병렬 계산을 위한 최대 병렬성 추출 방법

박두순†

요 약

대부분의 프로그램 실행 시간은 루프 구조에서 소비되기 때문에 순차 루프 프로그램으로부터 병렬성을 추출하는 것은 프로그램을 빠르게 실행하는 데 필수적이다. 병렬성을 추출하기 위한 기존의 연구들은 주로 불변 자료 종속 거리에 초점을 맞추어왔다. 본 논문에서는 중첩 루프에서 자료 종속성을 제거하는 방법과 자료 종속성 제거 방법을 확장한 프로시저 호출을 가진 루프에서 병렬성을 추출하는 방법을 제안한다. 이 두 가지 방법들은 모두 자료 종속 거리에 관계없이 적용할 수 있다. 중첩 루프에서의 자료 종속성 제거 방법과 프로시저 호출을 가진 루프에서 병렬성을 추출하는 방법을 기존의 방법들과 CRAY-T3E에서 성능 평가를 하였다. 두 개의 방법 모두가 기존의 방법들보다 매우 우수함을 보였다.

키워드 : 병렬 처리, 병렬 컴파일러, 병렬성 추출, 자료 종속성

Extracting Maximum Parallelism for Parallel Computing

Doo-Soon Park†

ABSTRACT

Since the most program execution time is consumed in a loop structure, extracting parallelism from sequential loop programs is critical for the faster program execution. Conventional studies for extracting the parallelism are focused mostly on a uniform data dependence distance. In this paper, we proposed data dependency elimination method for a nested loop and extended data dependency elimination method to extract parallelism from the loop with procedure calls. The data dependency elimination method and the extended data dependency elimination method can be applied to uniform and non-uniform data dependency distance. We compared our method with conventional methods using CRAY-T3E for the performance evaluation. The results show that the proposed algorithms are very effective.

Keywords : Parallel Processing, Parallel Compiler, Extracting Parallelism, Data dependency

1. 서 론

정보의 량이 폭발적으로 증가함에 따라 이를 빠른 시간 안에 처리하기 위한 방법으로 병렬 처리 시스템들이 출현하였다. 그러나 더 많은 프로세서들을 붙인다고 더 빠른 컴퓨터가 되는 것은 아니다. 병렬 컴퓨터의 성능을 향상시키기 위해서는 병렬성(parallelism)을 탐지해내고, 이를 병렬 코드로 변환해주는 병렬 컴파일러(parallel compiler)와 병렬성을 잘 표현할 수 있는 병렬 언어(parallel language)의 개발 등 병렬 환경이 좋아야 한다. 많은 연구들은 병렬 컴퓨터를 위한 새로운 병렬 환경들을 제시해왔다. 이러한 환경들 중에서 병렬 컴파일러는 원시 프로그램이 순

† 종신회원: 순천향대학교 정보기술공학부 교수
논문접수: 2004년 11월 19일, 심사완료: 2004년 11월 24일

차 프로그램(sequential program)이라고 하면 묵시적인 병렬성을 탐지하고, 순차 프로그램을 병렬 프로그램으로 변환한다. 이와 같은 병렬 컴파일러들은 일리노이 대학의 PARAFRASE I, II[10], IBM의 PTRAN[1], 본 대학의 SUPERB[17], 그리고 스탠포드 대학의 SUIF등이 있다.

병렬 컴파일러의 가장 기본적인 것은 순차 프로그램으로부터 얼마나 효율적으로 최대의 병렬성을 추출해내느냐 하는 것이다. 이 방법은 병렬처리 시스템의 속도 향상을 위해서는 필수적이다. 이러한 방법들은 자료 종속성 거리(data dependency distance)가 불변(uniform)인 경우에 적용하는 방법과 가변(non-uniform)에 적용 할 수 있는 방법으로 분류된다. 또한, 중첩 루프(nested loop)에서 적용할 수 있는 루프 변환 방법과 프로시저 호출(procedure call)을 가진 루프에서 적용할 수 있는 변환 방법들이 있다.

기존의 방법 중에서 중첩 루프에서 자료 종속성 거리가 불변인 경우에 적용 가능한 방법은 interchanging[2], tiling[16], unimodular[2], selective cycle shrinking[4], Hollander[4], Chen&Wang[3], skewing[2] 방법들이 있다. 자료 종속성 거리가 가변인 경우에 적용 가능한 방법은DCH[13]과 IDCH[11] 방법들이 있다. 이러한 방법들은 원시 프로그램의 자료 종속성을 분석하고, 자료 종속성 거리가 불변인 경우와 가변인 경우 각각 다른 방법들에 따라서 병렬 코드로 분할한다. 자료 종속성 거리에 따라 가장 알맞은 방법을 선택하여야 하고, 자료 종속성 거리가 불변이나 가변에 모두 적용 가능한 방법이 존재하지 않기 때문에 이러한 문제점을 해결하기 위하여 자료 종속성이 불변이든 가변이든 하나의 방법으로 사용 가능하고, 기존의 방법들보다 병렬성 추출이 뛰어난 자료 종속성 제거 방법을 제안한다.

또한, 프로시저 호출을 가진 루프에서 적용할 수 있는 방법으로 자료 종속성 거리가 불변인 경우 loop extraction[6], loop embedding[5], procedure cloning[9] 방법들이 있다. 그러나 자료 종속성 거리가 가변인 경우에 적용할 수 있는 방법은 아직 제안되지 않았다. 본 논문에서는 자료 종속성 제거 방법을 확장하여 프로시저 호출을 가진 루프에서 자료 종속성 거리가 불변이든 가변이든 한번에 사용 가능하며, 기존의 방법보다 병렬성 추출이 뛰어난 확장된 자료 종속성 제거 방법도 제안한다.

중첩 루프에서 본 논문에서 제안된 자료 종속성 제거 방법이 효율적이라는 것을 보여주기 위하여 자료 종속성 거리가 불변인 경우 기존의 대표적인 방법들인 tiling, interchanging, unimodular, selective cycle shrinking, Hollander, Chan&Wang 방법들과 CRAY-T3E에서 성능 평가하였고, 가변인 경우에는 DCH 와 IDCH 방법들과 CRAY-T3E에서 성능 평가하였다. 또한, 프로시저 호출을 가진 루프에서 자료 종속성 거리가 가변인 경우에는 아직 기존 방법이 제시되지 않았기 때문에 자료 종속 거리가 불변, 가변, 혼합인 경우에 본 논문에서 제시한 확장된 자료 종속성 제거 방법과 자료 종속성이 불변인 경우에 적용 가능한 loop extraction, loop embedding 그리고 procedure cloning 방법들과 CRAY-T3E에서 성능 평가하였다.

본 논문의 구성은 2장에서 자료 종속성 제거 방법을 제시하였고, 3장에서는 확장된 자료 종속성 제거 방법을 제시하였다. 4장에서는 성능 평가를 서술하였고, 5장에서는 결론을 내렸다.

2. 자료 종속성 제거 알고리즘

본 장에서는 중첩 루프에서 적용 가능한 자료 종속성 제거 알고리즘을 서술하기 위해 필요한 정의, 보조 정리, 정리, 알고리즘 등을 서술한다.

2.1. 자료 종속성

자료 종속성은 흐름 종속(flow dependence), 반종속(anti dependence), 그리고 출력 종속(output dependence) 등으로 분류할 수 있다. 두 문장 S_i와 S_j가 있을 때 문장 S_i에서 변수 X가 정의(define)되고 문장 S_j에서 변수 X를 사용(use)하면, 이를 흐름 종속($S_i \delta S_j$) 이 존재한다고 하고 문장 S_i는 문장 S_j보다 먼저 수행되어야 한다. 두 문장 S_i와 S_j가 같은

변수를 정의하고 문장 S_i가 문장 S_j 보다 먼저 수행되는 경우를 출력 종속($S_i\ \delta^o\ S_j$) 이라고 부른다. 문장 S_i가 변수 X를 먼저 사용하고 문장 S_j가 변수 X를 정의하는 경우에 문장 S_i가 문장 S_j보다 먼저 수행되면, 반 종속($S_i\ \delta^a\ S_j$)이라고 한다.

자료 종속성을 분석하는 방법은 여러 가지가 있다. separability test[14], GCD test[1], Power test[15], I test[7], λ test[8] 등이 자료 종속성 분석을 위해서 사용된다. 그러나 이 방법들은 모두 자료 종속성 거리가 불변인 경우에만 적용이 가능하고 자료 종속성 거리가 가변인 경우에는 적용하지 못한다.

배열 변수의 첨자식과 자료 종속성에 대한 연구[12]에 의하면 불변 종속 거리는 13.65%이고, 가변 종속 거리는 86.35%이다. 따라서 병렬처리를 효율적으로 수행하기 위해서는 불변 종속거리는 물론 가변 종속거리가 존재하는 루프에 대해서도 적용할 수 있어야 한다. 자료 종속 거리는 최적화된 병렬 루프를 변환하는데 중요한 정보로 사용되며, 첨자 식(subscript expression)에 따라 불변 종속 거리와 가변 종속 거리로 나눌 수 있다. 만약 하나의 루프 첨자 변수로만 구성된 첨자식이 각각 a×i+b, c×i+d(여기서 a, b, c, d는 정수, i는 루프 변수)라 하면 첨자 식에서 a=c이면, 종속 거리는 | (a×i±b)-(c×i±d) | 인 불변 종속 거리가 존재하고 a≠c이면 가변 종속 거리가 존재한다. 불변 종속 거리 종속은 하나의 종속 거리만 존재하게 되고, 가변 종속 거리 종속은 여러 개의 종속 거리가 존재하게 된다.

[정의 1]

종속성 행렬(Dependence Matrix; DM)은 자료 종속성을 계산하기 위한 정방행렬이다. DM의 각각의 원소는 각각의 자료 종속성을 표현하는 순서쌍이다. DM(k,t,m)은 중첩된 루프의 개수가 k(≥1)개 이고, t는 종속성 행렬이 계산된 횟수로서 초기치는 1이다. m(≥2)은 루프 안에 있는 문장의 개수를 나타낸다.

만일, 루프 안에 있는 문장이 n개이고 2개의 중첩된 루프를 가지면서 종속성 행렬의 초기상태를 나타내는 경우는 DM(2,1,n)이고, 이는 다음과 같이 표현할 수 있다.

$$DM\ (2,1,n) = \begin{pmatrix} (x_{11}, y_{11}), (x_{12}, y_{12}), \Lambda\ , (x_{1n}, y_{1n}) \\ (x_{21}, y_{21}), (x_{22}, y_{22}), \Lambda\ , (x_{2n}, y_{2n}) \\ \mathbf{M} \\ (x_{n1}, y_{n1}), (x_{n2}, y_{n2}), \Lambda\ , (x_{nn}, y_{nn}) \end{pmatrix}$$

각 순서쌍의 원소는 임의의 i, j($1 \le i,\ j \le n$)에 대하여 $x_{ij}=(a_{ij} \oplus u_{ij},\ b_{ij} \oplus v_{ij})$, $y_{ij}=(c_{ij} \oplus w_{ij},\ d_{ij} \oplus x_{ij})$ 이다. 위에서 $s \oplus t$는 첨자 식 s×i+t(i는 루프 첨자 변수이고, s와 t는 상수)를 표현한다. 이는 문장 S_i로부터 문장 S_j로 자료 종속이 존재한다는 것을 보여준다. 또한, DM에서 원소가 0인 경우는 자료 종속성이 존재하지 않음을 의미한다. (a_{ij},b_{ij})는 문장 S_i로부터 문장 S_j로 자료 종속이 존재하는 경우, 밖의 중첩 루프(outer nested loop) 첨자들 간의 diophantine 방정식 $v_{ij}\times J+g=u_{ij}\times P+h$의 초기해(initial solution)이고 (c_{ij},d_{ij})는 안쪽 중첩 루프(inner nested loop) 첨자들 간의 diophantine 방정식 $x_{ij}\times J+g'=w_{ij}\times Q+h'$의 초기해 이다. @$((a_{ij} \oplus u_{ij}, b_{ij} \oplus v_{ij}),(c_{ij} \oplus w_{ij}, d_{ij} \oplus x_{ij}))$는 불변 종속거리를 나타내며 #$((a_{ij} \oplus u_{ij}, b_{ij} \oplus v_{ij}),(c_{ij} \oplus w_{ij}, d_{ij} \oplus x_{ij}))$는 수정된 첨자의 상계(upper bound)이다.

[예제 1]

<그림 1>와 같이 3개의 문장을 가진 두 개의 중첩루프를 갖는 프로그램이 있다고 하자.

```
DO I = 1, M
  DO J = 1, N
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*I,6*J) = A(I-5,J-3) + B(I-2,J-3)+ C(I-1,J-2)
```

<그림 1> 중첩 루프의 예

종속성 행렬의 초기 상태는 DM(2,1,3)이다. 여기서 변수 B의 경우 문장 S_2로부터 문장 S_3으로 가는 흐름 종속이 존재한다. 이 경우에 자료 종속성은 5*I'와 I"-2의 값이 같아지는 정수해 I',I"의 값을 구해야 하고 4*J'와 J"-3의 값이 같아지는 정수해 J',J"값을 구해야한다. 이에 대한 초기 해는 DM(2,1,3) 행렬의 2행 3열에 나타내어

진다. <그림 1>의 DM(2,1,3)은 <그림 2>이다.

S_1	S_2	S_3
0	@((⊕3,3⊕1),(1⊕1,4⊕1))	@((⊕1,4⊕1),(1⊕1,3⊕1))
((0⊕4,0⊕5),(0⊕3,0⊕4))	((1⊕1,9⊕5),(0⊕3,0⊕4))	((1⊕1,7⊕5),(1⊕1,7⊕4))
((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> <그림 1>의 DM(2,1,3)

<그림 2>에서 문장 S_2에서 문장 S_3으로의 자료종속성이 존재하고, 초기 해는 ((1⊕1,7⊕5),(1⊕1,7⊕4))이다. 이는 DM(2,1,3)의 2행 3열이 $((a_{23}⊕u_{23}, b_{23}⊕v_{23}), (c_{23}⊕w_{23}, d_{23}⊕x_{23}))$이므로 Diophantine 방정식

$$5 * I' = I'' - 2$$
$$4 * J' = J'' - 3 \text{ ------- } (1)$$

의 해를 구하면 된다. I', I'', J', J'' 의 초기 정수해는 1, 7, 1, 7 이다. 그리고 v_{23}, u_{23} ,x_{23}, w_{23}은 1, 5, 1, 4 이다. 결국 $((a_{23}⊕u_{23}, b_{23}⊕v_{23}),(c_{23}⊕w_{23}, d_{23}⊕x_{23}))$ = ((1⊕1,7⊕5),(1⊕1,7⊕4)) 이다. 종속관계는 $I' = a_{23} = 1$, $J' = c_{23} = 1$ 인 경우 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 S_3은 C(49,42)=A(2,4)+**B(5,4)**+C(6,5)가 되어 문장 S_2에서 문장 S_3으로의 흐름 종속이 존재함을 알 수 있고 또한, 종속관계에 있는 S_2의 I'와J'의 증가치가 각각 1이므로 $I' = a_{23} + u_{23} = 1 + 1 = 2$, $J' = c_{23} + w_{23} = 1 + 1 = 2$인 경우 S_2는 **B(10,8)**=A(-2,-2)+B(-2,6)+C(5,6)가 되고 S_3의 I''와 J''의 증가치는 각각 5 와 4 이므로 $I'' = b_{23} + v_{23} = 7 + 5 = 12$, $J'' = d_{23} + x_{23} = 7 + 4 = 11$인 경우 S_3은 C(84,66)=A(7,8)+**B(10,8)**+C(11,9)가 되어 증가치에 따라서 종속관계가 형성됨을 알 수 있다. 그러므로 ((1⊕1,7⊕5),(1⊕1,7⊕4))는 문장 S_2에서 문장 S_3으로의 종속관계를 함축시켜 표시하고 있음을 알 수 있다.

[정의2]

종속성 행렬 DM(m,n,p)와 DM(m,1,p)의 곱은 연산자 ⊗에 대한 연산으로 종속성 행렬 DM(m,n+1,p)가 된다.

앞에서 정의한 DM 표현법을 사용하면

$$\begin{pmatrix} (x_{11}, y_{11}),(x_{12}, y_{12}),\Lambda\ ,(x_{1n}, y_{1n}) \\ (x_{21}, y_{21}),(x_{22}, y_{22}),\Lambda\ ,(x_{2n}, y_{2n}) \\ M \\ (x_{m1}, y_{m1}),(x_{m2}, y_{m2}),\Lambda\ ,(x_{mn}, y_{mn}) \end{pmatrix} \otimes \begin{pmatrix} (x_{11}, y_{11}),(x_{12}, y_{12}),\Lambda\ ,(x_{1n}, y_{1n}) \\ (x_{21}, y_{21}),(x_{22}, y_{22}),\Lambda\ ,(x_{2n}, y_{2n}) \\ M \\ (x_{n1}, y_{n1}),(x_{n2}, y_{n2}),\Lambda\ ,(x_{nn}, y_{nn}) \end{pmatrix} = \begin{pmatrix} AK\ B \\ M \\ C\Lambda\ D \end{pmatrix}$$

가 되며

$$A = \begin{pmatrix} \begin{pmatrix} ((a_{11}⊕u_{11}, b_{11}⊕v_{11}),(c_{11}⊕w_{11}, d_{11}⊕x_{11})) \\ ((a_{11}⊕u_{11}, b_{11}⊕v_{11}),(c_{11}⊕w_{11}, d_{11}⊕x_{11})) \end{pmatrix} \\ M \\ \begin{pmatrix} ((a_{1n}⊕u_{1n}, b_{1n}⊕v_{1n}),(c_{1n}⊕w_{1n}, d_{1n}⊕x_{1n})) \\ ((a_{n1}⊕u_{n1}, b_{n1}⊕v_{n1}),(c_{n1}⊕w_{n1}, d_{n1}⊕x_{n1})) \end{pmatrix} \end{pmatrix} D = \begin{pmatrix} \begin{pmatrix} ((a_{1n}⊕u_{1n}, b_{1n}⊕v_{1n}),(c_{1n}⊕w_{1n}, d_{1n}⊕x_{1n})) \\ ((a_{n1}⊕u_{n1}, b_{n1}⊕v_{n1}),(c_{n1}⊕w_{n1}, d_{n1}⊕x_{n1})) \end{pmatrix} \\ M \\ \begin{pmatrix} ((a_{1n}⊕u_{1n}, b_{1n}⊕v_{1n}),(c_{1n}⊕w_{1n}, d_{1n}⊕x_{1n})) \\ ((a_{n1}⊕u_{n1}, b_{n1}⊕v_{n1}),(c_{n1}⊕w_{n1}, d_{n1}⊕x_{n1})) \end{pmatrix} \end{pmatrix}$$

이다. 여기서 종속성 행렬 A의 한 원소 $\begin{pmatrix} ((a_{11}⊕u_{11}, b_{11}⊕v_{11}),(c_{11}⊕w_{11}, d_{11}⊕x_{11})) \\ ((a_{11}⊕u_{11}, b_{11}⊕v_{11}),(c_{11}⊕w_{11}, d_{11}⊕x_{11})) \end{pmatrix}$는 자료 종속성을 계산하기 위해 사용되는 행렬의 형태이다. 만일, $((a_{ij}⊕u_{ij}, b_{ij}⊕v_{ij}),(c_{ij}⊕w_{ij}, d_{ij}⊕x_{ij}))$이 문장 S_i로부터 문장 S_j로 종속관계를 나타내고, $((a_{kl}⊕u_{kl}, b_{kl}⊕v_{kl}),(c_{kl}⊕w_{kl}, d_{kl}⊕x_{kl}))$가 문장 S_j로부터 문장 S_k로 종속관계를 나타내면 $\begin{pmatrix} ((a_{ij}⊕u_{ij}, b_{ij}⊕v_{ij}),(c_{ij}⊕w_{ij}, d_{ij}⊕x_{ij})) \\ ((a_{kl}⊕u_{kl}, b_{kl}⊕v_{kl}),(c_{kl}⊕w_{kl}, d_{kl}⊕x_{kl})) \end{pmatrix}$는 문장 $S_i \rightarrow S_j \rightarrow S_k$로 가는 종속 관계를 나타낸다. 즉, 문장 $S_i \rightarrow S_k$로 가는 추이 종속 관계(transitive dependence relation)를 의미한다. 자료 종속에 대한 추이 종속 관계는 [보조 정리 1]과 같다.

[보조정리1]

종속성 행렬 DM(m,2,p)의 임의의 원소 $\begin{pmatrix} ((a_{ij} ⊕ u_{ij}, b_{ij} ⊕ v_{ij}),(c_{ij} ⊕ w_{ij}, d_{ij} ⊕ x_{ij})) \\ ((a_{kl} ⊕ u_{kl}, b_{kl} ⊕ v_{kl}),(c_{kl} ⊕ w_{kl}, d_{kl} ⊕ x_{kl})) \end{pmatrix}$는 통로(path)의 길이가 2인 추이 종속 관계를 나타내며, 만일

$$\begin{pmatrix} ((a_{ij} ⊕ u_{ij}, b_{ij} ⊕ v_{ij}),(c_{ij} ⊕ w_{ij}, d_{ij} ⊕ x_{ij})) \\ ((a_{kl} ⊕ u_{kl}, b_{kl} ⊕ v_{kl}),(c_{kl} ⊕ w_{kl}, d_{kl} ⊕ x_{kl})) \end{pmatrix}$$
$$= *((a_{mn} ⊕ u_{mn}, b_{mn} ⊕ v_{mn}),(c_{mn} ⊕ w_{mn}, d_{mn} ⊕ x_{mn}))$$

로 추이 종속 관계를 표현하면 a_{mn}, u_{mn}, b_{mn}, v_{mn}, c_{mn}, w_{mn}, d_{mn}, x_{mn}은 통로의 길이가 1인 첨자변수 a_{ij}, u_{ij}, b_{ij}, v_{ij}, c_{ij}, w_{ij}, d_{ij}, x_{ij}, a_{kl}, u_{kl}, b_{kl}, v_{kl}, c_{kl}, w_{kl}, d_{kl}, x_{kl}들의 최소공배수(LCM: Least Common Multiple)로 표현된다.

(증명)

$((a_{ij}⊕u_{ij},b_{ij}⊕v_{ij}),(c_{ij}⊕w_{ij},d_{ij}⊕x_{ij}))$는 문장 S_i로부터 문장 S_j로 종속 관계이고 $((a_{kl}⊕u_{kl},b_{kl}⊕v_{kl}),$

$(c_{kl} \oplus w_{kl}, d_{kl} \oplus x_{kl})$)는 문장 S_j로부터 문장 S_k로의 종속 관계를 나타낼 때 문장 S_i로부터 S_j를 거쳐서 S_k로 가는 통로가 2인 추이 종속 관계는 최소공배수에 의해서 구해진다는 것이다. (a_{ij}, b_{ij})는 문장 S_i로부터 문장 S_j로 자료 종속이 존재하는 경우, 밖의 중첩 루프 첨자들 간의 diophantine 방정식 $v_{ij} \times J + g = u_{ij} \times P + h$의 초기해이고, (u_{ij}, v_{ij})는 증가치이다. (c_{ij}, d_{ij})는 안쪽 중첩 루프 첨자들 간의 diophantine 방정식 $x_{ij} \times J + g' = w_{ij} \times Q + h'$의 초기해이고, (w_{ij}, x_{ij})는 증가치이다. 같은 방법으로 S_j에서 S_k로 가는 종속관계에 대해서는 (a_{kl}, b_{kl})는 문장 S_j로부터 문장 S_k로 자료 종속이 존재하는 경우, 밖의 중첩 루프 첨자들 간의 diophantine 방정식 $v_{kl} \times J + g = u_{kl} \times P + h$의 초기해이고, (u_{kl}, v_{kl})는 증가치를 나타내며 (c_{kl}, d_{kl})은 안쪽 중첩 루프 첨자들 간의 diophantine 방정식 $x_{kl} \times J + g' = w_{kl} \times Q + h'$의 초기해 이고, (w_{kl}, x_{kl})는 증가치를 나타낸다. $S_i \rightarrow S_j \rightarrow S_k$로 가는 추이 종속 관계인 $((a_{mn} \oplus u_{mn}, b_{mn} \oplus v_{mn}), (c_{mn} \oplus w_{mn}, d_{mn} \oplus x_{mn}))$에서, $S_i \rightarrow S_k$로 가는 추이 종속 관계의 초기치와 증가치인 a_{mn}과 u_{mn}은 S_i에서 S_j로 가는 첫 번째 첨자의 증가치인 v_{ij}와 S_j에서 S_k로 첫번째 첨자의 증가치인 u_{kl}의 최소공배수인 $a_{mn} = u_{mn} = LCM(v_{ij}, u_{kl})$임을 알 수 있다. 같은 방법으로

$c_{mn} = w_{mn} = LCM(x_{ij}, w_{kl})$

$b_{mn} = v_{ij} + b_{kl}(b_{kl} = 0$ 이면 $b_{kl} = v_{kl}) - 1$

$d_{mn} = x_{ij} + d_{kl}(d_{kl} = 0$ 이면 $d_{kl} = x_{kl}) - 1$

$v_{mn} = LCM(v_{ij}, v_{kl})$

$x_{mn} = LCM(x_{ij}, x_{kl})$가 성립된다.

즉, 종속 관계에 의한 추이 종속 관계는 증가치와 증가치에 대한 계산으로 수행된다. 결국, 추이 종속 관계는 통로의 길이가 1인 첨자변수들의 최소공배수로 표현된다.■

예를 들어, 한 종속 관계의 증가치가 3이고 다른 종속 관계의 증가치가 4이면 (3,4),(6,8),(9,12), … 에서 종속 관계가 성립된다. 12, 24, 36, … 는 증가치 3과 4의 최소공배수 값이다. 결국 추이 종속 관계는 두 종속 관계의 증가치들의 최소공배수가 된다.

2.2. 자료 종속성 제거 알고리즘

불변 종속거리 이거나 가변 종속거리를 갖는 루프에서의 종속성 제거 알고리즘을 도입한다.

일반적으로 불변 종속거리이거나 가변 종속거리를 갖는 루프의 일반문장은 다음과 같다.

$S_1 : a_1(a_1 I + b_1,\ e_1 J + f_1) = \cdots$

$S_2 : a_2(a_2 I + b_2,\ e_2 J + f_2) = a_1(c_1 I + d_1, g_1 J + h_1) + \cdots$

$\vdots \qquad\qquad \vdots$

$S_i : a_i(a_i I + b_i,\ e_i J + f_i) = a_{i-1}(c_{i-1} I + d_{i-1},\ g_{i-1} J + h_{i-1}) + \cdots$

$\vdots \qquad\qquad \vdots$

$S_n : a_n(a_n I + b_n,\ e_n J + f_n) = a_{n-1}(c_{n-1} I + d_{n-1},\ g_{n-1} J + h_{n-1}) + \cdots$

임의의 i, j($1 \le i,\ j \le n$)에 대하여 문장 S_i와 S_j 사이에 자료 종속이 존재한다면 기존의 방법들은 자료 종속성 관계를 알아본 다음 주로 분할하는 방법에 의해 병렬 코드를 만든다. 본 논문에서는 두 문장 사이에 자료 종속성이 존재하는 경우 분할을 하지 않고 문장들을 수행시키는 방법에 의해서 자료 종속성을 제거하는 방법이다. 두 문장 S_i와 S_j사이에 공통 변수 X가 자료 종속성이 존재하는 경우 먼저 두 문장 S_i 와 S_j를 동시에 수행시킨다. 그러면 문장 S_j의 X 값은 문장 S_i 의 X 값 때문에 올바를 수도 있지만 올바르게 않을 수도 있다. 이런 문제 때문에 자료 종속성이 존재하면 병렬처리가 불가능하다. 이를 해결하기 위해서 두 번째로 문장 S_j 만을 다시 한번 수행시키면 문장 S_j의 X 값도 올바르게 된다. 그러나 문장 S_i가 S_j에 자료 종속성을 갖고 문장 S_j가 문장 S_k에 자료 종속성을 갖는다면 이 경우에는 위와 같이 수행해도 문장 S_k의 X 값은 아직도 올바르지 않다. 이 경우에는 한 번 더 문장 S_k를 수행시켜야 문장 S_k의 X 값도 올바르게 된다.

결국 자료 종속성을 완전히 제거하기 위해서는 임의의 두 문장 사이에 추이 종속관계가 존재할 경우 몇 번의 통로의 길이를 거쳐서 추이 종속관계가 형성되고 있는지 즉, 통로의 길이를 알아야 하며 통로의 길이가 k라고 하면 k번 반복적으로 수행함으로써 자료 종속 관계는 완전히 사라지게 될 것이다.

[정리1]

두 개의 문장 사이에 종속관계의 증가치들의 최소공배수로부터 구해지는 추이 종속 관계는 유

한번 안에 종속 관계를 완전히 제거 할 수 있다.

(증명)

<보조 정리 1>에서 추이 종속 관계는 각각의 종속 관계에 따른 증가치에 대한 최소공배수로 계산 되어진다. 두 종속 관계의 증가치를 m_1, m_2 라고 하고, m_3 = LCM(m_1, m_2)라 하자. m_3이 루프 첨자 값의 최종치 n보다 크지 않다면 최소공배수를 계속 반복적으로 구한다.

그래서 최소공배수의 값을 $m_1,m_2,m_3,\cdots,m_K$ 라 하면 $m_1,m_2,m_3,\cdots,m_K$는 $1 \leqq m_1 \leqq m_2 \leqq m_3 \leqq \cdots \leqq m_K$ 인 정수가 된다. 결국 m_1부터 m_K까지의 개수를 k라 하면 $m_K \geqq n$이 되는 k가 항상 존재한다는 것이다. 만약 k=n이라면 순차적으로 수행하는 결과이므로 n번 반복 수행하면 자료 종속성이 사라지고, 만약 k>n 이라면 k번 수행하면 모든 자료 종속성이 사라진다. 그러므로 최소공배수에 의한 추이 종속 관계는 유한번 안에 종속 관계를 완전히 제거 할 수 있다.■

자료 종속성 제거 방법을 구현하는 가장 간단한 방법은 통로의 길이만큼 모든 문장들을 반복 수행하면 되지만 그런 경우 프로세서의 개수가 매우 많이 필요하게 된다. 프로세서의 개수를 가장 줄일 수 있는 방법은 정확하게 어떤 문장과 어떤 문장 사이에 자료 종속 관계가 있는지를 아는 것이다. 이를 위해서 최대공약수(Greatest Common Divisor)와 최소공배수를 이용하면 해결할 수 있다. 이와 같이 제안된 방법이 <알고리즘 1>, <알고리즘 2>, <알고리즘 3>이며, <알고리즘 3>을 수행하면 된다.

<algorithm 1>

```
FOR all a_ij, c_ij <> 0
IF the number of computation = 1 THEN
   IF  U_ij and W_ij = 1 THEN
      DOALL K=1, M-b_ij +1
        DOALL L=1, N-d_ij +1
                S_i
        ENDDOALL
      ENDDOALL
     ELSE
      DOALL K=U_ij, M, U_ij
        DOALL K=W_ij, N, W_ij
                S_i
        ENDDOALL
      ENDDOALL
  ELSE IF the number of computation = 2 THEN
     IF  U_K1 and W_K1 = 1 THEN
       DOALL K=b_K1, M
         DOALL L=d_K1, N
                 S_j
         ENDDOALL
       ENDDOALL
      ELSE
       DOALL K=U'_K1, M, U'_K1
         DOALL K=W'_K1, N, W'_K1
                 S_j
         ENDDOALL
       ENDDOALL
ELSE IF the number of computation = 3 THEN
     IF  U_uv and W_uv = 1 THEN
       DOALL K=b_K1 + b_uv - 1, M
         DOALL L=d_K1 + d_uv - 1, N
                 S_v
         ENDDOALL
       ENDDOALL
      ELSE
       DOALL K=U'_UV, M, U'_UV
         DOALL K=W'_uv, N, W'_uv
                 S_V
         ENDDOALL
       ENDDOALL
```

<algorithm 2>

Mutation of DM and calculation of new data dependency variable

1. LCM is the Least Common Multiplier.
2. IF path length is 2.

$$DM = \begin{pmatrix} ((a_{ij} \oplus u_{ij}, b_{ij} \oplus v_{ij}), & (c_{ij} \oplus w_{ij}, d_{ij} \oplus x_{ij})) \\ ((a_{kl} \oplus u_{kl}, b_{kl} \oplus v_{kl}), & (c_{kl} \oplus w_{kl}, d_{kl} \oplus x_{kl})) \end{pmatrix}$$

$$THEN \quad \begin{matrix} U'_{kl} = LCM(b_{ij} \oplus v_{ij}, a_{kl} \oplus u_{kl}) \\ W'_{kl} = LCM(d_{ij} \oplus x_{ij}, c_{kl} \oplus w_{kl}) \end{matrix}$$

$$\text{Translation of DM} = \begin{pmatrix} ((a_u \oplus u_u, b_u \oplus v_u), (c_u \oplus w_u, d_u \oplus x_u)) \\ ((u_u, u_u \times b_u \div a_u), (w'_u, w'_u \times d_u \div c_u)) \end{pmatrix}$$

3. IF path length is 3.

$$DM = \begin{pmatrix} ((a_{ij} \oplus u_{ij}, b_{ij} \oplus v_{ij}), & (c_{ij} \oplus w_{ij}, d_{ij} \oplus x_{ij})) \\ ((a_{kl} \oplus u_{kl}, b_{kl} \oplus v_{kl}), & (c_{kl} \oplus w_{kl}, d_{kl} \oplus x_{kl})) \\ ((a_{uv} \oplus u_{uv}, b_{uv} \oplus v_{uv}), & (c_{uv} \oplus w_{uv}, d_{uv} \oplus x_{uv})) \end{pmatrix}$$

$$THEN \quad \begin{matrix} U'_{uv} = LCM(b_{kl} \oplus v_{kl}, a_{uv} \oplus u_{uv}) \\ W'_{uv} = LCM(d_{kl} \oplus x_{kl}, c_{uv} \oplus w_{uv}) \end{matrix}$$

$$\text{Translation of DM} = \begin{pmatrix} ((a_{\cdots}, b_{\cdots}), (c_{\cdots}, d_{\cdots})) \\ ((u'_{\cdots}, u'_{\cdots} \times b_{\cdots} \div a_{\cdots}), (w'_{\cdots}, w'_{\cdots} \times d_{\cdots} \div c_{\cdots})) \end{pmatrix}$$

<algorithm 3>

1. Make a DM of path length 1.
2. IF there are DMs with identical subscript

THEN change to other name.
3. For all i, j
IF(a_{ij}⊕u_{ij}) > M *or* (c_{ij}⊕w_{ij}) > N
THEN change the element to 0
4. For all non-zero elements, apply <algorithm 1>.
5. IF b_{ij} > M or d_{ij} > N
THEN change the element to 0.
6. IF all element =0 THEN goto 12
ELSE increase path length by 1.
7. IF there exists identical elements
THEN remove all elements except one.
8. In DM, apply <algorithm 2> and look for data dependency and mutate DM.
9. IF the selected LCM value is bigger than that of the final DO loop
THEN change that element to 0.
10. Apply <algorithm 1> to every non-zero element.
11. Goto 6
12. IF there exists added number
THEN apply <algorithm 1>.
13. Except for the mutated sentences,
perform doall Si to every sentence.

3. 확장된 자료 종속성 제거 알고리즘

본 장에서는 자료 종속성 제거 방법을 확장하여 프로시저 호출을 가진 루프에서 병렬성 추출 방법을 서술한다.

프로시저 호출을 가진 루프의 변환 방법 중에서 호출된 프로시저(called procedure) 자리에 모든 문장들을 대체하는 방법인 inline 방법과 Loop extraction, Loop embedding, Procedure cloning 방법 등이 있다[5,6].

자료 종속성 제거 방법을 확장하여 프로시저 호출을 가진 루프에서 병렬성을 추출하기 위해서 프로시저 호출을 가진 루프에서 변환 방법 중 inline 방법을 먼저 적용하고 그 다음에 자료 종속성 제거 방법을 적용한다. 이에 대한 알고리즘은 <알고리즘4>, <알고리즘 5>, <알고리즘 6>, <알고리즘 7>과 같다.

<algorithm 4>

1. Draw procedure call multi-graph
2. Expand it into augmented call graph
3. Calculation of information between procedures
4. Dependency analysis
5. IF (one procedure is called and the caller related variables are not changed)
THEN goto <algorithm 6> ELSE goto <algorithm 7>
6. Insert the data dependency elimination algorithm <algorithm 5> and make parallel code

<algorithm 5>

1. Initialization
S is number of statements
Declaration of dynamic memory array DMA, DMB, DMC
s_w= 0
2. Use GCD arithmetic function call for diophantine method calculation, and derive pass=1 and S*S size 2 dimensional dependent matrix DMB.
IF (The index variable is same) THEN rename
3. Set loop index variables i, j.
They are N_1, N_2, at best.
For all DMB matrix i,j
IF ((a_{ij}⊕u_{ij}) > N_1 || (c_{ij}⊕w_{ij}) > N_2)
THEN change that element to 0
IF s_w== 0 THEN DMA = DMB, s_w = 1
4. Using DMB matix on all nonzero element
Forall i, j
IF (u_{ij} & w_{ij} == 1) THEN uniform
ELSE IF (u_{ij} & w_{ij} != 1) THEN non-uniform
ELSE complex type to doall S_i change
5. IF the same element exists,
THEN eliminate all except one
6. IF all element = 0 THEN go to 8
ELSE
to increase pass matrix production $S*S^{pass+1}$, derive sized dependent matrix
DMC = DMA * DMB
pass++
DMB = DMC
free DMC
ENDIF
7. Go to 3
8. Except changed statements, change all statements into DOALL S_i statement.

<algorithm 6>

/* Apply the inter procedure transformation first, and then apply inlining except for the inlining procedures*/
1. repeat(until all the procedure be optimized)
/* inter procedure changed property */
2. repeat(until the inter procedure be optimized)
/* except inlining inter procedure transformation */
3. apply inlining in a reverse topological order

<algorithm 7>

/* after inlining, perform changes in procedure */
1. apply inlining in a reverse topological order
2. repeat(until all the procedure be optimized)
/* interprocedure changed property */

4. 성능 분석

본 장에서는 본 논문에서 제안된 두 가지 방법(중첩 루프에서 자료 종속성 제거 방법, 프로시저 호출을 가진 루프에서 확장된 자료 종속성 제거 방법)의 성능 평가를 하였다.

먼저, 중첩 루프에서 자료 종속성 제거 방법에 대한 성능 평가를 하였다. 이 성능 평가는 가장 넓게 사용된 예제 코드[11, 13]을 이용하여 자료 종속 거리가 불변인 경우와 가변인 경우에 대해 CRAY-T3E에서 평가하였다.

자료 종속 거리가 불변인 경우에 <그림 3(a)>에 있는 코드를 가지고 제안된 방법과 unimodular, selective cycle shrinking, Hollander, and Chen&Wang 방법과 비교 분석하였다.

```
DO  I = 3, N1        DO I = 1, N1
 DO  J = 5, N2         DO J = 1, N2
   A(I,J)=B(I-3,J-5)     A(2*I+J+1,I+J+3)=···
   B(I,J)=A(I-2,J-4)           ···=A(2*J+3,I+1)
```

(a) 불변 코드 (b) 가변 코드

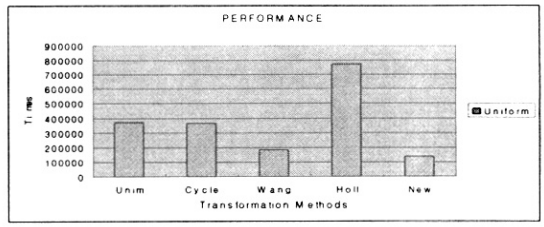
<그림3> 예제 코드

<그림 4(a)>는 <그림 3(a)>에 있는 코드를 성능 평가한 결과이다. 성능 평가는 CRAY-T3E에서 고정된 값인 N_1=20, N_2=100으로 놓고, 프로세서의 개수는 4개를 사용하였다. Unimodular 방법과 cycle shrinking 방법은 성능 평가에서 유사한 결과를 보였다. 왜냐하면 그들은 자료 종속 거리에 따라 코드들을 분할 한 후 실행하는 방법이기 때문이다. 성능이 가장 좋은 것은 본 논문에서 제안한 자료 종속성 제거 방법이다.

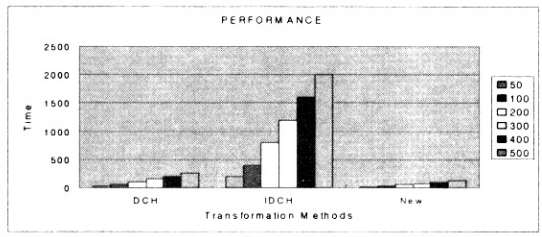
병렬 처리하는 횟수를 알기위하여, 병렬 코드로 분할 된 개수(=λ)는 tiling, interchanging, selective shrinking이 $\lambda=2N_1-4$ 이고, skewing, unimodular, Chen&Wang 방법이 $\lambda=\lceil (N_2-4)/4 \rceil$ 이며, Hollander 방법은 $\lambda=(N_1\times N_2)/2$ 이고, 자료 종속성 제거 방법은 $\lambda=\lceil 2+ N_1/10 \rceil$ 이다. 이 결과에서 보듯이 N_1과 N_2를 증가시켜도 자료 종속성 제거 방법이 가장 우수함을 보인다.

자료 종속 거리가 가변인 경우는 <그림 3(b)>의 코드를 가지고 수행하였으며, 그 결과는 <그림 4(b)>와 같다. 여기서는 가변 종속 거리에 적용할 수 있는 방법 중 DCH 방법, IDCH 방법, 그리고 제안된 방법들과 성능 평가를 하였다. 실험은 CRAY-T3E에서 N_1과 N_2를 50으로부터 500까지 증가시키면서 성능 평가를 하였고, 프로세서의 개수는 4개를 사용하였다. 이 성능 평가에서도 제안된 방법이 가장 우수하였으며 DCH 방법, IDCH 방법 순서였다. 여기서도 분할 된 병렬 코드의 개수는 DCH 방법은 $\lambda = N_1/2$ 이고, IDCH 방법이 $\lambda = N_1\times N_2/T_n$($T_n$: tile의 개수) 이고, 제안된 방법은 $\lambda = 1+\lceil Min(N_1,N_2)/4 \rceil$ 이다. 이 결과에서 보듯이 N_1과 N_2를 증가시켜도 자료 종속성 제거 방법이 가장 우수함을 보인다.

이상에서와 같이 자료 종속 거리가 불변이든 가변이든지 본 논문에서 제안된 방법인 자료 종속성 제거 방법이 실행 시간 면에서도 매우 효율적인 방법일 뿐만 아니라 자료 종속 거리가 불변이든지 가변이든지 간에 하나의 알고리즘을 가지고 적용 할 수 있어 매우 효율적인 방법이다.



(a) 불변 종속 거리



(b) 가변 종속 거리

<그림4> 예제 코드에 대한 수행결과

이제, 프로시저 호출을 가진 루프에서 확장된 자료 종속성제거 방법에 대한 성능 평가이다. 이것에 대한 성능 평가도 CRAY-T3E를 사용하였다. 프로세서의 개수는 2개, 4개, 8개, 16개, 32개로 증가하여 수행하였으며, <그림 5>에 있는 예제 코드를 사용하여 자료 종속성 거리가 불변, 가변, 불변과 가변 혼합 코드에 대해서 loop extraction 방법, loop embedding 방법, procedure cloning 방법과 제안된 방법 등을 비교 분석하였다.

확장된 자료 종속성 제거 방법은 inline 방법을 적용한 후 자료 종속성이 제거 될 때까지 반복 수행하였다. Loop extraction과 loop embedding에 위와 유사하게 적용하였으며 이는 프로시저 호출에서의 오버헤드를 감소시킬 수 있다. Procedure cloning은 순차 처리와 병렬 처리 부분으로 나누어서 처리하였다. 가변 종속 거리와 혼합 종속 거리에 대해서는 확장된 자료 종속성 제거 방법만이 적용 가능하므로 제안된 방법은 병렬화를 시켜서 적용하였고, 다른 방법들은 순차적인 실행을 하였다.

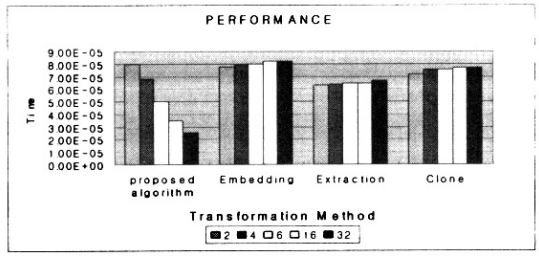
```
SUBROUTINEP                SUBROUTINEP                SUBROUTINEP
 real a(n, n)               real a(n, n)               real a(n, n)
 integer i                  integer i                  integer i
  do i = 1, 10               do i = 1, 10               do i = 1, 10
    call Q(a, i)            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

SUBROUTINQ(f,i)            SUBROUTINQ(f,i)            SUBROUTINQ(f,i)
 real f(n, n)               real f(n, n)               real f(n, n)
 integer i, j               integer i, j               integer i, j
  do j = 1, 100              do j = 1, 100              do j = 1, 100
    f(i, j) = f(i,j) +         f(i,j*5)=f(i, j*4)+...     f(i, j)=f(i, j)+...
  enddo                      enddo                      enddo
```

(a)불변 코드 (b)가변 코드 (c)혼합 코드

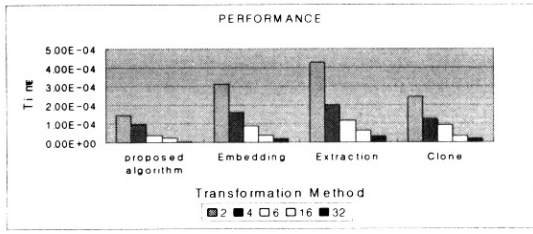
<그림5> 예제 코드

이를 CRAY-T3E에서 수행한 결과는 <그림 6>과 같다. 제안된 방법인 확장된 자료 종속성 제거 방법은 불변 종속 거리, 가변 종속 거리, 혼합 종속 거리에 대해서 다른 방법들 보다 성능이 우수하였으며, 특히 프로세서를 증가시키면 훨씬 더 좋은 성능을 보였다.

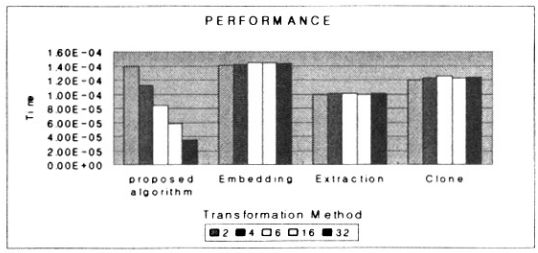
이상의 성능 평가에서 프로시저 호출을 가진 루프에서 자료 종속 거리가 불변, 가변, 혼합 면에서 제안된 방법인 확장된 자료 종속성 제거 방법이 가장 우수함을 보였다.



(a) 불변 코드



(b) 가변 코드



(c) 혼합 코드

<그림6> 비교분석결과

5. 결 론

대부분의 프로그램들은 루프 구조에서 실행 시간의 대부분을 소비한다. 이러한 이유로 루프 구조를 가진 프로그램의 실행 시간을 줄이기 위한 방법들이 많이 연구되어왔다. 이러한 연구들의 대부분은 병렬성 추출에 초점을 맞추어왔고, 그 중에서도 불변 자료 종속 거리에 적용가능 한 방법들이 많이 연구되었다. 본 논문에서는 실제 코드들이 불변 자료 종속 거리 보다는 가변 종속

거리나 혼합 종속 거리가 더 많다는 것에 착안해서 불변 종속 거리, 가변 종속 거리, 혼합 종속 거리에 모두 적용 가능한 방법을 제안하였다. 또한, 이 방법을 확장하여 프로시저 호출을 가진 프로그램에서도 적용 가능한 알고리즘을 제시하였다. 제안된 방법의 성능이 우수함을 보이기 위하여 4가지 방법(불변 종속 거리를 갖는 중첩 루프, 가변 종속 거리를 갖는 중첩 루프, 불변 종속 거리를 갖는 프로시저 호출을 가진 프로그램, 가변 종속 거리를 갖는 프로시저 호출을 가진 프로그램)에 대해서 성능 평가를 하였으며, 4가지 방법 모두에서 제안된 방법이 가장 우수함을 보였다.

또한, 기존의 방법들이 자료 종속 거리가 불변, 가변등 한 가지에만 적용 가능하지만 본 논문에서 제시한 방법은 하나의 알고리즘을 가지고 종속 거리가 불변, 가변, 혼합 등에 모두 사용 가능하기 때문에 더욱더 효과적이다.

앞으로는 성능 평가를 벤치마크 프로그램에 적용하여 실제성을 보완 할 것이다.

참 고 문 헌

[1] Allen, F., M. Burke, P. Charles, R. Cytron, and J. Ferrante(1998). An Overview of the PTRAN analysis System for Multiprocessing. Journal of Parallel and Distributed Computing 5(# 5).

[2] Banerjee, U.(1993). Loop Transformations for Restructuring Compilers: The Foundations. Kluwer Academic Publishers.

[3] Chen, Y-S and S-D Wang(1993). A Parallelizing Compilation Approach to Maximizing Parallelism within Uniform Dependence Nested Loops. Dept. of Electrical Engineering, National Taiwan University.

[4] D'Hollander, E. H.(1992). Partitioning and Labeling of Loops by Unimodular Transformations. IEEE Trans. on Parallel and Distributed Systems 3(# 4).

[5] Hall, M. W.(1991). Managing Interprocedural Optimization. Ph.D thesis, Dept. of Computer Science, Rice University.

[6] Hall, M. W., K. Kennedy, and K. S. Mckinley(1991). Interprocedural Transformations for Parallel Code Generation. Technical Report 1149-s, Dept. of Computer Science, Rice University.

[7] Kong, X., D. Klappholz, K. Psarris(1991). The I Test: An Improved Dependence Test for Automatic Parallelism and Vectorization. IEEE Trans. on Parallel and Distributed Systems 2(# 3).

[8] Li, Z., P. C. Yew, and C. Q. Zhu(1990). An Efficient Data Dependence Analysis for Parallelizing Compilers. IEEE Trans. on Parallel and Distributed Systems 1(# 1).

[9] Mckinley, K. S.(1998). A Compiler Optimization Algorithm for Shared-Memory Multiprocessors. IEEE Trans. on Parallel and Distributed Systems 9(# 8).

[10] Polychronopoulos, C. D.(1990). Parallel Programming Issues. CSRD Report No. 1004, Univ. of Illinois at Urbana-Champaign.

[11] Punyamurtula, S., and V. Chaudhary(1996). Compile-Time Partitioning of Nested Loop Iteration Spaces with Non-uniform Dependences. Journal of Parallel Algorithm and Architecture.

[12] Ramanujam, J.(1995). Beyond Unimodular Transformations. The Journal of Supercomputing (# 9).

[13] Tzen, T. H. and L. M. Ni(1993). Dependence Uniformization: A Loop Parallelization Technique. IEEE Trans. on Parallel and Distributed Systems.

[14] Wolfe, M. E.(1982). Optimizing Supercompiler for Supercomputing. Ph. D. Thesis, CSRD Report No. 82-1105, Univ. of Illinois at Urbana-Champaign.

[15] Wolfe, M. E., and C. W. Tseng(1992). The Power Test for Data Dependence. IEEE Trans. on Parallel and Distributed Systems 3(# 5).

[16] Wolfe, M. J.(1996). High Performance compiler for Parallel Computing. Oregon Graduate Institute of Science & Technology.

[17] Zima, H., and B. Chapman(1991).

Supercompilers for Parallel and Vector Computers. Addison-Wesley.

박 두 순

1981 고려대학교 수학과 (이학사)
1983 충남대학교 대학원 (전산학전공, 이학석사)
1988 고려대학교 대학원(전산학전공, 이학박사)
1985~현재 순천향대학교 정보기술공학부 교수
2004~현재 미국 콜로라도대학 객원교수
2004~현재 한국멀티미디어학회 이사
2002~2003 순천향대학교 공과대학 학장
관심분야: 병렬 처리, 데이터 마이닝
E-Mail: parkds@sch.ac.kr