다중첨자루프에서 변수를 이용한 종속성 제거 기법

박상일(朴相一)*, 박현호(朴賢鎬)**, 김형욱(金炯昱)*, 윤성대(尹成大)*
*부경대학교 전자계산학과
**부경대학교 전산정보학과
전화 : (051) 620-6398 / 팩스 : (051) 620-6398

A Data dependency Elimination method in multidimensional Subscript Loop Using a variable

Sang Il Park*, Hyun-Ho Park**, Hyung-Wook Kim*, Sung-Dae Youn*
*Dept of Computer Science, Pukyong National University
**Dept of Computer and Information, Pukyong National University
E-mail : sangil94@dol.pknu.ac.kr

Abstract

In this paper, we propose a new method to parallelize multidimensional subscript loop with non-uniform distance. A loop comprise most of the computation in a program and the most important source of parallelism. Multidimensional subscript within a loop difficult to determine for distance to be required loop dependence Elimination. Therefore we propose new methods that is three algorithm to search subscript.

I. 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 구조이며, 루프를 병렬로 처리하기 위한 연구가 활발히 이루어져 왔다[3]. 또한 병렬화 컴파일러 중에서 가장 기본적이고, 많이 활용할 수 있는 부분은 순차 프로그램에 있는 루프로부터 병렬성을 추출하여 병렬 코드로 변환해주는 루프 재구조화 방법이다. 이 방법은 병렬처리 시스템의 속도를 향상시키는데 매우 중요하다[6]. 기존의 연구 대상이 된 루프를 분류해 보면, 루프의 중첩도에 따라 단일 루프와 다중 루프로 나눌 수 있고, 종속거리에 따라 불변 종속 거리와 가변 종속 거리로 나눌 수 있다. 불변 종속 거리를 가지는 경우는 tiling, interchanging, skewing[3] 등이 연구 되었고, 가변 종속 거리는 DCH[6], IDCH[5], CDCH[1]등이 연구되었다. 또한 Coset을 이용한 루프 병렬화 기술에 대한 연구[5]와, DMLCS행렬을 이용한 불변과 가변 종속거리에 적용 가능한 알고리즘[6]이 연구되었다. 다차원 첨자에 의해 색인되는 데이터 배열의 경우 종속성이 컴파일 시에 결정되기가 매우 어렵다[3]. 그러므로 본 논문에서는 가변 종속 거리를 가지면서, 다차원 첨자에 의해 색인되는 데이터 배열을 가지는 루프에서 종속성이 존재하는 문장의 첨자를 이용하여 종속성이 발생할 때의 첨자를 구하는 방법을 제안하고, 이를 이용하여 종속성 제거 알고리즘을 제안한다.

II. 본론

가변 종속거리를 가지는 다중 첨자 루프는 루프의 외부첨자와 내부첨자가 종속거리에 영향을 주기 때문에 종속거리를 측정하기 어렵고, 다수의 종속성이 발생한다. 그러므로 이를 해결하기 위해 종속성이 존재하는 문장에 대해서 종속방정식을 세우고 종속 방정식

에 루프의 범위를 적용함으로 루프의 범위 내에서 발생하는 종속성에 대한 첨자 값을 구해낸다.

2.1 종속성이 발생하는 첨자값을 구하는 방법

```
for i = 1, N1
  for j = 1, N2
      ...
    A(2i+j+1) = ...
    ...  = A(i+2j+3)
      ...
```

(그림 1) 예제 루프

(그림 1)의 예에서 변수 A에 대한 종속 방정식을 구하면 다음과 같다.

$$2i' + j' + 1 = i'' + 2j'' + 3 \quad (1)$$

식(1)에서 좌변을 첨자 i'과 j'로만 표시하면 다음과 같다.

$$2i' + j' = i'' + 2j'' + 2 \quad (2)$$

식(2)의 $2i'$과 j'은 다음과 같은 값을 가질 수 있다.

$$2i'' = \{0, 2, i'', 2i'', i''+2, 2i''+2, i''+2i'', i''+2j''+2\}$$
$$j'' = \{i''+2j''+2, i''+2i'', 2i''+2, i''+2, 2i'', i'', 2, 0\}$$

여기서 $2i' = 0$이면 $j' = i'' + 2j'' + 2$을 만족해야 하고, $2i' = 2$이면 $j' = i'' + 2j''$을 만족해야 한다. 나머지 항에 대해서도 동일하게 적용된다.
위의 식을 다시 표현하면 아래와 같다.

$$① \begin{cases} 2i'' = 0 \\ j'' = i'' + 2j'' + 2 \end{cases} \quad ② \begin{cases} 2i'' = i'' \\ j'' = 2j'' + 2 \end{cases}$$

$$③ \begin{cases} 2i'' = 2j'' \\ j'' = i'' + 2 \end{cases} \quad ④ \begin{cases} 2i'' = 2 \\ j'' = i'' + 2j'' \end{cases}$$

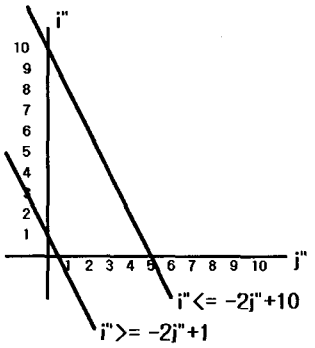
$$⑤ \begin{cases} 2i'' = i'' + 2 \\ j'' = 2j'' \end{cases} \quad ⑥ \begin{cases} 2i'' = 2j'' + 2 \\ j'' = i'' \end{cases}$$

$$⑦ \begin{cases} 2i'' = i'' + 2j'' \\ j'' = 2 \end{cases} \quad ⑧ \begin{cases} 2i'' = i'' + 2j'' + 2 \\ j'' = 0 \end{cases}$$

위의 식에서 ①,④,⑦,⑧은 i' 또는 j'중 한 값이 결정되어 있는 식이고 ②,⑤식은 i'은 i''에 영향을 받고, j'은 j''첨자에 영향을 받는다. ③,⑥식에서는 첨자 i'이 j''에 영향을 받고, j'은 i''에 영향을 받는 식이다. ①,④,⑦,⑧ : Case1, ②,⑤ : Case2, ③,⑥ : Case3으로 두고 3가지 경우에 대해 종속성이 발생하는 첨자식 (i', j', i'', j'') 값을 구하는 방법을 제안한다.

2.1.1 Case1의 알고리즘

Case1의 경우 하나의 첨자값이 정해져 있으므로 나머지 첨자에 대한 값만을 구하면, Case1에 의해서 발생하는 종속성에 대한 첨자 값을 구할 수 있고 제거할 수 있다. 예로서 ①,⑧식은 루프의 범위($1 \le i \le N_1$, $1 \le j \le N_2$)의 시작값 "1"에 의해 고려하지 않아도 된다. 식④를 보면 i'의 값은 1로 정해져 있고, j'이 가질 수 있는 범위를 구하면 j'은 $1 \le j' \le N_2$의 루프범위를 가지므로 $1 \le i'' + 2j'' \le N_2$을 구할 수 있다. 여기서 N_2를 10이라고 하면, 이 식은 $1 \le i'' + 2j''$, $i'' + 2j'' \le 10$ 식으로 표현 할 수 있다.



(그림 2) j'을 결정하기 위한 i'', j''의 그래프

위의 두 식을 그래프로 표현하면 루프의 범위와 두 식에 의한 공통 부분이 존재함을 알 수 있는데 이 부분의 정수 값이 종속성이 발생하는 i'', j''의 값이 된다. 그래프 내의 i''과 j''로 j'을 구하면 Case1에서 발생하는 종속성의 모든 첨자를 구할 수 있다. Case1의 알고리즘은 다음과 같다.

Case1의 알고리즘

```
for i = S1, E1
  for j = S2, E2        /*종속성이 존재하는 문장*/
    {A = ...;
     ... = A';}
```

$$① \begin{cases} ai' = b \\ cj' = xi'' + yj'' + e \end{cases} \quad ⑧ \begin{cases} ci' = xi'' + yj'' + e \\ aj' = b \end{cases}$$

$$④ \begin{cases} a\,i' = b \\ c\,j' = x\,i'' + y\,j'' \end{cases}, \quad ⑦ \begin{cases} c\,i' = x\,i'' + y\,j'' \\ a\,j' = b \end{cases}$$

```
procedure parallel_algorithm( )
{
```

initialize S_1' , E_1' , S_2' , E_2' ;

/* $S_1' = c*S_1$; $E_1' = c*E_1$;
$S_2' = c*S_2$; $E_2' = c*E_2$; */

if $(S \le \frac{b}{a} \le E)$ *then* {

if $(E_1' > E_1)$ *then* $E_1' = E_1$;

else if $(E_2' > E_2)$ *then* $E_2' = E_2 - i$;

else $E_2' = E_2' - i$;

for $i = \frac{S_1'}{x}, \frac{E_1'}{x}$
for $j = \frac{S_2'}{y}, \frac{E_2'}{y}$
{$A' = \cdots$
$\cdots = A'$}

}

}

2.1.2 Case2의 알고리즘

Case2는 동일한 첨자 i', i'' 와 j', j'' 사이에 관계식이 성립한다. 예로서 식②의 경우를 구해보면 다음과 같은 결과를 가진다.

i′	i″	j′	j″
1	2	4	1
2	4	6	2
3	6	8	3
4	8	10	4
5	10		

첨자 $i' = 1$일 때 i'' 은 2를, $i' = 2$이면 i'' 은 반드시 4 값을 가져야 한다. 이런 특성은 첨자 j의 경우에도 적용되는데, Case2는 루프를 i'', j'' 으로 변형하여 문장에 적용함으로써 종속성을 제거한다. Case2의 알고리즘은 다음과 같다.

Case2의 알고리즘

for $i = S_1, E_1, C_1$
for $j = S_2, E_2, C_2$ /*종속성이 존재하는 문장*/
{$A = \ldots$;
$\ldots = A'$;}

② $\begin{cases} a\,i' = b\,i'' \\ c\,j' = d\,j'' + e \end{cases}$, ⑤ $\begin{cases} a\,i' = b\,i'' + f \\ c\,j' = d\,j' \end{cases}$

procedure parallel_algorithm( )
{

initialize S_1' , E_1' , S_2' , E_2' ;

/* $S_1' = \lceil \frac{a*S_1 - f}{b} \rceil$;, $E_1' = \lceil \frac{a*E_1 - f}{b} \rceil$;
$C_1' = \lceil \frac{a*C_1}{b} \rceil$; , $S_2' = \lceil \frac{c*S_2 - e}{d} \rceil$;
$E_2' = \lceil \frac{c*E_2 - e}{d} \rceil$; $C_2' = \lceil \frac{c*C_2}{d} \rceil$; */

if $(E_1' > E_1)$ *then* $E_1' = E_1$;
else if $(E_2' > E_2)$ *then* $E_2' = E_2$;
else if $(C_1' < 1)$ *then* $C_1' = C_1$
else if $(C_2' < 1)$ *then* $C_2' = C_2$
for $i = S_1', E_1', C_1'$
for $j = S_2', E_2', C_2'$
{$A' = \cdots$;
$\cdots = A'$;}

}

2.1.3 Case3의 알고리즘

Case3은 i' 첨자가 j'' 첨자와 관계가 있고, j' 은 i'' 과 관계가 성립되는 경우이다. 각 식 i'과 j'' 사이 또는 j'과 i'' 사이에 정수의 값이 존재하지 않으면 그 식에 의해서 발생하는 종속성이 없음을 뜻한다. 정수 값이 존재한다면 Case1의 방법과 동일하게 첨자에 대한 루프의 범위를 이용하여 종속성이 발생하는 첨자 i', j', i'', j'' 값을 구할 수 있다. Case3이 알고리즘은 다음과 같다.

Case3의 알고리즘

for $i = S_1, E_1, C_1$
for $j = S_2, E_2, C_2$ /*종속성이 존재하는 문장*/
{$A() = \ldots$;
$\ldots = A()$;}

③ $\begin{cases} a\,i' = b\,j'' \\ c\,j' = d\,i'' + e \end{cases}$, ⑥ $\begin{cases} a\,i' = b\,j'' + f \\ c\,j' = d\,i'' \end{cases}$

procedure parallel_algorithm( )
{

initialize S_1' , E_1' , S_2' , E_2' ;

/*
$S_1' = \lceil \frac{a*S_1 - f}{b} \rceil$;, $E_1' = \lceil \frac{a*E_1 - f}{b} \rceil$;
$C_1' = \lceil \frac{a*C_1}{b} \rceil$; , $S_2' = \lceil \frac{c*S_2 - e}{d} \rceil$;
$E_2' = \lceil \frac{c*E_2 - e}{d} \rceil$; , $C_2' = \lceil \frac{c*C_2}{d} \rceil$,
*/

$if\ (E_1' > E_1)\ then\ E_1' = E_1;$

$else\ if\ (E_2' > E_2)\ then\ E_2' = E_2;$

$else\ if\ (C_1' < 1)\ then\ C_1' = C_1$

$else\ if\ (C_2' < 1)\ then\ C_2' = C_2$

for $j = S_1', E_1', C_1'$

for $i = S_2', E_2', C_2'$

$\{A' = \cdots;$

$\cdots = A';\}$

}

2.2 전체적인 알고리즘

전체적인 알고리즘은 아래와 같다.

① 종속방정식을 이용해서 종속성이 발생하는 첨자식을 만들고, Case1, Case2, Case3로 분류.

② 각 Case의 첨자값에 대해 루프의 범위를 적용, 발생 불가능한 값을 포함하는 식을 제거.

③ Case1에서 루프를 i'', j''으로 변경, 병렬코드를 생성.

④ Case2에서, i', i''과 j', j''의 관계를 이용하여 루프를 변환하고, 병렬코드를 생성.

⑤ Case3에서, i', j''과 j', i''의 관계를 이용하여 루프를 변환하고, 병렬코드를 생성

⑥ ③,④,⑤에 포함되지 않는 루프범위를 코드에 첨가.

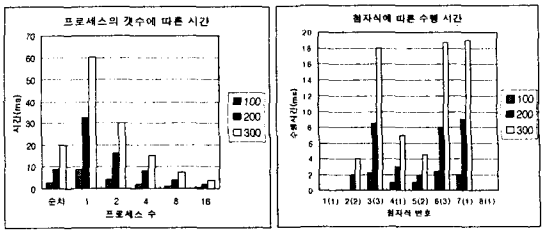
⑦ 변형된 루프에 대해 DOALL을 수행.

III. 모의 실험 및 분석

제안한 알고리즘을 (그림 1)의 예제 그래프에 적용하여 실험을 한다. N1, N2를 각각 100, 200, 300으로 두고 순차적 실행과 제안한 알고리즘에 적용시 생성된 병렬 코드를 병렬로 수행했을 때의 시간을 측정하였다. (그림 3)은 생성된 병렬 코드로 프로세스의 개수를 달리하여 실행한 것과 원래의 순차코드를 비교해 놓은 것이다. (그림 4)는 각 Case에서 첨자식에 따른 수행 시간을 비교해 놓은 것이다. Case1의 ①, ⑧은 루프의 범위가 0을 포함 하지 않기 때문에 고려 되지 않았다.

VI. 결론 및 향후 연구

가변 종속거리를 가지는 다중 첨자 루프에서 종속성이 발생하는 첨자값을 구하는 방법과 병렬코드를 생성하는 알고리즘을 제안했다. 또한 모의 실험을 통해서 제안한 알고리즘으로 병렬 코드를 생성하고, 병렬로



(그림 4) 프로세스 수에 다른(그림 3) 각 첨자식에 따른 수행 시간

수행 할 수 있음을 보였다. 종속성이 존재하는 문장에 대해서 최소 6개의 새로운 루프가 생성되므로 종속성이 많이 존재하는 루프에 대해서는 지나치게 많은 코드가 생성된다는 단점을 가지고 있다. 2차원 배열을 갖는 루프에의 적용은 제안한 알고리즘 외에 몇 가지 더 고려해야 할 점이 있다. 또한 병렬 코드를 생성할 때 Case1, Case2, Case3에서 중복되는 부분을 고려하지 않았는데 이를 제거하고 좀더 최적화 시키는 것은 계속 연구해야 할 부분이다.

참고문헌

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

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

[3] 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

[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] 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

[6] 송월봉, 박두순, "최대 병렬성 추출을 위한 자료종속성 제거 알고리즘", KISS Trans. Vol. 26, No. 1, 1999