論文94-31B-10-1

완전 중첩 루프에서 병렬처리를 위한 새로운 동기화 기법

(A New Synchronization Scheme for Parallel Processing on Perfectly Nested Do Loops)

李 鑛 炯*, 黃 鍾 善*, 朴 斗 淳**, 金 秉 洙***

(Kwang Hyung Lee, Chong Sun Hwang, Doo Soon Park and Byung Soo Kim)

要約

일반적으로 응용 프로그램에서 루프는 전체 수행시간중 상당 부분을 차지하는 자원이고 가장 중요한 병렬성 추출의 기본이 된다.이러한 루프의 반복들을 멀티 프로세서상에서 수행할 때, 각 반복간에 존재하는 데이타 종속이 프로세서간의 동기화에 의해 유지되어야 한다. 본 논문에서는, 기존의 데이타 지향 기법에서 과도한 동기화 변수의 사용으로 인해 발생하는 오버헤드를 줄이고,문장 지향 기법에 동기화 명령으로 인해 발생하는 시간 지연을 줄이기 위한 새로운 동기화 기법(Free/Hold)을 제시한다. 이 기법은 완전 중첩 루프에서 데이타간의 실제종속거리를 이용하여 종속관계를 갖는 각 인스턴스에 동기화 명령을 삽입함으로써 올바른 수행순서를 유지할 수 있다. 또한 일대다 종속에서 불필요한 종속을 제거하기 위한 기법을 제시한다.

Abstract

In most application programs,loops usually contain most of the computation in a program and are the most improtant source of parallelism. When loops are executed on multiprocessors,the cross iteration data dependences need to be enforced by synchronization between processors.In this paper, we propose a new synchronization scheme(Free/Hold) for reducing overheads occured by synchronization variables in data_oriented scheme and delay of time occured by synchronization instruction in statement_oriented scheme. The Free/Hold mechanism enforces the correct execution order by inserting synchronization instruction between each instance with data dependence relationship using the RD(Real dependence Distance).We also present an algorithm for removing unnecessary dependences in one-to-many dependences.

*正會員, 高麗大學校 電算科學科
(Dept. of Computer Science, Korea Univ.)
**正會員, 順天鄕大學校 電算學科
(Dept. of computer Science, Soonchunhyang univ.)
***正會員, 建陽大學校 電子計算學科
(Dept. of Computer Science, Keonyang Univ.)
接受日字 : 1994年 2月 3日

Ⅰ. 서 론

폰노이만 방식의 컴퓨터 시스템은 주어진 연산을 순차적으로 수행하기 때문에 연산속도를 빠르게 하는 데는 한계성을 갖는다. 그러나 하드웨어 구성요소들의 성능이 향상되고 가격이 저렴화됨에 따라 병렬처리 시스템이 성능향상을 위한 필수적인 요인으로 인식되면서 Cray시스템, Alliant FX/8, IBM 3090, Cedar 그리고 RP3와 같이 병렬처리를 지원하는 시스템들이 개발되고 있다.[4,11,13]

이러한 컴퓨터 시스템들을 효율적으로 이용하기 위해서는 사용자가 해당 시스템의 구조를 이해하고, 응용 프로그램이 시스템 내부에서 어떻게 처리되는지를 알아야 하기 때문에 사용자가 직접 병렬처리가 가능한 형태로 프로그래밍을 하는 것은 상당한 부담이 된다. 그러므로 이러한 과중한 부담을 해결하기 위해서는 자동적으로 응용 프로그램을 병렬처리가 가능한 형태로 바꾸어 주는 재구조화(restructuring)기법 즉, 병렬 컴파일러의 필요성이 대두된다. 이러한 재구조화 기법에서 루프의 구조는 가장 중요한 부분이 되는데 이는 응용 프로그램의 전체 수행시간 중에서 상당 부분을 차지하며 병렬성 추출의 기본이 되기 때문이다.[2] 루프의 구조는 중첩(nested)정도에 따라 완전 중첩 루프(perfectly nested do loop)와 불완전 중첩 루프(nonperfectly nested loop)로 분류할 수 있다.

```
do i1 = 1, N1
  do i2 = 1, N2
     . . .
    do im = 1, Nm
      statement-1;
        . . .
      statement-n;
    end do
      . . .
  end do
end do
(a) m차 완전 중첩 루프
```

```
do i1 = 1, N1
  statement-1;
  do i2 = 1, N2
    statement-2;
      . . .
    do im = 1, Nm
      statement-n;
    end do
      . . .
  end do
end do
(b) m차 불완전 중첩 루프
```

그림 1. 루프의 구조
Fig. 1. The structure of a loop.

이러한 중첩 루프는 모든 반복(iteration)들간에 종속성이 없는 경우(do-all)와 서로 다른 반복간에 종속성이 발생하는 경우(do-across)가 존재한다. 또한 루프를 펼쳤을 때 (unrolled)모든 문장 각각을 인스턴스(instance)라 하며 특히, 종속의 근원이 되는 인스턴스를 source 인스턴스, 종속이 되는 인스턴스를 sink 인스턴스라 한다. Do-all 형태에서는 반복간에 종속이 발생하지 않기 때문에 모든 인스턴스를 프로세서간에 어떠한 상호작용(interaction)없이 병렬처리할 수 있으나, do-across 형태에서는 한 반복에서 생성된 결과가 나중에 다른 반복에서 사용되거나 한 반복에서 가져온 인스턴스가 다른 반복에서 수정되는 종속성이 발생하기 때문에 부분적으로 순차처리해야 한다. 이러한 종속성 유지를 위한 프로세서간의 상호 정보교환을 동기화 기법이라 하며 프로그램의 올바른 수행을 위해서 반드시 유지되어야 한다. 종속성 유지를 위한 동기화 기법은 병렬성 추출 형태에 따라 데이타 지향 기법(date-oriented scheme)[4,5,7,14]과 문장 지향 기법(statement-oriented scheme)[12,14]으로 나눌 수 있다. 데이타 지향 기법은 데이타 종속관계를 갖는 각각의 모든 인스턴스에 동기화 변수를 할당하기 때문에 과도한 동기화 변수의 사용으로 인한 오버헤드가 존재한다. 즉 m차 중첩 루프에서 필요한 동기화 변수의 수는 πN_i(N_i는 루프의 상한값, $1 \le i \le m$)가 된다. 문장 지향 기법은 종속관계를 갖는 모든 인스턴스에 하나의 동기화 변수를 할당하기 때문에 source 인스턴스의 수행이 완료된 후 해당 sink 인스턴스를 실행하기 위해서는 이전의 모든 반복의 수행이 완료되어야 하기 때문에 동기화 명령에 의한 시간 지연이 발생하게 된다.

기존의 동기화 기법의 문제점들을 해결하기 위해서 본 논문에서는 공유 메모리를 갖는 밀착 결합(tightly coupled) MIMD 시스템에서 완전 중첩 루프(그림 1의 (a)참조)형태의 데이타 접근 순서를 효율적으로 관리하기 위한 방법으로 새로운 동기화 기법(Free/ Hold)을 제시한다.

Ⅱ. 프로그램 종속성 정의

Fortran, Pascal, C 등과 같은 언어로 작성된 프로그램에서 문장들의 수행순서는 위에서 아래로 순차적이다. 이러한 문장들의 병렬수행이 순차적인 수행과 동일한 결과를 가져 오는가를 알기 위해서는 문장 또는 데이타간에 어떠한 종속관계가 존재하는 가를 조사하는 것이 필수적이다.

[정의 2.1]

그림 1의 (a)와 같은 m차 중첩 루프내의 문장 $S_p(i_1, i_2, \cdots i_m)$, $S_q(j_1, j_2, \ldots, j_m)$에 대해

i) 어떤 정수 $k(1 \le k \le m)$에 대해 $i_1=j_1, i_2=j_2, \cdots, i_{k-1}=j_{k-1}, i_k < j_k$ 이거나

ii) $i_1=j_1, i_2=j_2, \cdots, i_k = j_k$ 이고 $p<q$일때

문장 S_p는 문장 S_q보다 먼저 수행되는 것을 의미하고 ($Sp \ll S_q$로 표현)이를 문장이 사전식 수행 순서(lexicographic execution order)라 한다.

치환 연산자(=)를 포함하는 문장 S에 대해 치환 연산자의 왼쪽에 있는 변수들의 집합을 OUT(S)라 하고 치환 연산자 오른쪽에 나타나는 변수들의 집합을 IN(S)라 할때, 다음과 같이 세가지 종속성, 흐름 종속성(flow dependence), 반 종속성(anti dependence)및 출력 종속성(output dependence)을 정의할 수 있다.[3,8]

[정의 2.2]

m차 중첩 루프내의 두 문장 S_p,S_q에 대해

i) $S_p \ll S_q$ 이고 $OUT(S_p) \cap N\ IN(S_q) \neq \phi$ 일때 S_q는 S_p에 흐름 종속이 된다고 하고 $S_p \delta S_q$로 표시한다.

ii) $S_p \ll S_q$ 이고 $IN(S_p) \cap OUT(S_q) \neq \phi$ 일때 S_q는 S_p에 반 종속이 된다고 하고 $S_p \delta^a S_q$로 표시한다.

iii) $S_p \ll S_q$ 이고 $OUT(S_p) \cap OUT(S_q) \neq \phi$ 일때 S_q는 S_p에 출력 종속이 된다고 하고 $S_p \delta^\circ S_q$ 로 표시한다.

데이타 종속성은 데이타간의 종속관계 유뮤를 파악할 수 있으나 두 데이타간의 정확한 종속위치를 파악하기 위해 데이타 종속거리를 이용하는데, 이는 종속관계를 갖는 두 문장 S_p와 S_q사이에 실행되는 반복 횟수를 S_p와 S_q 사이의 종속거리라 하고 $D(S_p,S_q)$로 나타낸다. m차 중첩 루프에서 종속관계를 갖는 두 문장 S_p, S_q의 배열 첨자식이 각각 $[a_1i_1+b_1, a_2i_2+b_2, \cdots, a_mi_m+b_m]$, $[c_1i_1+d_1, c_2i_2+d_2, \cdots, c_mi_m+d_m]$ (여기서 a_k,b_k,c_k,d_k는 정수, i_k 는 루프 변수$(1 \le k \le m)$)라고 하면 두 문장사이의 종속거리 $D(S_p,S_q)$는 다음과 같이 표현할 수 있다.

$$D(S_p,S_q) = \begin{cases} |b_k - d_k| & \text{if } a_k = c_k \\ \text{variant} & \text{if } a_k \neq c_k \end{cases}$$

여기서 루프 변수의 계수 a_k와 c_k는 증가 계수(increment factor)하고 한다. 본 논문에서는 배열 첨자식의 단순 표현으로 a*i+b형태로 나타내며, 종속관계를 갖는 데이타의 증가 계수가 동일한 경우 (불변 종속거리)에 대해서 다룬다.

[정의 2.3]

종속거리, $D(S_p,S_q)$가 루프 변수 값(index value)에 관계없이 항상 일정하게 유지될 때, 즉, $D(S_p(i),S_q(j)) = D(S_p(i'),S_q(j'))$ (여기서 i,j,i',j'는 종속관계를 갖는 시점에서 의 루프 변수의 값) 일때 불변 종속거리를 갖는다고 하며 루프내의 모든 종속관계가 불변 종속거리를 갖는 루프를 불변 종속거리 루프라 한다.

```
do i = 1, 10
  Sp : A[i] =
  Sq : ... + A[i-2]
end do
```

그림 2. 불변 종속거리 루프

Fig. 2. A constant dependence distance loop.

그림 2에서 종속관계를 갖는 두 문장간의 종속거리 $D(S_p(1), S_q(3)) = D(S_p(2),S_q(4)) = \cdots = D(S_p(8), S_q(10))=2$가 된다. 즉, 종속관계를 갖는 두 문장간의 종속거리는 항상 2가 되기 때문에 불변 종속거리를 갖는다.

Ⅲ. 동기화 기법의 종류

기존의 프로그램 재구조화 기법들은 순차 프로그램을 병렬화할 때 조건문과 같이 프로그램의 구조상 적용에는 한계가 있다. 비록 병렬화할 수 있다 하더라도 대개의 경우 병렬성 추출을 위한 조건조사,최적값 적용, 최대공약수 및 최소공배수 등의 계산에 소요되는 오버헤드가 발생하게된다. 동기화 기법[1]은 이러한 결점들을 보완하는 유력한 방법으로 병렬성 추출 형태 및 구현방법에 따라 데이타 지향 기법(data-oriented scheme), 문장 지향 기법 (statement-oriented scheme) 등으로 나눌 수 있다.[10]

1. 데이타 지향 기법

데이타 지향 기법은 데이타의 접근순서를 유지하기 위해서 데이타 종속을 이루는 source데이타 각각의 모든 인스턴스에 동기화 변수, AC(Access Count)를 할당한다. 종속관계를 갖는임의의 인스턴스(source 인스턴스, sink 인스턴스)를 수행할 때 먼저 source 인스턴스를 수행한 후 수행 완료를 표시하기 위해서 해당 AC의 값을 수정하고 , sink 인스턴스를 수행하기 전에 해당 source 인스턴스가 완료되었는가를 확인한 후 (AC의 값 확인) 수행 여부를 결정한다.

그림 3에서 보는 바와같이 source 데이타 S1의모든 인스턴스 (A[i], $1 \le i \le N$) 문장 지향 기법과는 달리 source 인스턴스의 수행이 완료된후 바로 이에 대한 신호 (AC의 변경, A[i]%key=2)를 보낼 수 있기 때문에 source 인스턴스 수행직 후 해당 sink 인스턴스를 즉시 처리할 수 있지만 종속관계를 갖는 각 데이타의 모든 인스턴스에 루프의 상한값에 해당하는 동기화 변수(A[i]%KEY, $1 \le i \le N$)를 할당하기 때

문에 과도한 동기화 변수의 사용으로 인한 오버헤드가 존재한다. 즉, m차의 중첩 루프인 경우, 요구되는 동기화 변수의 수는 πN_i(N_i는 각 루프의 상한값, $1 \le i \le m$)가 된다. 또한 이러한 동기화 변수는 수행중 임의의 한 시점에서만 사용되기 때문에 과도한 메모리 손실을 초래하게 된다.

```
do i = 1, N
S1 : A[i] = . . .
S2 : . . . = A[i-2]
end do
```

(a) 예제 루프

```
doacross i = 1, N
  A[i]%DATA = . . .
  A[i]%KEY = 2
  with A[i-2] when (A[i-2]%KEY = 2)
    . . . = A[i-2]%DATA
  end with
```

(b) 동기화된 루프

그림 3. 확장된 Full/Empty 동기화 기법의 예
Fig. 3. An example of Extended Full/Empy synchronization scheme.

이러한 데이타 지향 기법으로는 Denelcor HEP의 Full/Empty tag 기법[4,14], Cedar시스템의 integer-key 기법(synch-read, synch-write)[5,7]등이 있다.

2. 문장 지향 기법

문장 지향 기법은 루프의 올바른 수행순서를 유지하기 위해서, 종속관계를 갖는 모든 인스턴스에 하나의 동기화 변수, SC(Statement Counter)를 할당한다. 즉, 이 동기화 변수는 종속관계를 갖는 모든 인스턴스에 의해서 공유된다. 문장의 수행순서 유지는 임의의 반복 k가 source 인스턴스를 수행한 후 SC=k-1이 될때까지 대기한 후 SC의 값을 k의 값으로 수정한다. 또한 반복 k에서 sink 인스턴스를 수행하기 위해서는 이전의 모든 source 인스턴스의 수행이 완료되기까지 대기한 후 해당 인스턴스를 수행한다.

```
doacross
S1 : A[i] = . . .
TS : testset(R)
S2 : . . . = A[i-2]
end doacross
```

```
testset(R)
if (R < i) then
  with R when(R=i-1)
    R = R + 1
  end with
end if
```

i =	1	2	3	4
time 1	S1	S1	S1	S1
	TS			
	S2	TS		
		S2	TS	
			S2	TS
				S2

(a) 그림 3의 동기화된 루프 (b) testset 명령 (c) 수행 순서

그림 4. Testset/test 동기화 기법의 예
Fig. 4. An example of testset/test synchronization scheme.

위의 그림 4의 (b)에서 보는 바와같이 종속관계를 갖는 문장(S1δS2)은 하나의 동기화 변수 (R)를 공유하기 때문에 데이타 지향 기법에서의 과도한 동기화 변수의 사용으로 인한 오버드는 존재하지 않지만 (c)에서와 같이 i=1,2,3,4상의 문장 S2는 S1의 수행이 완료 된후 바로 수행될 수 있지만 동기화 명령(testset)에 의해 결과적으로 순차적으로 수행된다. 따라서 병렬성을 최대한으로 활용할 수 없는 문제가 발생하게 된다. 이러한 문장 지향 기법으로는 Cray X-MP의 Lock/Unlock, Alliant FX/8의 Advance/Wait[12,14] 에서 제안한 Set/Wait, Testset/Test 동기화 기법등이 있다.

3. 효율적인 동기화 기법의 요건

[9]에서는 7개의 벤치마크(benchmark) 프로그램(표 1 참조)을 이용하여 각각의 수준(operation-level, Statement-level, subroutine-level)에 대하여 병렬성의 크기를 추출하였다. 그림 5에 의하면 연산수준에서의 병렬성 정도가 가장 크고 서브루틴 수준에서 병렬성은 가장 적음을 보여준다. 그러나 병렬성 추출의 수준이 작을수록 해당 프로그램에서 얻을 수있는 병렬성의 크기는 증가하지만 병렬성의 크기가 증가할수록 그에 대한 오버

표 1. Benchmark 프로그램의 특징
Table 1. Characterictic of Benchmark program.

Benchmark	Lines	Applications	No. of Subroutine
SIMPLE	2121	Hydrodynamics and heat flow	14
PIC	378	Particle in cell	2
LINPACK	609	Linear system software	9
QCD	2326	Quantum chromodynamics	34
FLO52Q	2250	Computational fluid dynamics	27
TRACK	3784	Signal processing	31
MDG	1231	Liquid water simulation	15

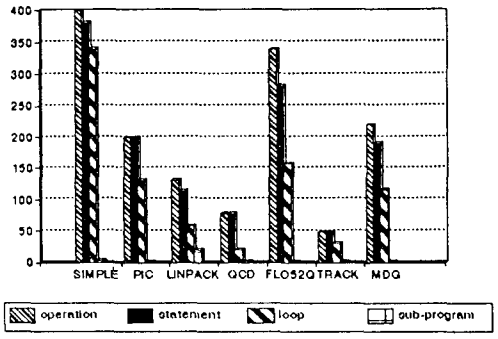


그림 5. 벤치마크 프로그램의 병렬성
Fig. 5. Granularity and parallelism of benchmark programs.

헤드가 커지게 된다. 따라서 가장 효율적인 동기화 기법은 병렬성 추출의 수준은 낮게하고 초래되는 오버헤드를 최소화시킬 수 있는 기법임을 알 수 있다.

Ⅵ. 새로운 동기화 기법

본 절에서 제시하는 동기화 기법(Free/Hold)은 불변 종속거리 루프를 효율적으로 동기화할 수 있는 기법으로, 데이타 지향 기법에서의 과도한 동기화 변수의 사용으로 인해 발생하는 오버헤드와 문장 지향 기법에서의 동기화 명령에 의해 발생하는 시간지연을 최소화할 수 있는 동기화 기법이다.

1. Free/Hold동기화 기법

주어진 루프에 대해서 어느 정도의 동기화 변수가 필요한지를 결정하기 위해서는 데이타 종속을 이루는 데이타간의 종속거리(dependence distance)가 필요하다. 이에 따라 본 절에서는 기존의 종속거리보다 더 정확한 거리를 표현할 수 있는 실제종속거리(Real dependence Distance : RD)를 제시한다.

[정의4.1]

m차의 완전 중첩루프에서 실제종속거리(RD)는 다음과 같이 정의한다.

$$RD = (\sum_{i=1}^{m} (T * D(S_p, S_q) * \prod_{k=i+1}^{m} N_k) + (1-f)$$

여기서 1 = sink 데이타의 문장번호,
f = source 데이타의 문장번호
T = 루프내의 전체 문장의 수,
N_k = 각 루프의 상한값

[정의 4.1]의 RD는 [7]에서 제시하는 종속거리(True Distance : TD)보다 더 정확한 종속거리를 표현할 수 있다. 즉, TD에서는 종속관계를 이루는 source 데이타와 sink 데이타사이의 전체 반복의 수를 나타내지만 실제종속거리는 source 데이타와 sink 데이타사이의 문장의 수를 나타낸다.

```
Do i = 1, N
  S1 : A[i]=.....
  Sk : ... = A[i-3]
End do
```

그림 6. 예제 루프
Fig. 6. An example loop.

그림 6에서 변수 A에 대한 $D(S_1,S_k)$=3이기 때문에 TD=3이 되고 이를 인스턴스간의 종속으로 표현하면 3k가 되고,RD는 3k+(k-1)이 된다. 따라서 RD는 TD보다 항상 (k-1)의 거리를 더 정확하게 표현할 수 있다. 또한 이 RD는 주어진 루프내에서 필요한 동기화 변수의 수를 결정하는 기본이 된다.

[정리 1]

데이타간의 종속성을 유지하기 위한 동기화 변수의 수는 RD의 값을 이용한다.

(증명)

〈step -1〉

RD가 1인 종속 체인 (dependence chain)을 구성하면 [1,2,3,…]형태로 하나의 종속체인이 구성된다. RD가 루프를 펼쳤을 때 첫번째로 발생하는 source 인스턴스와 sink 인스턴스 바로 전의 종속거리를 나타내기 때문에 1개의 동기화 변수를 이용하면 주어진 루프를 올바르게 수행할 수 있다.

〈step-2〉

RD가 k-1일 때 위의 정리가 성립한다고 가정하면 k-1개의 종속 체인, [1,(k-1)+2(k-1)+1,…],[2,(k-1)+2,2(k-1)+2,…],…,[k-1,2(k-1),3(k-1),…]이 구성되고 k-1개의 동기화 변수를 공유하여 종속성을 유지할 수 있다.

〈step-3〉

RD가 k일때 종속 체인을 구성하면 [1, k+1, 2k+1, …,], [2, k+2, 2k+2m, …], …, [k, 2k, 3k, …] 즉, [1,(k-1)+1+1, 2(k-1)+2+1, …], [2,(k-1)+1+2, 2(k-1)+2+2…), …,

[k,(k-1)+1+K,2(k-1)+2+k,…]형태가 된다. 이를 다시 표현하면 다음과 같다.

〈step-2〉에 의해 [1,(k-1)+1,2(k-1)+1,…]
k-1개의 동기화 [2,(k-1)+2,2(k-1)+2,…]
변수 공유 …
[k,(k-1)+k,2(k-1)+k,…]
+ …
〈step-1〉에 의해 [0,2,2,3,...]
1개의 동기화 [0,2,2,3,...]
변수 공유 …
[0,2,2,3,...]

따라서 RD가 k일때 〈step-1〉과 〈step-2〉에 의해 k개의 동기화 변수를 공유하여 주어진 루프의 종속성을 유지 할 수 있다. ■

여기서 제안하는 동기화 변수의 수는 첫번째 발생하는 source와 sink 인스턴스사이의 RD에 의해서 결정되기 때문에 전체 루프의 수행을 위해서는 다른 모든 인스턴스들은 해당 동기화 변수를 공유해야 한다. 즉, X를 해당 시스템에서 사용되는 동기화 변수의 수라 할 때 [i,X+i,2X+i,…] 형태로 동기화 변수를 공유한다.

2. Free/Hold 동기화 알고리즘

Free/Hold 동기화 기법은 source 인스턴스 바로 다음에 해당 인스턴스의 수행이 완료되었음을 나타내는 Free 명령을 삽입하고 sink 인스턴스 바로 전에 해당 source 인스턴스의 수행이 완료되기까지 대기하게 하는 Hold 명령을 삽입함으로써 동기화를 수행한다. 불변 종속거리 루프에 올바른 동기화 명령을 삽입하기 위한 방법은 [알고리즘 1]과 같다. 이 알고리즘에서 모든 인스턴스의 정확한 위치를 표현하기 위해서 전체문장번호(unrolled statement number)를 사용하는데, 이는 루프를 펼쳤을 때 각각의 모든 인스턴스에 붙인 번호를 의미하며 종속관계를 이루는 첫번째 source 인스턴스부터 번호를 붙이고 이를 첫번째 전체문장번호라 한다.

[알고리즘 1] Free/Hold동기화 알고리즘

* X_j= number of synchronization variables $(0 \le j \le RD-1)$
* k= unrolled statement number
* α= unrolled statement number of the first source instance
* β= unrolled statement number of the first sink instance
* T = number of total statement in a loop
* N_i= upper bound of each loop$(1<i<m)$

(1) Insertion of Free instruction

for each statement $(k = \alpha; k \le \prod_{i=1}^{m} N_i * T; k = k + T)$

switch (k mod RD)

case j : Insert the statement Free (k, X_j)

immediately after the statement S_k;

(2) Insertion of Hold instruction

for each statement $(k = \beta; k \le \prod_{i=1}^{m} N_i * T; k = k + T)$

switch (k mod RD)

case j : Insert the statement Hold(k, X_j)

immediately before the statement S_k);

[알고리즘 1]를 중첩 루프에 적용할 때, 종속관계를 갖지 않는 sink 인스턴스에 동기화 정보가 포함되는 경우가 발생한다. 이는 각 루프의 종속거리에 의해서 발생되는데 $N_i(1 \le i \le m)$를 각 중첩 루프의 상한값이라 하고, N_i-$D_i(S_p, S_q)$=k_i라 할때, 각 중첩 루프의 상한값내의 끝부분부터 N_i-K_i(m-1차 이하의 루프에서는 $\sum(N_i - k_i) * \prod N_j, i+1 \le j \le m$)개의 인스턴스는 종속관계가 존재하지 않는다.

[알고리즘 1]에 의해 재구조화된 루프의 올바른 수행 순서를 유지하기 위한 동기화 연산은 다음과 같다.

[알고리즘 2] Free/Hold 동기화 연산

* k = unrolled statement number
* X_j= number of synchronization variables $(0 \le j \le RD-1)$
* mod = modulus operation

(1) Free (k, X_j)

Switch (k mod RD)

case j : increment X_j;

(2) Hold (k, X_j)

j = k mod RD

wait until (X_j == ⌐(k div (RD*T))¬ ;

위의 free 연산에서 switch 문은 대응되는 동기화 변수를 선택하여 source 인스턴스의 수행이 완료되었음을 나타내기 위하여 동기화 변수의 값을 수정하고, Hold 연산에서 until문은 현재 수행하고자 하는 sink 인스턴스에 대응되는 동기화 변수를 점검하여 수행여부를 결정한다.

3. Free/Hold의 적용

일반적으로 루프내에서 데이타간의 종속성은 하나의 데이타에 단지 하나의 데이타가 종속관계를 이루는 경우와 여러개의 데이타와 종속관계를 이루는 경우가 존재한다. 전자의 경우를 일대일(one-to-one) 종속관계(그림 7참조), 후자를 일대다(one-to-many) 종속관계(그림 8참조)라 한다.

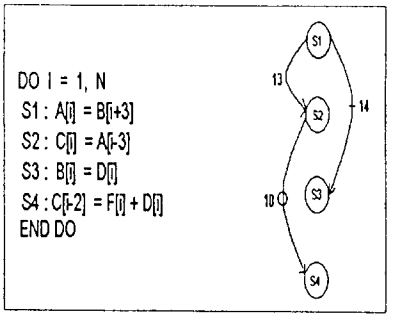


그림 7. 일대일 종속관계

Fig. 7. One-to-one dependence.

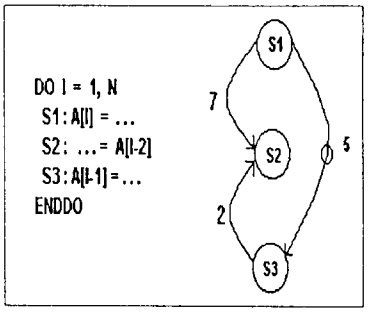
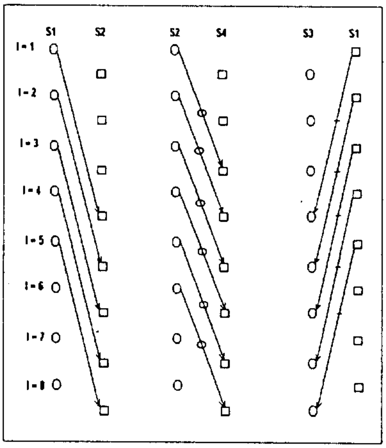


그림 8. 일대다 종속관계

Fig. 8. One-to-Many dependence.

종속 그래프에서 →는 흐름종속, ↛는 반종속 그리고 ↠는 출력종속을 의미하며, 그래프상의 정수값은 실제종속거리(RD)를 의미한다.



(a) A의 종속관계 (b) C의 종속관계 (c) B의 종속관계

그림 9. 각 변수의 인스턴스에 대한 종속관계

Fig. 9. The dependence for instance of each variable.

반복	전체문장번호	동기화명령의삽입
I=1	1	S1
		Free(1, X_1)
	2	S2
	3	S3
	4	S4
I=2	5	S1
		Free(5, X_5)
	6	S2
	7	S3
	8	S4
I=3	9	S1
		Free(9, X_9)
	10	S2
	11	S3
	12	S4
I=4	13	S1
		Free(13, X_0)
		Hold(14, X_1)
	14	S2
	15	S3
	16	S4
I=5	17	S1
		Free(17, X_4)
		Hold(18, X_5)
	18	S2
	19	S3
	20	S4
I=6	21	S1
		Free(21, X_8)
		Hold(22, X_9)
	22	S2
	23	S3
	24	S4
...	...	...

그림 10. 재구조화된 루프

Fig. 10. A restructured loop.

1) 일대일 종속관계

그림 7에서는 변수 A,B,C 각각은 일대일 종속관계가 존재한다. 여기서 각 변수에 대한 모든 인스턴스의 종속관계를 표현하면 그림 9와 같다.

그림 7의 루프에서 변수 A에 대한 RD는 13, B에 대한 RD는 14 그리고 C에 대한 RD는 10이기 때문에 각 데이타에 대해서 13,14,10개의 동기화 변수를 공유하면 주어진 루프의 종속성을 유지하면서 병렬처리를 할 수 있다. [알고리즘 1]를 이용하여 루프를 재구조하 시킨 unrolled 루프는 그림 10과 같다. (여기서 변수 A에 대해서만 표현)

2) 일대다 종속관계

그림 8에서 변수 A는 $S_1\delta S_2$, $S_3\delta S_2$ 형태의 일대다 종속관계가 존재한다. 이와 같은 일대다 형태의 종속성을 유지하기 위한 기본적인 방법은 일대일 종속성과 마찬가지로 각 종속성을 독립적을 유지하는 것이다. 따라서 $S_1\delta S_2$에 대해서 [알고리즘 1]를 적용하여 종속관계를 유지하고 $S_3\delta S_2$ 대해서도 [알고리즘 1]를 적용하여 전체 종속관계를 유지한다. 따라서 $S_1\delta S_2$에 대한 RD는 7이기 때문에 7개의 동기화 변수 그리고 $S_3\delta S_2$에 대한 RD는 이기 때문에 2개의 동기화 변수를 공유함으로써 전체 종속성을 유지할 수 있다. 그러나 두개 이상의 데이타에 종속이되는 데이타는 항상 종속거리가 가장 작은 데이타에 의해서만 종속성 여부가 결정되기 때문에 일대다 종속관계에서 RD가 가장 작은 종속관계만 유지하고 그외의 데이타에 대한 종속성은 제거할 수 있다.

[정리 2]

일대다 루프에서, 가장 작은 RD를 갖는 source 인스턴스의 루프 변수의 값이 가장 크다.

〈증명〉

일대다 종속관계($S_r\delta S_t$, $1\le r$, $t\le n$ 여기서 S_r는 두개 이상의 source 데이타, S_t는 하나의 sink 데이타)를 갖는 루프에서, 데이타 종속을 이루는 각 데이타의 RD를 각각 $RD_{r,t}$라 하자.

또한 $RD_{r',t}$를 모든 RD중 가장 작은 값을 갖는 RD라 가정하자.

($RD_{r',t} < RD_{r',t}$ 여기서 $1\le r'\le n$, $r' \ne r$).

해당 전체문장번호를 $r_{r,t}$로 표현할 때, 임의의 한 시점에서 종속관계를 갖는 모든 인스턴스의 전체문장번호 중 $r_{r',t}$는 가장 큰 문장번호를 갖는다(∵가정에 의해)

임의의 한 시점에서, 각 인스턴스의 루프 변수의 값은 다음과 같다.

$\lceil r_{1,t}/T \rceil = I_1$

$\lceil r_{2,t}/T \rceil = I_2$

.... (i)

$\lceil \gamma_{r',t}/T \rceil = I_{r'}$

...

$\lceil \gamma_{n,t}/T \rceil = I_n$, T는 루프내의 전체 문장의 수

(i)에서 $\gamma_{r',t} > \gamma_{r,t}$이기 때문에 $I_{r'} > I_r$ 가 된다. ■

[정리 3]

일대다 종속관계에서 S_t는 항상 RD가 가장 작은 source 데이타에 의해서만 종속된다.

〈증명〉

임의의 한 시점에서 종속관계를 갖는 각 인스턴스의 전체문장번호를 $\gamma_{r,t}$ 라하자. 루프를 반복할 때마다 각 인스턴스의 전체문장번호는 다음과 같이 증가한다.

$\gamma_{1,t}+T,\ \gamma_{2,t}+T,\ \ldots,\ \gamma_{n,t}+T$ /*한번 반복 후*/

$\gamma_{1,t}+2T,\ \gamma_{2,t}+2T,\ \ldots,\ \gamma_{n,t}+2T$ /*두번 반복 후*/

...

$\gamma_{1,t}+NT,\ \gamma_{2,t}+NT,\ \ldots,\ \gamma_{n,t}+NT$ /*N번 반복 후*/

위의 식에서 각각의 반복때마다 전체문장번호가 γ_t인 sink 인스턴스는 항상 모든 γ_r에 의해 종속된다

그러나 γ_t는 최종적으로 가장 마지막에 영향을 미치는 source인스턴스 즉, 루프 변수의 값이 가장 큰 source 인스턴스에 의해 종속이 결정된다.

따라서 [정리 2]에 이해 실제종속거리가 가장 작은 $\gamma_{r'}$에 의해서만 종속이 결정된다. ■

위의 [정리 2]와 [정리 3]을 이용하여 일대다 종속관계를 갖는 불변 종속거리 루프에서 종속성을 제거하기 위한 알고리즘은 다음과 같다.

[알고리즘 3] 일대다 종속에서의 불필요한 종속성의 제거

input : dependence relationships between each data

output: dependence relationship which is deleted unnecessary dependences

begin : find the dependence relationship such that $S_r \delta S_t$(here St, exist only one,

S_t exists more than two)

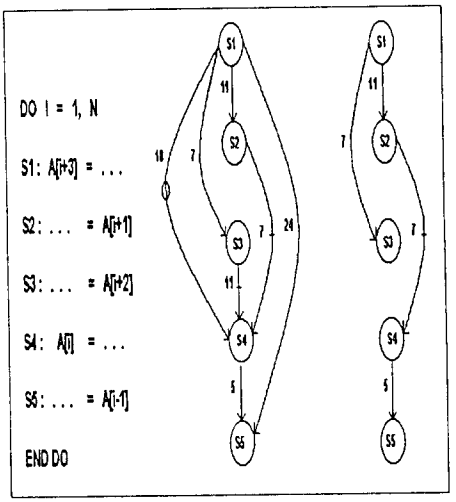
calculate the RD for each dependence relationship

$RD_{min} = \min (RD_{rt})$

remove all dependence except the dependnece RDmin.

end

그림 11에서 문장 S4는 항상 문장 S1(i가 1일때 A[4]), S2(i가 3일때 A[4]), S3(i가2일때 A[4])에 종속된다. 그러나 S1,S2,S3의 종속위치 (루프 변수의 값)를 볼때 S2의 종속 위치가 S1과 S3의 종속위치보다 크다. 즉, S2의 RD가 가장 작다. 이는 최종적으로 문장 S4는 S2에 종속됨을 알 수 있다. 따라서 [알고리즘 3]에서 제시한 바와 같이 S1δS4, S3δS4의 종속성을 제거해도 S4의 수행결과에는 아무런 영향을 미치지 않는다.



(a) 예제 루프 (b) 종속 그래프 (c) 불필요한 종속이 제거된 그래프

그림 11. 일대다 종속 루프의 예

Fig. 11. An example of one-to-many dependence.

V. 성능 비교

기존의 동기화 기법(Extended Full/Empty :EFE)과 본 논문에서 제시하는 Free/Hold 동기화 기법을 비교하기 위해서, 주어진 루프를 동기화하는데 필요한 동기화 변수의 수와 동기화 명령을 수행하는데 소요되는 시간을 비교한다. Free/Hold 기법은 데이타 지향 기법과 같이 모든 인스턴스에 동기화 변수를 할당하는 것이 아니라 루프의 수행 순서를 유지하기 위해서 필요한 최소의 동기화 변수 (첫번째 발생하는 source와 sink 인스턴스사이의 RD의값)만을 할당한다. 따라서 데이타 지향 기법에서 발생하는 동기화 변수에 의한 오버헤드를 줄일 수 있다.

본 절에서는 요구되는 동기화 변수의 수를 비교하기 위해서 그림 1 의 (a)와 같은 루프구조(여기서, $1 \le m \le 4$, $1 \le n \le 2$, $N_m = 100$ 으로 가정) 에서 다음과 같은 4가지 형태의 배열 변수의 첨자식(sub-script)에 따른 필요한 동기화 변수의 수를 이용한다.(표 2 참조)

〈타입-1〉	〈타입-2〉	〈타입-3〉	〈타입-4〉
$j_1 = i_1$	$j_1 = i_1$	$j_1 = i_1$	$j_1 = i_1-2$
$j_2 = i_2$	$j_2 = i_2$	$j_2 = i_2 -2$	$j_2 = i_2-2$
$j_3 = i_3$	$j_3 = i_3 -2$	$j_3 = i_3 -2$	$j_3 = i_3-2$
$j_4 = i_4 -2$	$j_4 = i_4 -2$	$j_4 = i_4 -2$	$j_4 = i_4-2$

표 2. 첨자식의 형태에 따른 동기화 변수의 수
Table 2. The number of synchronization variable according to subscript forms.

차원(m) / 동기화 기법		$N_m = 100\ (1 \le m \le 4)$			
		1	2	3	4
EFE		100	10,000	1,000,000	100,000,000
F/H	타입-1	#	#	#	5
	타입-2	#	#	5	405
	타입-3	#	5	405	40,405
	타입-4	5	405	40,405	4,040,005

※ # : 반복간에 종속성이 없음을 의미

표 2에서 보는 바와같이 Free/Hold 기법은 배열 변수의 첨자식, $i_p=j_p$, $i_q>j_q$ $(1 \le p,q \le m)$에서 q의 값이 적을수록 즉, 1에 가까울수록 사용되는 동기화 변수의 수는 증가하지만 EFE기법과 비교할 때 최대 25% 정도의 동기화 변수를 사용하기 때문에 Free/Hold 기법이 더 효율적으로 도익화를 수행할 수 있다.

동기화 명령을 수행하는데 필요한 시간은 해당 시스템에서 사용되는 프로세서(processor)의 수(P)에 의해 결정된다. EFE 기법에서 만약 프로세서의 수에 제한이 없다면 N개의 인스턴스에 대해서 2 단위시간 (단위시간: 하나의 인스턴스를 수행하는데 요구되는 시간)이 소요되고 프로세서의 수에 제한이 있다면 (P=k) 2*(N/k) 단위시간이 소요된다. 그러나 Free/Hold 기법은 일반적으로 종속관계를 갖는 데이타간의 실제 종속거리(RD)에 의해서 동기화 시간이 좌우된다. 즉, 2*(N/RD) 단위시간이 소요된다. 만약 RD>P라면 EFE 기법에 비해 최대 25%정도의 동기화 변수를 가지고 동일한 단위시간 내에 동기화를 수행할 수 있다. 그러나 RD<P라면 사용되는 동기화 변수의 수는 적지만 더 많은 단위시간이 요구된다. 그러나 루프의 상한값이 크고 다중 루프인 경우 RD>P이거나 RD=P가 되기 때문에 Free/Hold 기법이 더 효율적으로 동기화를 수행할 수 있다.

Ⅵ. 결론

본 논문에서는 기존의 동기화 기법인 데이타 지향 기법에서의 동기화 변수에 의한 오버헤드와 문장 지향 기법에서 동기화 명령에 의한 시간 지연을 해결하기 위한 새로운 동기화 기법인 Free/Hold 기법을 제시하였다. 이 기법은 완전 중첩 루프에서 종속거리가 불변인 경우, 최소의 동기화 변수를 이용하여 동기화 명령에 의한 시간 지연을 최소화할 수 있다. 또한 데이타간의 종속거리를 정확하게 계산하기 위한 실제종속거리를 제시하였으며, 일대다 종속관계에서 불필요한 종속관계를 제거하기 위한 기법을 제시하였다. 그러나 데이타간의 종속거리가 가변인 경우와 루프내에 조건문과 같은 제어 종속(control dependence)이 존재하는 경우는 효율적으로 동기화를 할 수 없다. 따라서 앞으로 연구되어야 할 과제는 데이타간의 불변, 가변 종속거리가 동시에 존재하는경우와 제어 종속이 존재하는 루프를 효율적으로 동기화할 수 있는 기법이 연구되어야 한다.

參 考 文 獻

[1] A Dinning, "A survey of Synchronization Methods for Paralled Computers", IEEE Computer pp.66-77, July 1989.

[2] Alfred V. Aho, Jeffery D. Ullman, "Principles of Computer Design", Addison Wesley, pp.408-511, April 1979.

[3] Banergee U., "Speedup of Ordinary Programs", Ph.D Thesis, Univ. of Illionis at Urb.-Champ. Dept. of Comp. Sci., Rpt. No. 79-987, Oct. 1979.

[4] B. J. Smith, "A Pigelined, Shared Resource MIMD Compurter", Proc. conf.on Parallel Processing, Aug. 1978.

[5] Peiyi Tang, Pen-Chung Yew, Chuan-Qi Zhu, "Compiler Techniques for Data sync- hronization in Nested Parallel Loops", Proc. of the ACM Int. Conf. on supe- rcomputing, pp.176-181, July 1990.

[6] Constantine D. Polychronopoulos, "More on Advanced loop Optimization " CSRD Rpt. No. 667, Center for supercomputer Research and Development at Univ. of Illinois, Oct. 1987.

[7] Chuan-Qi Zhu, Pen-chung Yew, "A Scheme to Enforce Data Dependence on Large Multiprocessor System", IEEE Tran. on Software Eng. Vol. SE-13, No. 6, pp.726-739, June 1987.
[8] Uptal Banerjee, "Dependance Analysis for supercomputing", KluwerAcademic Publisher, 1988.
[9] Ding-Kai Chen, Hong-Men Su, Pen-chung Yew, "The Impact of Synchronization and Granularity on Parallel system", CSRD Rpt. No. 942, at Univ. of I11inois May 1990.
[10] Hong-Men su, Pen-Chung Yew, "On Data synchronization forMultiprocessors", Int. Symp. Comp. Architecture, pp. 416-423, May 1989.
[11] Hui Cheng, "Vector Pipelining, Chaining and speed on the IBM 3090 and Cray X-MP", IEEE Computer pp.31-46, sept. 1989.
[12] M. Wolfe, "Multiprocessor Synchronization for concurrent Loops", IEEE Software, Jan. 1988
[13] R. Eigenman, J. Hofflinger, G. Jaxon, D. Padua, "Cedar Fortan and Restructuring Compiler", CSRD Rpt. No. 1041, at Univ. of I11inois, sept. 1991.
[14] S. P. Midkiff, David A. Padua, "Compiler Algorithms for Synchonization", IEEE Tran. on Comp. vol. C-36, No. 12, pp. 1485-1495, Dec. 1987.

著者紹介

李 鑛 炯(正會員)

1988年 순천향대학교 전산학과 졸업(학사). 1991年 고려대학교 전산과학과 대학원 졸업(석사). 1992年 ~ 현재 고려대학교 전산과학과 대학원 박사과정. 주관심 분야는 병렬처리, 컴파일러, 프로그래밍 언어론 등임.

朴 斗 淳(正會員)

1981年 고려대학교 수학과 졸업(학사) 1983年 충남대학교 계산통계학과 졸업(석사) 1988年 고려대학교 대학원(전산학 전공) 졸업(박사). 1992年 ~ 1993年 미국 Univ. of I11inois at Urbana-champaign CSRD객원교수. 1985年 ~ 현재 순천향대학교 전산학과 부교수. 주관심 분야는 병렬처리 컴파일러, 계산이론, 프로그래밍 언어론 등임.

黃 鍾 善(正會員)

1978年 Univ. of georgia, Dept. of Stat. & Computer Science 박사. 1978年 ~ 1980年 미국 South Carolina 주립대학교 조교수. 1980年 ~ 1981年 미국 상무성 연방 표준국 연구위원. 1981年 ~ 1982年 한국표준연구소 전자계산실장. 1986年 ~ 1989年 한국정보과학회 부회장. 1982年 ~ 현재 고려대학교 전산과학과 교수. 주관심 분야는 병렬처리 알고리즘, 인공지능론, 분산처리 등임.

金 秉 洙(正會員)

1981年 서울대학교 자연과학대학 수학과 졸업(학사). 1987年 미국 오클라호마 주립대 전산학과 졸업(석사). 1993年 고려대학교 대학원(전산학전공) 졸업(박사). 1987年 ~ 1991年 한국과학기술연구원 시스템공학연구소 근무. 1991年 ~ 현재 건양대학교 전자계산학과 조교수. 주관심 분야는 병렬처리, 컴파일러, 알고리즘 등임.