개선된 자료 종속성 제거 알고리즘

장유숙*, 이원규**, 유헌창**, 박두순*
*순천향 대학교 정보기술공학부
** 고려 대학교 컴퓨터 교육과

An Improved Data Dependency Elimination Algorithm

Yu-Sug Chang*, Wongyu Lee**, Heonchang Yu**, Doo-Soon Park*
*Division of Computer Science and Computer Engineering,
College of Engineering Soonchunhyang University
**Dept. of Computer Education, College Education, Korea University
E-mail : *usjang@unitel.co.kr, **lee@comedu.korea.ac.kr,
**yuhc@comedu.korea.ac.kr, *parkds@sch.ac.kr

요 약

프로그램 수행시간의 대부분이 루프 구조에서 소비되고 있기 때문에 루프 구조를 가진 순차 프로그램에서 병렬성을 추출하는 연구들이 많이 행해지고 있고 그 연구들은 하나의 프로시저 내 루프 구조의 변환에 치중되고 있다. 그러나 대부분의 프로그램들은 프로시저 간 잠재된 병렬성을 가지고 있다. 본 논문에서는 프로시저 호출을 가진 루프에서 병렬성 추출 방식을 제안한다. 프로시저 호출을 포함하는 루프의 병렬화는 대부분 자료종속거리가 uniform 형태의 코드에서만 집중되었다. 본 논문에서는 자료종속거리가 uniform 코드, nonuniform 코드 그리고 복합된 코드를 가진 프로그램에서 적용 가능한 알고리즘을 제시하였으며, 제안된 알고리즘과 loop extraction, loop embedding 그리고 procedure cloning변환 방법을 CRAY-T3E로 성능 평가하였다. 성능평가 결과는 제안된 알고리즘이 효율적이라는 것을 보여준다.

1. 서론

프로그램 수행시간의 대부분이 루프 구조에서 소비되고 있기 때문에 순차 프로그램을 병렬 프로그램으로 변환하는 연구들은 루프 구조의 변환에 치중되고 있다. 그러나 루프에 프로시저호출을 포함하는 경우, 많은 잠재적인 병렬성을 가지고 있으며 컴파일러는 그것으로부터 최대의 병렬성을 추출하기를 원할 것이다.

프로시저 호출을 포함하는 루프의 병렬화는 대부분 자료종속거리가 uniform 형태의 코드에서만 집중되었다. 그러나 대부분의 프로그램들은 자료종속거리가 nonuniform이다. 자료 종속 거리가 nonuniform인 경우 자료 종속 거리를 컴파일 시간에 알기 어렵다. 그래서 컴파일 시간 병렬성을 추출해서 병렬코드로 생성하기가 어렵다.

본 논문에서는 프로시저 호출을 가진 루프에서 자료종속거리가 uniform, nonuniform 그리고 complex 코드를 가진 프로그램에서 적용 가능한 병렬성 추출 알고리즘을 제시한다. 본 논문에서 제시한 방법은 프로시저 내 변환 방법 중 가장 효과적인 방법인 자료 종속성 제거 방법[16]을 확장하여 프로시저 간 변환 방법에 적용하는 방법이다. 이를 위하여 프로시저간 변환 방법인 inlining 확장 변환 방식을 적용하고, 확장된 자료 종속성 제거 방법을 적용하는 방법이다.

성능평가는 제안된 알고리즘과 프로시저간 변환 방법 중 성능이 우수한 방법으로 알려진 loop extraction, loop embedding 그리고 procedure cloning 변환 방법을 CRAY-T3E로 비교하였다. 그리고 제안된 알고리즘이 효율적인 방법이라는 것을 보여준다.

본 논문의 구성은 2장에서 프로시저 분석과 변환을 서술하였고 3장에서는 프로시저 호출을 가진 루프에서 자료종속성 제거 방법을 이용한 병렬성 추출알고리즘을 제시하였다. 4장에서는 성능분석을 하고 5장에서 결론을 서술한다.

2. 프로시저 분석과 변환

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 내 변환은 프로시저 내에서 사용되는 변환 방법으로 loop distribution, loop fusion, loop reversal, loop skewing, loop interchange, loop tiling 등의 변환 방법이 있다[8,11,13].

본 논문에서는 프로시저 내 변환 방법 중 자료종속성 제거 방법을 이용하므로 자료종속성 방법에 대해서 간단하게 서술한다.

병렬성을 추출하기 위한 자료 종속성 제거 방법에서 종속성 행렬 DMLCS(Dependence Matrix with the number of nested Loop, the number of computation, and the number of Statement)는 자료 종속성을 계산하기 위해 자료 종속성을 표현하는 정방 행렬이다. 자료 종속성 제거 방법은 중첩 루프를 갖는 그림 2.1의 예를 가지고 서술한다[16].

```
      DO I = 1, N1
        DO J = 1, N2
S1:       A(I+2,J+1) = B(4*I,3*J) + (C(4*I,5*J+2)
S2:       B(5*I,4*J) = A(I+4,J+4) + B(I+4,3*J) + C(3*I+1,4*J+2)
S3:       C(7*5,6*J) = A(I+5,J+3) + B(I+2,J+3) + C(I+1,J+2)
        ENDDO
      ENDDO
```

그림 2.1 두 개의 중첩 루프를 갖는 예

그림 2.1은 2개의 중첩된 루프를 가졌고 루프 안에 있는 문장의 개수는 3이다. 이에 대한 종속성 행렬의 초기 상태는 DMLCS(2,1,3)이다. 여기서 변수 B의 경우 문장 S_2에서 문장 S_3으로 가는 흐름 종속이 존재한다. 이 경우에 자료 종속성은 $5*I'$와 $I''-2$의 값이 같아지는 정수 해 I', I''의 값을 구해야 하고 $4*J'$와 $J''-3$의 값이 같아지는 정수 해 J', J''값을 구해야한다. 이에 대한 초기 해는 DMLCS(2,1,3)행렬의 2행 3열에 나타내어진다.

그림 2.1의 DMLCS(2,1,3)은 그림 2.2와 같고,

	S_1	S_2	S_3
S_1	0	@((1⊕3,3⊕1),(1⊕1,4⊕1))	@((1⊕1,4⊕1),(1⊕1,3⊕1))
S_2	((0⊕4,0⊕5),(0⊕3,0⊕4))	((1⊕1,9⊕5),(0⊕3,0⊕4))	((1⊕1,7⊕5),(1⊕1,7⊕4))
S_3	((0⊕4,0⊕7),(3⊕5,4⊕6))	((2⊕3,5⊕7),(1⊕4,2⊕6))	((1⊕1,8⊕7),(1⊕1,8⊕6))

그림 2.2 자료 종속성 행렬 DMLCS(2,1,3)

그림 2.2에서 문장 S_2에서 문장 S_3으로 자료 종속성이 존재하고 초기 해는 $((1⊕1,7⊕5),(1⊕1,7⊕4))$이며 $((1⊕1,7⊕5),(1⊕1,7⊕4))$가 구해지는 과정과 의미는 다음과 같다.

먼저 문장 S_2에서 문장 S_3으로 종속 관계를 나타내는 2행 3열의 순서쌍 원소 즉 $((a_{23}⊕u_{23},b_{23}⊕v_{23}),(c_{23}⊕w_{23},d_{23}⊕x_{23}))$은 문장 S_2의 변수 $B(5*I,4*J)$와 문장 S_3의 변수 $B(I-2,J-3)$의 첨자 값이 모두 같은 경우를 표현하는 것으로 diophantine방정식을 이용하여 구할 수 있다. 즉 이에 대한 diophantine방정식은

$$5 * I' = I'' - 2$$
$$4 * J' = J'' - 3$$

을 만족하는 초기 정수 해 I', I'', J', J''을 구하는 것으로 diophantine방정식을 이용하면 I', I''은 각각 1과 7이 되며 J', J''도 1과 7이 된다. 또한 I, I'', J' J''의 계수 v_{23}, u_{23}, x_{23}, w_{23}은 각각 1, 5, 1, 4가 됨을 확인할 수 있다. 따라서 초기 정수 해 I', I'', J', J''는 각각 a_{23}, b_{23}, c_{23}, d_{23}에 대응하므로 2행 3열의 순서쌍 원소 $((a_{23}⊕u_{23},b_{23}⊕v_{23}),(c_{23}⊕w_{23},d_{23}⊕x_{23}))$는 $((1⊕1,7⊕5),(1⊕1,7⊕4))$으로 표현된다. 또한 위와 같이 구해진 순서쌍의 집합이 갖는 의미는 다음과 같다. 즉, $((1⊕1,7⊕5),(1⊕1,7⊕4)) = ((a_{23}⊕u_{23},b_{23}⊕v_{23}),(c_{23}⊕w_{23},d_{23}⊕x_{23}))$가 되므로 종속 관계에 있는 문장 S_2와 문장 S_3은 $I'=a_{23}=1$, $J'=c_{23}=1$인 경우 문장 S_2는 $B(5,4) = A(-3-3)+B(-3,3)+C(2,2)$가 되고 $I''=b_{23}=7$, $J''= d_{23} = 7$인 경우 문장 S_3은 $C(49,49) = A(2,4)+B(5,4)+C(6,5)$가 되어 변수 $B(5,4)$에 의하여 종속 관계가 존재함을 알 수 있고, 이어서 종속 관계에 있는 문장 S_2의 I'와 J'의 증가치가 각각 1이므로 $I' = a_{23}+u_{23} = 1+1 = 2$, $J' = c_{23}+w_{23} = 1+1 = 2$인 경우 문장 S_2는 $B(10,8) = A(-2,-2) + B(-3,6) + C(5,6)$가 되고, 문장 S_3의 I''과 J''의 증가치는 각각 5와 4이므로 $I'' = b_{23} + v_{23} = 7+5 = 12$, $J'' = d_{23}+x_{23} = 7+4 = 11$인 경우 문장 S_3은 $C(84,66) = A(7,8)+B(10,8)+C(11,9)$가 되어 변수 $B(10,8)$에 의하여 종속관계가 존재함을 알 수 있다. 고로 증가 치에 따라서 첨자 변수의 최종 치까지 종속관계가 형성됨을 알 수 있다. 그러므로 $((1⊕1, 7⊕5),(1⊕1, 7⊕4))$는 문장 S_2에서 문장 S_3으로 종속 관계를 함축시켜 표시하고 있음을 알 수 있다.

이러한 행렬의 계산을 이용하여 자료종속성을 계산하고, 프로그램을 수행하는 기법에 의해 자료종속성을 제거하는 방법이다.

프로시저 간 변환은 프로시저 사이에 적용되는 변환 방법으로 inlining 확장, loop extraction, loop embedding 그리고 procedure cloning변환 방법 등이 있다[3, 6, 7, 12].

3. 프로시저 호출을 가진 루프에서 자료 종속성 제거 방법을 이용한 병렬성 추출

프로시저 호출을 가진 루프에서 병렬성 추출을 하기 위하여 자료 종속성 제거 방법을 확장하였다. 자료 종속성 제거 방법은 프로시저 내 변환 방법이지만 이를 프로시저 호출을 가진 루프에 사용하기 위하여 호출 프로시저의 호출 위치들을 피 호출 프로시저의 코드로 대체하는 inlining을 수행한 후 자료종속성 제거 방법을 적용하는 알고리즘을 제시한다.

<알고리즘 1>
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 병렬 코드 생성

<알고리즘 2>
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 N_1, N_2인 경우
   Forall DMB행렬의 i, j에 대하여
   IF ($(a_{ij} \oplus u_{ij}) > N_1$ || $(c_{ij} \oplus w_{ij}) > N_2$)
   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 (u_{ij} & w_{ij} == 1)
   THEN uniform
   ELSE IF (u_{ij} & w_{ij} != 1) THEN nonuniform
   ELSE complex형태의 doall S_i 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 S_i 문장으로 변환

<알고리즘 3>
/* 프로시저 내 변환 먼저 적용하고 inlining을 제외한 프로시저 간 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 4>
/* inlining 후 프로시저 내 변환 적용 */
{
1. reverse topological order로 inlining 적용
2. repeat(각 프로시저가 최적화될 때까지)
   { /* 프로시저 내 변환 적용 */ }
}

4. 성능 평가

자료종속성 거리가 uniform, nonuniform 그리고 complex에 대해서 프로시저를 포함하는 예를 사용하여 자료 종속성 제거 방법을 이용한 프로시저 변환 알고리즘과 loop extraction변환 방법, loop embedding변환 방법 그리고 procedure cloning변환 방법과 비교 분석한다.

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

SUBROUTIN Q(f, i)               SUBROUTIN Q(f, i)
real f(n, n)                    real f(n, n)
integer i, j                    integer i, j
do j = 1, 100                   do j = 1, 100
  f(i, j) = f(i,j) + ...          f(i, j*5) = f(i, j*4) +
enddo                           ...
                                enddo
   (a) uniform                     (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 4.1 자료종속성거리

4.1 프로시저를 포함한 코드의 최적화 알고리즘을 이용한 병렬코드

프로시저를 포함한 코드의 최적화 알고리즘은 자료 종속성 제거 방법을 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 4.1(a)의 예제 코드를 프로시저를 포함한 코드의 최적화 알고리즘을 적용하여 병렬코드로 변환한 코드는 그림 4.2(a)와 같다.

```
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 4.2 최적화 알고리즘

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. 그림 4.2의 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 4.1(a) 예제코드의 최적화 알고리즘은 병렬 수행을 두 번 수행한다.

4.2 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 4.1(a) 예제 코드에 loop extraction을 적용한 코드가 그림 4.3(a) 코드이고 병렬화를 위해서 loop fusion변환 방법과 loop interchange변환 방법을 적용한 코드는 그림 4.3(b)과 같다. 그림 4.3(b)코드에서 호출 프로시저 P는 외부 루프 j는 병렬로 수행하고 내부 루프 i는 순차적으로 피 호출 프로시저 Q들을 호출한다.

```
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) + ...
```

```
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) + ..
```

(a) Loop extraction (b) Fusion, Interchang 후 병렬화

그림 4.3 Loop extraction

4.3 Loop embedding변환 방법을 이용한 병렬코드

프로시저 호출을 포함하는 루프 헤더를 피 호출 프로시저로 이동하는 변환 방법인 loop embedding을 그림 4.1(a)의 예제 코드에 적용한 후 병렬화를 위해 loop interchange변환 방법을 적용하였다.

```
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
```

```
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
```

(a) Loop embedding (b) Loop interchang 후 병렬화

그림 4.4 Loop embedding

변환된 병렬코드는 그림 4.4와 같이 호출 프로시저 P는 두개의 피 호출 프로시저 Q를 호출하고, 피 호출 프로시저 Q는 외부 루프 j는 병렬로 수행하고 내부 루프 i는 순차적으로 실행한다.

4.4 Procedure cloning변환 방법을 이용한 병렬코드

임의 프로시저가 여러 차례 호출될 때 프로시저를 최적화한 프로시저로 복사하여 많은 프로시저가 호출하게 하는 변환 방법이다. 그림 4.1(a)의 예제 코드를 procedure cloning을 적용한 코드는 그림 4.5와 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                    SUBROUTIN Qclone(f, i, j)
real a(n, n)                    real f(n, n)
integer i, j                    integer i, j

parallel do j = 1, N2           parallel do i = 1, N1
      call Qclone(a, i, j)            f(i, j) = f(i, j) + ..
      call Q(a, i+1, j)         end parallel do
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
```

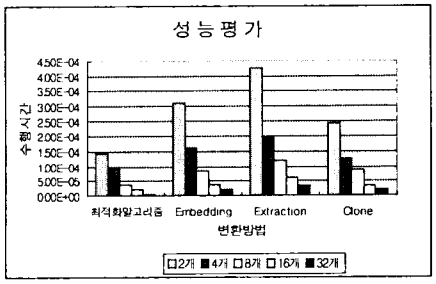
그림 4.5 Procedure cloning

4.5 성능평가

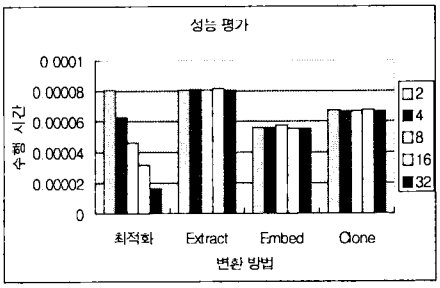
데이터 수는 N_1은 *20*, N_2는 *100*개로 하고 프로세서 수를 *2, 4, 8, 16, 32*개로 증가하였다.

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 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 순차 처리한다.

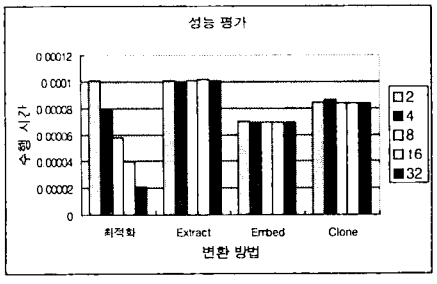
성능 평가 결과는 그림 4.6과 같이 자료 종속성 제거 방법을 이용한 최적화 알고리즘은 자료종속거리 uniform, nonuniform 그리고 complex 모두 프로세서 증가에 비례하여 성능이 좋아지고, loop embedding, loop extraction 그리고 procedure변환방법 중에서 uniform인 경우 procedure cloning 변환 방법이 우수하고, nonuniform과 complex 코드인 경우에는 본 논문에서 제시한 방법이 가장 우수함을 알 수 있다.



(a) uniform



(b) nonuniform



(c) complex

그림 4.6 프로시저를 포함한 병렬 코드의 성능평가

5. 결 론

성능평가를 위하여 사용된 코드들은 컴파일 시간에 프로시저 내 변환에 주로 사용되는 uniform형태뿐만 아니라 자료 종속성의 거리를 컴파일 시간에 알지 못하기 때문에 실행시간에 수행하는 프로시저 내 변환 방법으로 사용되는 nonuniform코드와 uniform과 nonuniform을 모두 포함하는 복합된 코드를 사용하였다. 제시된 방법은 자료 종속성 거리가 uniform, nonuniform, complex코드들에 대하여 컴파일 시간에 적용 가능하며 다른 방법들보다 매우 효율적인 방법이라는 것을 보여준다. 즉, 자료 종속거리가 uniform, nonuniform, complex코드에 대하여 컴파일 시간에 병렬성을 최대로 추출하는 프로시저간 변환 방법임을 보인다.

[참고문헌]

[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, P152-161, 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, volume 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] Song W. B., Park D. S., Kim B. S., Kong Y. H., "*Extracting Parallelism in Nested Loops*", Proceedings of International Computer Software & Applications Conference, IEEE, Seoul, p 41-47, Aug. 1996.