2)
integer i, j
  a(i, j-7) = b(i-1, j-3) + c(i-9, j+5)
  b(i+2, j+8) = a(i-5, j+4) + c(i+6, j-4)
  c(i+1, j-6) = a(i-6, j+3) + b(i-2, j+3)
```

(b) Loop Extraction

```
Procedure P
real a(n1, n2), b(n1, n2), c(n1, n2)
  call Q

Procedure Q
real a(n1,n2), b(n1,n2), c(n1,n2)
integer i, j
 do i = 1, n1
   do j = 1, n2
     a(i, j-7) = b(i-1, j-3) + c(i-9, j+5)
     b(i+2, j+8) = a(i-5, j+4) + c(i+6, j-4)
     c(i+1, j-6) = a(i-6, j+3) + b(i-2, j+3)
   enddo
 enddo
```

(c) Loop Embedding

(그림 4.2) uniform형태 코드



(그림 4.3) 프로시저간 변환 방법의 성능평가

변환된 코드를 성능 평가한 결과는 (그림 4.3)과 같이 이중 차트를 이용하여 표현하였다. 세 가지 방법 모두 inline substitution변환 방법이 가장 효율적이었고 loop extraction을 수행한 코드가 가장 비효율적임을 알 수 있다.

4.2 확장된 자료 종속성 제거 방법

자료종속성 거리가 uniform, nonuniform 그리고 complex에 대해서 성능 평가를 위하여 기존의 프로시저 간 변환 방법에서 사용된 예제 코드[7, 11]를 이용하여 자료 종속성 제거 방법을 이용한 프로시저 변환 알고리즘과 loop extraction변환 방법, loop embedding변환 방법 그리고 procedure cloning 변환 방법과 비교 분석한다.

4.2.1 확장된 자료 종속성 제거방법을 이용한 병렬코드

확장된 자료 종속성 제거방법을 이용한 프로시저 변환 방법을 적용하여 병렬코드로 변환 후 병렬로 수행한다. 그리고 자료 종속성이 발생하지 않을 때까지 자료 종속성 검사하여 자료 종속성이 발생하면 자료 종속성 계산으로 자료 종속성이 있는 데이터를 찾아 다시 병렬 수행한다. (그림 4.4) (a)의 예제 코드를 프로시저를 포함한 코드의 최적화 알고리즘을 적용하여 병렬코드로 변환한 코드는 (그림 4.5)와 같다. 변환된 병렬 코드를 병렬로 수행하고, 수행된 병렬 코드에서 자료 종속성이 발생하는 곳에는 잘못된 결과 값을 가지게 되므로 자료 종속성 검사를 하여 자료 종속성이 발생하면 자료 종속성이 발생하는 문장만 다시 병렬 수행한다.

```
SUBROUTINE P
real a(n, n)
integer i
do i = 1, 10
  call Q(a, i)
  call Q(a, i+1)
enddo

SUBROUTIN Q(f, i)
real f(n, n)
integer i, j
do j = 1, 100
  f(i, j) = f(i, j) + ···
enddo
```

(a) uniform

```
SUBROUTINE P
real a(n, n)
integer i
do i = 1, 10
  call Q(a, i*3)
  call Q(a, i*5)
enddo

SUBROUTIN Q(f, i)
real f(n, n)
integer i, j
do j = 1, 100
  f(i, j*5) = f(i, j*4) + ···
enddo
```

(b) nonuniform

```
SUBROUTINE P
real a(n, n)
integer i
do i = 1, 10
  call Q(a, i)
  call Q(a, i*5)
enddo

SUBROUTIN Q(f, i)
real f(n, n)
integer i, j
do j = 1, 100
  f(i, j) = f(i, j) + ···
enddo
```

(c) complex

(그림 4.4) 예제 코드 II

```
SUBROUTINE P
real a(n, n)
integer i, j
parallel do i = 1, N1
 parallel do j = 1, N2
     f(i, j) = f(i, j) + ···
     f(i+1, j) = f(i+1, j) + ···
 end parallel do
end parallel do

parallel do i = 2, N1
 parallel do j = 1, N2
     f(i, j) = f(i, j) + ···
     end parallel do
end parallel do
```

(그림 4.5) 확장된 자료 종속성 제거 알고리즘

(그림 4.5)의 병렬 코드의 자료 종속성은 두 번째 문장의 f(i+1, j)와 첫 번째 문장의 f(i, j)에서 발생한다. 자료 종속성이 발생하는 문장을 병렬로 수행한다. 자료 종속성이 더 이상 발생하지 않으므로 (그림 4.4) (a) 예제코드의 최적화 알고리즘은 병렬 수행을 두 번 수행한다.

4.2.2 Loop extraction변환 방법을 이용한 병렬코드

프로시저 간 변환 방법인 loop extraction은 루프에서 호출을 가진 프로시저에서 피 호출 프로시저의 루프를 프로시저의 호출 위치 외부로 이동하는 방법이다. (그림 4.4) (a) 예제 코드에 loop extraction을 적용한 코드가 (그림 4.6) (a) 코드이고 효율적인 병렬성 추출을 위해서 loop fusion

```
SUBROUTINE P
real a(n, n)
integer i, j

do i = 1, N1
  do j = 1, N2
     call Q(a, i, j)
  enddo
  do j = 1, N2
     call Q(a, i+1, j)
  enddo
enddo

SUBROUTIN Q(f, i, j)
real f(n, n)
integer i, j

  f(i, j) = f(i, j) + ···
```

(a) Loop extraction

```
SUBROUTINE P
real a(n, n)
integer i, j

parallel do j = 1, N2
  do i = 1, N1
     call Q(a, i, j)
     call Q(a, i+1, j)
  enddo
end parallel do

SUBROUTIN Q(f, i, j)
real f(n, n)
integer i, j

     f(i, j) = f(i, j) + ···
```

(b) Fusion, Interchang 후 병렬화

(그림 4.6) Loop extraction

변환 방법과 loop interchange변환 방법을 적용한 코드는 (그림 4.6) (b)과 같다. (그림 4.6) (b)코드에서 호출 프로시저 P는 외부 루프 j는 병렬로 수행하고 내부 루프 i는 순차적으로 피 호출 프로시저 Q들을 호출한다.

4.2.3 Loop embedding변환 방법을 이용한 병렬코드

프로시저 호출을 포함하는 루프 헤더를 피 호출 프로시저로 이동하는 변환 방법인 loop embedding을 (그림 4.4) (a)의 예제 코드에 적용한 코드는 (그림 4.7) (a)이고, 효율적인 병렬성 추출을 위해 loop interchange변환 방법을 적용한 결과 코드는 (그림 4.7) (b)이다.

변환된 병렬코드는 (그림 4.7)과 같이 호출 프로시저 P는 두개의 피 호출 프로시저 Q를 호출하고, 피 호출 프로시저 Q는 외부 루프 j는 병렬로 수행하고 내부 루프 i는 순차적으로 실행한다.

```
SUBROUTINE P
real a(n, n)
integer i
 call Q(a, i)
 call Q(a, i+1)

SUBROUTIN Q(f, i)
real f(n, n)
integer i, j
do i = 1, N1
 do j = 1, N2
 f(i, j) = f(i, j) + ...
 enddo
enddo
```

(a) Loop embedding

```
SUBROUTINE P
real a(n, n)
integer i
 call Q(a, i)
 call Q(a, i+1)

SUBROUTIN Q(f, i)
real f(n, n)
integer i, j
parallel do j = 1, N2
 do i = 1, N1
 f(i, j) = f(i, j) + ...
 enddo
end parallel do
```

(b) Loop interchang 후 병렬화

(그림 4.7) Loop embedding

4.2.4 Procedure cloning변환 방법을 이용한 병렬코드

임의 프로시저가 여러 차례 호출될 때 프로시저를 최적화한 프로시저로 복사하여 많은 프로시저가 호출하게 하는 변환 방법이다. (그림 4.4) (a)의 예제 코드를 procedure cloning을 적용한 코드는 (그림 4.8)과 같이 피 호출 프로시저 Q에서 병렬 가능한 문장과 불가능한 문장을 분리하여 병렬 가능한 문장을 Qclone 프로시저로 복사한다. 그리고 호출 프로시저 P의 병렬 루프 j 내부에서 병렬 가능한 문장과 불가능한 문장을 따로 호출하여 병렬성을 증가시킨다.

```
SUBROUTINE P
real a(n, n)
integer i, j

parallel do j = 1, N2
      call Qclone(a, i, j)
      call Q(a, i+1, j)
end parallel do

SUBROUTIN Q(f, i, j)
real f(n, n)
integer i, j

 do i = 1, N1
     f(i, j) = f(i, j) + ···
 enddo
```

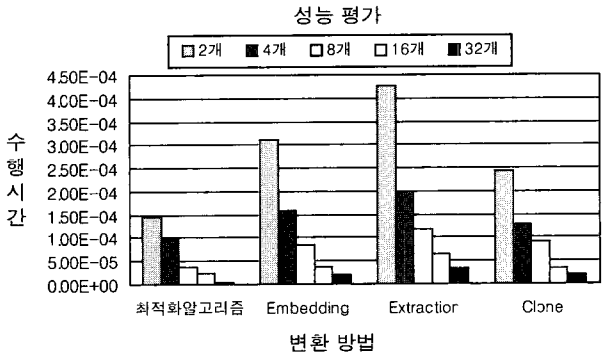
```
SUBROUTIN Qclone(f, i, j)
real f(n, n)
integer i, j

 parallel do i = 1, N1
      f(i, j) = f(i, j) + ···
 end parallel do
```

(그림 4.8) Procedure cloning

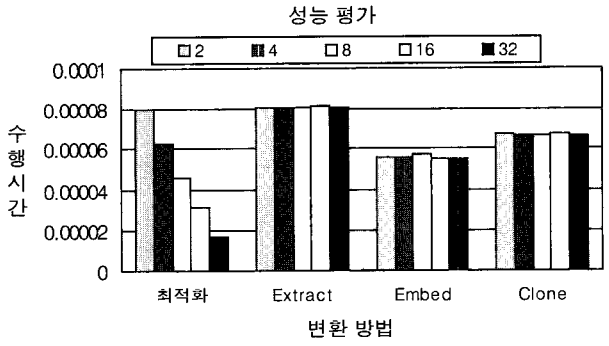
4.2.5 성능평가

데이터 수는 N_1은 *20*, N_2는 *100*개로 하고 프로세서 수를 *2, 4, 8, 16, 32*개로 증가하였다.

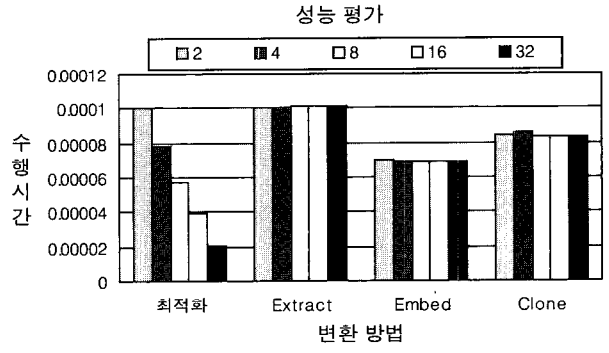
확장된 자료 종속성 제거 방법을 이용한 병렬성 추출 알고리즘은 inlining 확장 후 자료 종속성 제거 알고리즘을 이용하여 병렬로 코드를 변환 후 자료 종속성이 없을 때까지 병렬로 수행한다. Loop extraction과 loop embedding이 같은 조건일 때, loop embedding을 선택하는 것이 프로시저



(a) uniform



(b) nonuniform



(c) complex

(그림 4.9) 프로시저를 포함한 병렬 코드의 성능평가

호출 오버헤드가 감소된다. Procedure cloning은 순차처리 부분과 병렬처리 부분의 프로시저를 따로 작성 후 병렬화를 최대화시킨다. 자료 종속 거리가 nonuniform과 complex인 경우 확장된 자료 종속성 제거 방법을 이용한 병렬성 추출 알고리즘만 병렬화가 가능하여 병렬 처리하고, loop extraction변환 방법, loop embedding변환 방법 그리고 procedure cloning변환 방법은 병렬화가 어렵기 때문에 순차처리한다.

성능 평가 결과는 (그림 4.9)와 같이 자료 종속성 제거 방법을 이용한 최적화 알고리즘은 자료종속거리 uniform, nonuniform 그리고 complex 모두 프로세서 증가에 비례하여 성능이 좋아지고, loop embedding, loop extraction 그리고 procedure변환방법 중에서 uniform인 경우 procedure cloning 변환 방법이 우수하고, nonuniform과 complex 코드인 경우에는 본 논문에서 제시한 방법이 가장 우수함을 알 수 있다.

5. 결 론

병렬처리에서 루프가 전체 프로그램 실행 시간의 대부분을 차지하기 때문에 하나의 프로시저에서 루프 변환 방법에 대한 기법들이 많이 제안되었다. 그러나 프로시저간이나 루프 내부에 프로시저를 포함하는 코드들에 대한 병렬처리 연구는 많지 않다.

본 논문에서는 프로시저간이나 루프 내부에 프로시저를 포함하는 코드들에 대한 병렬처리를 위한 효율적인 알고리즘을 제시하였다.

성능평가는 자료종속성 거리가 uniform과 nonuniform 그리고 complex에 대해 자료 종속성 제거 방법을 이용한 프로시저 변환 방법과 loop extraction, loop embedding 그리고 procedure cloning변환 방법을 CRAY T3E 시스템에서 실행해서 비교 분석하였다. 자료종속 거리가 uniform과 nonuniform 그리고 complex 코드에 대해 모두 본 논문에서 제안한 방법이 가장 효율적이었고 loop extraction을 수행한 코드가 가장 비효율적임을 알 수 있었다. 프로세서 수 증가에 따른 병렬 코드의 성능평가는 프로세서 수가 증가하면서 비례적으로 성능이 좋아짐을 알 수 있었다.

본 논문에서 제안한 자료 종속성 제거 방법을 이용한 프로시저 변환 방법에 대해 성능 평가를 일반적인 benchmark 프로그램에 적용하여 일반화된 알고리즘임을 보여주는 것을 향후 과제로 남겨둔다.

참 고 문 헌

[1] Dale Allan Schouten, "*An overview of Interprocedural analysis Techniques for High Performance Parallelizing Compilers*," MS thesis, University of Illinois at Urbana-Champaign, 1986.

[2] D. Callahan, K.D. Cooper, K. Kennedy, and L. Torczon, "*Interprocedural constant propagation*," Journal of the ACM, pp.152 161, 1986.

[3] K. D. Cooper, M. W, Hall, L. Torczon, "*An experiment with inline substitution*," Technical Report Tr90-128, Dept. of computer Science, Rice University, 1990.

[4] Keith D. Cooper, Ken Kennedy, and Linda Torczon. "*Interprocedural constant propagation*," Technical Report TR-85-29, Department of Computer Science, Rice University, 1985.

[5] V. A. Guarna. "*A technique for analyzing pointer and structure references in parallel restructuring compilers*," In Proceedings of ICPP 88, Vol.2, Penn State Press, August, 1988.

[6] M. W. Hall, "*Managing Interprocedural Optimization*," PhD thesis, Dept. of computer Science, Rice University, 1991.

[7] M. W. Hall, Ken Kennedy, Kathryn S. Mckinley. "*Interprocedural Transformations for Parallel Code Generation*," Technical Report 1149-s, Dept. of computer Science, Rice University, 1991.

[8] C. A. Huson. "*An inline subroutine expander for Parafrase*," Masters Thesis, Dept. of computer Science, University of Illinois, 1982.

[9] Z. Li aand P. C. Yew, "*Efficient interprocedural analysis for program restructuring for parallel programs*," In Proceedings of the SIGPLAN : Experience with Applications, Languages and Systems, 1988.

[10] Z. Li aand P. C. Yew, "*Interprocedural analysis and program restructuring for parallel programs*," Technical Report CSRD 720, University of Illinois, Urbana-Champaign, 1988.

[11] Kathryn S. Mckinley, "A Compiler Optimization Algorithm for Shared Memory Multiprocessors," IEEE Transactions on Parallel and Distributed Systems, 9(8) : 769 787, August, 1998.

[12] R. W. Scheifler. "*An analysis of inline substitution for a structured programming language*," Communications of the ACM, 1977.

[13] M. J. Wolfe, "*Optimizing Supercompilers for Supercomputers*," PhD thesis, University of Illinois at Urbana-Champaign, 1982.

[14] M. J. Wolfe, "*High Performance Compilers for Parallel Computing*," Addison-Wesley Publishing Company, 1995.

[15] Hans Zima, "*Supercompilers for Parallel and Vector computers*," ACM press, 1990.

[16] 송원봉, 박두순, "*중첩루프에서 병렬화를 위한 종속성 제거*", 한국정보처리학회논문지, Vol.5, No.8, Aug. 1998.

[17] 안준선, 최광훈, 김성훈, 한태숙, 최광무, "*병렬화 컴파일러의 소개*", 정보과학회지 제14권 제7호, 1996.

[18] 이만호, "*병렬화를 위한 루프구조의 변환*", 정보과학회지 제12권 제5호, 1994.

[19] 장유숙, 박두순, "*프로시저 호출을 가진 루프에서 병렬성 추출*", 한국정보처리학회 춘계 학술발표논문집 제8권 제1호, 2001.

장 유 숙

e-mail : usjang@unitel.co.kr

1989년 한밭대학교 전산학과 졸업(학사)

1991년 한남대학교 대학원 컴퓨터공학과 졸업(공학석사)

2001년 순천향대학교 대학원 전산학과 졸업(공학 박사)

관심분야 : 병렬 처리, 병렬 컴퓨터, 컴파일러, 멀티미디어

박 두 순

e-mail : parkds@sch.ac.kr

1981년 고려대학교 수학과 졸업 (학사)

1983년 충남대학교 전산학과(이학석사)

1988년 고려대학교 전산학전공 (이학박사)

1985년~현재 순천향대학교 정보기술공학부 교수

2000년~현재 순천향대학교 컴퓨터교육원 원장

2000년~현재 정보처리학회 논문지 편집위원

관심분야 : 병렬 처리, 컴파일러, 멀티미디어 정보검색, 가용성, 컴퓨터 교육