외부루프 펼침에 의한 다중첨자 루프의 종속성 제거 기법

박상일*, 박월선*, 박현호** 윤성대*
*부경대학교 전자계산학과
**부경대학교 전산정보학과
e-mail:sangil94@dol.pknu.ac.kr

A Data Dependency Elimination Method for Multidimensional Subscript Loop by Outer Loop Unrolling

Sang-Il Park*, Weol-Seon Park*, Hyun-Ho Park**, Sung-Dae Youn*
*Dept of Computer Science, Pukyong National University
**Dept of Computer and Information, Pukyong National University

요 약

본 논문에서는 외부 루프를 펼침으로서 불변 종속거리를 가지는 다중 첨자 루프에서의 병렬화를 이룰 수 있는 새로운 알고리즘을 제시한다. 루프는 프로그램의 수행 시간중 많은 부분을 차지하고, 병렬성 추출의 기본이 되는 구조이다. 루프에서 병렬성을 추출하는 기존의 연구는 종속성이 단일 첨자 또는 복수 첨자에 영향을 받는 경우에만 한정되었다. 다중 첨자를 가지는 루프는 이중 또는 그 이상의 첨자 때문에 기존의 방법을 이용해서 루프의 종속성을 제거하는데 필요한 종속거리를 결정할 수 없다. 그러므로 본 논문에서는 종속거리를 측정하기 위한 새로운 기법을 제안하고, 제안된 알고리즘을 모의 실험에 의해 타당성을 확인한다.

1. 서론

하드웨어의 비약적인 발전에 힘입어 컴퓨터의 속도가 현저하게 빨라짐에 따라 순차적인 프로그램을 보다 효율적으로 처리하기 위한 병렬 컴파일러에 대한 연구가 많이 이루어져 왔다. 일반적인 순차 프로그램에서 가장 많은 시간을 차지하고, 병렬성이 많은 구조는 루프 구조이고, 루프를 병렬로 처리하기 위한 연구가 많이 이루어져 왔다[1,2]. 또한 병렬화 컴파일러중에서 가장 기본적이고, 많이 활용할 수 있는 부분은 순차프로그램에 있는 루프로부터 병렬성을 추출하여 병렬코드로 변환해주는 루프 재구조화 방법이다. 이 방법은 병렬처리 시스템의 속도를 향상시키는데 상당한 효과가 있기 때문이다[3].

루프는 중첩도에 따라 단일 루프와 다중 루프로 나눌 수 있고, 종속거리에 따라 불변 종속과 가변 종속으로 나눌 수 있다. 불변인 경우는 tiling, interchanging, skewing 등이 있고[1], 가변 종속 거리는 DCH[4], IDCH[5], CDCH[6]등이 있다. 또한 Coset을 이용한 루프 병렬화 기술에 대한 연구[7]와 DMLCS행렬을 이용한 불변과 가변 종속거리에 적용 가능한 알고리즘[3]이 연구되었다. 이들 연구에서 사용된 루프를 살펴보면 종속성이 존재하는 변수가 존재하고 거기에 실제로 영향을 주는 첨자들이 있는데, 외부루프와 내부루프가 섞여 있는 경우, 즉 다차원 첨자에 의해 영향을 받는 경우에 대한 연구는 없었다. 왜냐하면 다차원 첨자에 의해 색인되는 데이터 배열의 경우 종속성을 컴파일 시에 결정하기가 매우 어렵기 때문이다[1].

본 논문에서는 불변 종속 거리를 가지면서, 다차원 첨자에 의해 색인되는 데이터 배열을 가지는 루프에 적용 가능한 종속성 제거 알고리즘을 제안한다. 다차원 첨자에 의해 색인되는 데이터 배열을 가지는 루프에서 나올 수 있는 모든 경우를 4가지로 나누어 제안한 알고리즘에 적용해 보고, 그 타당성과 성능을 평가해 본다.

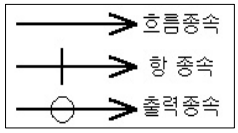
본 논문의 구성은 다음과 같다. 2장에서는 종속성에 대해서 알아보고, 3장에서는 제안한 알고리즘을 보이고, 4장에서 실험을 통한 결과를 분석한 후 끝으로 5장에서 결론을 맺는다.

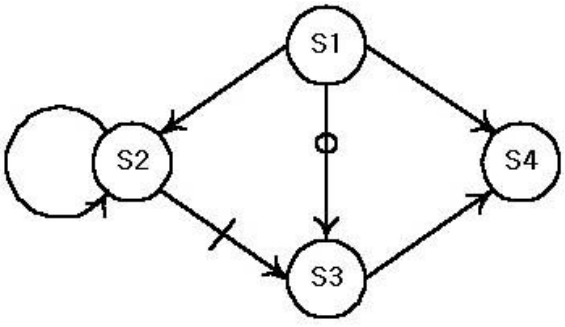
2. 종속성

자료 종속성에 대해 간단히 살펴보면 흐름 종속, 항 종속, 출력 종속으로 나눌 수 있다[1,3,7]. 문장 S1에서 S2로 실행되는 순서가 존재하고, S1의 출력 중 최소한 1개가 S2의 입력으로 사용될 때, S2에서 S1으로의 흐름 종속이 존재한다. S2는 프로그램 순서상 S1의 뒤에 나오고 S2의 출력이 S1의 입력과 중복될 때, S2에서 S1으로 항 종속성이 존재한다. 두 명령어가 같은 변수에 결과 값을 출력한다면 출력 종속이 존재한다고 말할 수 있다. 그림 1은 흐름 종속과 항 종속, 출력 종속에 대하여 보여주고 있다.

```
S1 : Load R1, A
S2 : Add R2, R1
S3 : Move R1, R3
S4 : Store B, R1
```

(a)예제 프로그램





(b) 종속 그래프

(그림 1) 자료 종속성

자료 종속성을 분석하는 방법에는 여러 가지 종류가 있다. GCD(Greatest Common Divisor)테스트[1,3,7]는 루프의 영역과 상관없이 종속 방정식이 정수해를 가지는 지의 여부를 판단하는 방법이다. 그 외에도 Seperability테스트[3], Power테스트[8], I 테스트[9] 등이 있다. 자료 종속거리는 종속성이 존재하는 두 문장 사이의 거리로써 종속성을 제거하는데 반드시 필요한 정보이다.

3. 제안한 병렬화 알고리즘

다중 첨자를 가지는 루프는 이중 또는 그 이상의 첨자 때문에 기존의 방법을 이용해서 루프의 종속성을 제거하는데 필요한 종속 거리를 결정할 수 없다. 그러므로 이를 해결하기 위해 불변 종속거리를 가지는 루프에 대해 외부 루프를 분리하여 펼치는 기법으로 다중 첨자를 하나의 첨자로 축약하여 병렬로 처리하는 기법을 제안한다.

3.1 외부 루프의 종속성 제거 알고리즘

외부 루프를 분리하기 위해 최 외곽 루프의 첨자를 포함하는 항의 상수들에 대한 최대공약수를 구한다. 최대공약수를 구하는 목적은 해당 항을 최소화하기 위해서이다. ‘GCD(a1, a2, b1, b2) = d’라고 하면, 최 외곽 루프의 첨자와 최대 공약수 값 d를 원래 루프에서 추출한다

```
for i = 1, M1
  for j = 1, M2
S1:  A(i+j+2, i+j+4) = … ;
S2:  …   = A(i+j+6, i+j+4);
  end for
end for
```

(a) i+2를 추출하기 전

```
 for j = 1, M2
S1:  A(j, j+2) = … ;
S2:  …   = A(j+3, j+1);
 end for
```

(b) i+2를 추출한 후

```
 for i = 1, M1
P1:      temp[i] = i+2;
 end for
 barrier
 for k = 1, M1
    for j = 1, M2
S1:     A(j+temp[k], j+2+temp[k]) = … ;
S2:    … = A(j+3+temp[k], j+1+temp[k]);
 end for
```

(c) i+2를 보정한 루프

(그림 2) 예제 루프

그림 1에서 GCD(2, 4, 6) = 2이고 외곽 첨자인 i와 2를 (a)에서 추출하면 (b)와 같이 만들 수 있다. 추출된 i+2를 처리하기 위해 (b)를 변형하면 (c)와 같다. (c)의 변형된 루프를 살펴보면 P1의 문장은

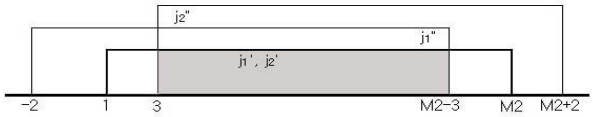
추출된 i+2을 처리하기 위해 새로 생성된 문장이고, 병렬로 처리 될 수 있다. 문장 S1과 S2는 기존의 문장에서 i+1값이 추출된 후에 변형된 문장으로 외부 루프의 첨자인 k에 대해 병렬로 처리할 수 있다. (c)의 예에서 보인 것과 같이 i루프를 추출함으로써 i에 대한 종속성을 제거할 수 있고, j에 대한 종속성을 찾아서 제거하면 병렬로 처리할 수 있다.

3.2 내부 루프에서의 종속성 제거를 위한 알고리즘

외부 루프인 i에 대한 종속은 외부 루프를 추출하는 기법으로 제거 할 수 있다는 것을 3.1절에서 보였다. 다중 첨자를 갖는 루프를 병렬로 처리하기 위해 남은 것은 내부 루프인 j루프에 의한 종속성을 제거하는 것이다. 이를 위해 S1문장에 영향을 주는 첨자 j에 대해 첫 번째 항을 j_1', 두 번째 항을 j_2'이라 두고 S2문장에 영향을 주는 첨자 j에 대해서도 동일하게 j_1'', j_2''이라고 하자. 이 때 각 항에 대한 종속 방정식을 구하면 다음과 같다.

$$\begin{cases} \text{첫 번째 항의 종속방정식 :} \\ \quad j_1' + temp[k] = j_1'' + 3 + temp[k] \quad (1) \\ \text{두 번째 항의 종속방정식 :} \\ \quad j_2' + 3 + temp[k] = j_2'' + 1 + temp[k] \quad (2) \end{cases}$$

그림 2의 (a)는 2차원 배열을 갖는 루프에 관한 것으로 이 경우 j_1'와 j_2'는 $j_1'=j_2'$인 j값을 갖고, j_1'', j_2''의 경우에 대해서도 역시 같은 j"의 값을 갖는다. 그러므로 식(1)과 (2)를 이용하여 종속성이 발생하는 j_1', j_2'과 j_1'', j_2''의 값을 구할 수 있다. 위의 예에서 계산하면 j"+6 = j"+1이 되어 j"값을 구할 수가 없다. 여기서 temp[k]는 k에 대해 병렬로 처리하므로 상수값처럼 인식되어 종속성에 영향을 주지 않는다. 이 경우외에도 종속성을 발생시키는 j값이 있는지 찾아보자. 식(1)에서 j_1'={1,…,M2}의 범위 값을 가질 때 j_1''의 범위를 생각하면 j_1''={-2,…,M2-3}의 값을 가진다. 동일한 방법을 식(2)에 적용하면 j_2'={1,…,M2}일 때 j_2''={3,…,M2+2}의 값을 가진다. 이때 j_1', j_2'이 같은 범위를 가질 때 j_1'', j_2''이 다른 범위를 가짐을 알 수 있고, 종속성이 발생하는 j',j"를 구하기 위해 공통 범위를 구하면 j'={1,…,M1}, j"={3,…,M2-3}사이의 값을 구할 수 있다. 그림 3은 위에서 설명한 내부루프 첨자 j에 의해서 종속성이 발생할 수 있는 j'과 j"의 범위를 나타내고 있다. 즉 j"={3,…,M2-3}의 범위에 따른 j_1'과 j_2'을 구해서 j_1' = j_2'인 경우를 조사하면 내부루프의 첨자 j에 의해서 종속성이 발생할 수 있는 범위를 구할 수 있다.



(그림 3) j_1'과 j_2'가 같은 범위를 가질 때 j_1'', j_2''의 범위

2차원 배열을 갖는 경우 첫 항과 두 번째 항에 j'이 공통으로 영향을 주므로 두 식을 서로 대입해서 값을 구한 경우 이외에는 공통 범위가 존재하더라도 종속성이 존재하는 j값을 찾을 수 없다. 1차원 배열을 갖는 경우는 기존의 첨자가 단일일 때 종속성을 제거하는 기법으로 내부루프인 j에 대한 종속성을 제거할 수 있다.

3.3 알고리즘

다중 첨자를 갖는 루프에서 발생할 수 있는 모든 경우는 다음과 같다.

(1) Case 1 : 그림 4의 (a)와 같이 루프가 1차원 배열을 갖는 경우
(2) Case 2 : 그림 4의 (b)와 같이 루프가 2차원 배열을 갖는 경우
(3) Case 3 : 그림 4의 (c)와 같이 2차원 배열이면서 공통범위가 존재하는 경우
(대입했을 때 j', j"의 값을 구할 수 없는 경우)
(4) Case 4 : 그림 4의 (d)와 같이 2차원 배열이면서 공통범위가 존재하는 경우
(대입했을 때 j', j"의 값을 구할 수 있는 경우)

```
for i = 1, N1
  for j = 1, N2
S1:  A(i+j+3) = … ;
S2:       … = A(i-j+1);
  end for
end for
```

(a)Case 1

```
for i = 1, N1
  for j = 1, N2
S1: A(i+j+1,i+j+3) = … ;
S2: …   = A(i+j+4,i-j+2);
  end for
end for
```

(b)Case 2

```
for i = 1, N1
 for j = 1, N2
  A(i+j+1, i+j+3) = … ;
  …   = A(i+j+4, i+j+2);
 end for
end for
```

(c)Case 3

```
for i = 1, N1
 for j = 1, N2
  A(i+j+1, i+j+3) = … ;
  …   = A(i+j+5, i-j-1);
 end for
end for
```

(d)Case 4

(그림 4) 다중 첨자 루프의 4가지 경우

Source program :

```
for i = N1, M1
 for j = N2, M2
  S1 :  A(i+j+a1, i+j+a2) = ···      ;
                          ···        ;
  S2 :      ···   = A(i+j+b1, i-j+b2);
 end for
end for
```

(그림 5) 소스 프로그램

그림 5는 제안한 알고리즘에 적용하기 위한 소스 프로그램이다. 그림 6은 제안한 알고리즘으로, 외부 루프 i가 미치는 영향을 제거하기 위해 첨자 i를 추출한다. 이때 각 상수에 대한 GCD 값을 구하는 이유는 추출한 후에 남는 항을 최소화하기 위함이다. 내부루프를 병렬로 처리하기 위해 단일 첨자인 j에 의해서 발생하는 종속성을 조사한다. Case1의 경우 기존의 연구에서 다루어진 루프의 형태이므로 쉽게 처리가 가능하다. 나머지 3가지 경우는 하나의 루프 첨자에 대한 종속성을 테스트하는 것으로, ①' = ③', ②' = ④'를 동시에 만족하는 j 값을 찾아야 한다. 그 결과 3.2절에서 살펴본 공통범위를 구해서 종속성 여부를 살펴보는 방식을 쓰지 않고도 R', R" 값이 존재하는 경우만 종속성이 발생한다는 것을 알 수 있다. 외부 루프와 내부 루프에 대한 처리가 끝나면 비로소 병렬 코드가 생성될 수 있다.

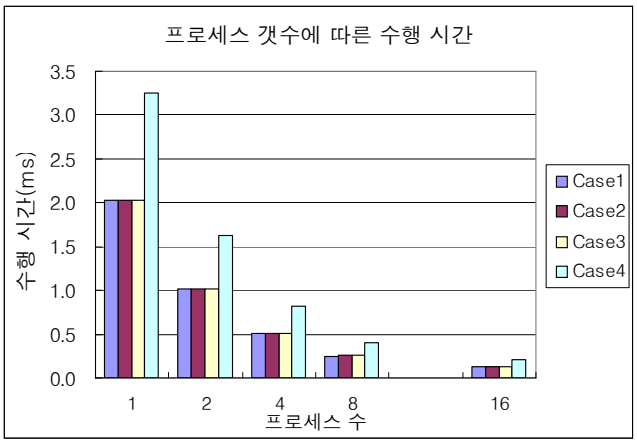
```
Procedure parallel_algorithm()
{
/* 외부루프 i에 대한 종속성 제거 */
/* 그림 4의 소스프로그램에서 종속 방정식을 추출
   하기 위해 첨자식을 추출 */
    S1' = i'+j'+a1   /* S1의 첫째 항 - ① */
    S1" = i'+j'+a2   /* S1의 둘째 항 - ② */
    D1' = i"+j"+b1   /* S2의 첫째 항 - ③ */
    D2" = i"-j"+b2   /* S2의 둘째 항 - ④ */
   GCD(a1, a2, b1, b2) = d
/* i+d를 추출 */
    S1' = j'+(a1-d)  /* S1의 첫째 항 - ①' */
    S2" = j'+(a2-d)  /* S1의 둘째 항 - ②' */
    D1' = j"+(b1-d)  /* S2의 첫째 항 - ③' */
    D2" = -j"+(b2-d) /* S2의 둘째 항 - ④' */
/* j에 대한 종속성 여부 test */
   j'+(a1-d)=j"+(b1-d)    /* ①' = ③' */
   j'+(a2-d)=-j"+(b21-d) /* ②' = ④' */
/*  위 두 식을 이용하여 j'과 j"을 구한다.
  이때 j'의 값 : R', j"의 값 : R"이라 하면 */
   if (R'={} || R"={})
        j에 대한 종속성이 존재하지 않음.
   else if(N1≤R'≤M1 && N2≤R"≤M2){
        R'과 R"에 대한 종속성을 해결.}
   else j에 대한 종속성이 존재하지 않음.
}
```

```
/* 루프의 재구성 결과 */

  for i = N1, M1
P1:     temp[i] = i+2;
  end for

  barrier

  for k = N1, M1
     for j = N21, M2
S1:    A(j+temp[k], j+2+temp[k]) = ··· ;
S2:   ··· = A(j+3+temp[k], j+1+temp[k]);
     end for
  end for
```

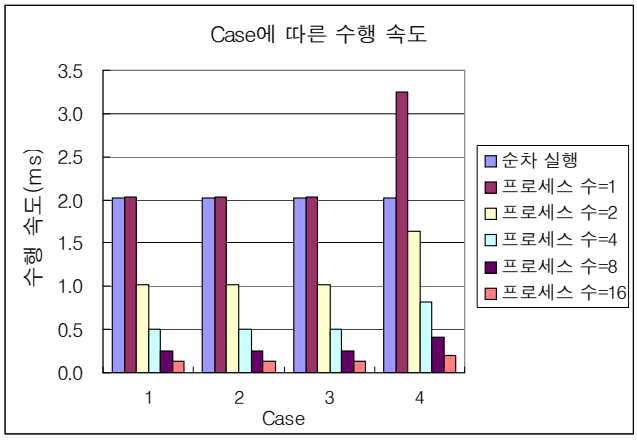
(그림 6) 다중첨자를 갖는 루프의 병렬화 알고리즘

4. 실험 및 분석

제안한 알고리즘을 그림 4의 4가지 경우(Case1, Case2, Case3, Case4)에 적용하여 성능을 평가한다. N1 = N2 = 100으로 하여 실험을 하였다. Case1은 다중 첨자를 가지는 1차원 배열 루프의 예이고, Case2는 2차원 배열을 가지는 루프의 예이다. Case3은 Case2에서 내부루프의 첨자인 j에 대한 공통 범위가 존재하면서 실제로는 종속을 생성하는 j값이 존재하지 않는 경우이고, Case4의 경우는 종속을 생성하는 j값 쌍이 존재하는 경우에 대한 예이다. 그림 5의 (a)는 그림 4의 (a), (b), (c), (d)를 제안한 알고리즘에 적용한 후 순차적인 실행을 했을 때 수행 속도와 프로세서의 갯수를 2, 4, 8, 16개로 변화했을 때의 속도를 나타낸 것이고, (b)는 각 Case별 수행 속도를 비교 해놓은 것이다.



(a) 프로세스 갯수에 따른 수행속도



(b) Case에 따른 수행 속도

(그림 7) 실험 결과

그림 7의 결과에서 알 수 있듯이 프로세스 갯수가 하나일 때, 즉 순차적인 실행 보다 병렬로 처리할 때가 보다 나은 수행 속도를 가짐을 알 수 있고, Case1, Case2, Case3의 예는 종속을 생성하는 내부 루프 j가 존재하지 않거나 쉽게 제거 할 수 있기 때문에 속도차가 거의 없음을 알 수 있다. Case4의 경우는 종속을 유발하는 한 쌍의 j값이 존재하기 때문에 다른 예보다 수행 속도가 떨어짐을 알 수 있다. 시간 복잡도 측면에서 살펴보면 순차적인 실행의 경우 $O(N1 \cdot N2)$이고, 프로세스의 갯수($nproc$)에 따라 $O(\frac{N1(1+N2)}{nproc})$의 시간 복잡도가 됨을 알 수 있다.

6. 결론 및 향후과제

다중 첨자를 가지는 루프에 대한 병렬화는 종속거리를 측정하고 종속성을 제거함에 있어서 여러 개의 첨자가 영향을 줌에 따라 제거가 어려웠다. 불변 종속거리를 갖는 다중 첨자 루프에서, 외부루프 펼침에 의해 영향을 주는 첨자의 갯수를 줄여가서 결국 하나의 첨자만을 남겨 종속성을 제거하는 알고리즘을 제안했다. 또한 발생 가능한 4가지 경우를 실험을 통해 살펴보았다. 실험 결과 제안한 알고리즘에 의해 병렬로 처리된 경우 순차적인 처리에 비해 수행 속도가 높음을 보였고, 이는 제안한 알고리즘에 적용한 후 생성된 코드를 병렬로 수행시 순차적인 경우 보다 우수하다는 것을 나타내며, 또한 알고리즘의 타당성을 보이고 있다. 그러나 다중 첨자 루프가 각 첨자에 대해 불변 종속 거리를 가질 때만 적용이 가능하다는 제약을 가진다.

향후 연구과제로는 각 첨자에 대해 불변과 가변 종속거리를 가지는 경우에 대해서도 적용이 가능한 알고리즘의 제안 및 실제 구현하는 것이다.

참고문헌

[1] Kai Hwang, Advanced Computer Architecture, 3rd Ed., McGraw-Hill, 1993

[2] Kuck, D. J., A. H. Sameh, R. Cytron, A. V. Veidenbaum et al., "The effect of program restructuring, algorithm chadnges, and architecture choice on program performance", Proc. Int. Conf. Parallel Processing '84, pp. 129-138, 1984

[3] 송월봉, 박두순, “최대 병렬성 추출을 위한 자료 종속성 제거 알고리즘”, Journal of KISS. Vol. 26, No. 1, January 1999

[4] Tzen, T. H., and L. M. Ni, "Dependence Uniformization : A Loop Parallelzation Technique", IEEE Trans. on Parallel and Distributed System, Vol. 4, No. 5, May, 1993

[5] Punyamurtula, V. Chaudhary, J. Ju. and S.Roy, "Compile time partitioning of Nested Loop Iteration space with Non-uniform Dependencies", In Journal of Parallel Algorithm and Architecture, 1996

[6] Ju.,J. and V. Chaudhary, "Unique Sets Orinted Partitioning of Nested Loops with Non-uniform Dependencies", 1996

[7] Teruaki, K., Kazuki J, "A Loop Parallelization Technique for Linear Dependence Vector", Proceedings of the IFIP WG10.3 Working Conference on Parallel Architectures & Compilation Techniques, PACT '95, 1995

[8] Wolfe, M. E., C. W. Tseng, "The Power Test for Data Dependence", IEEE Trans. on Parallel and Distributed Systems, Vol. 5, No. 5, Oct 1988

[9] Kong, X., D. Klappholz, K. Psarris, "The I Text: An Improved Dependence Test for Automatic Parallel and Vectorization", IEEE Trans. on Parallel and Distributed System. Vol. 2, No. 3, July 1991