論文95-32B-5-4

불변 및 가변 종속거리를 갖는 루프의 병렬처리를 위한 새로운 동기화 기법

(A New Synchronization Scheme for Parallel Processing of Loop with Constant and Variable Dependence Distance)

李 鑛 炯 *, 黃 鍾 善 *, 朴 斗 淳 **

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

요 약

일반적으로 응용 프로그램에서 루프는 전체 수행시간중 상당부분을 차지하는 자원이고 가장 중요한 병렬성 추출의 기본이 된다. 이러한 루프의 반복들이 멀티 프로세서상에서 수행될 때, 각 반복간에 존재하는 데이타 종속은 프로세서간의 동기화에 의해 유지되어야 한다. 기존의 동기화 기법은 주로 종속거리가 불변인 루프에 대해서만 연구되어졌다. 이러한 동기화 기법을 종속거리가 가변인 루프에 적용할 때, 불필요한 동기화 변수의 사용 및 동기화 명령의 수행으로 인한 오버헤드가 발생하게 된다. 비록 몇몇 가변 종속거리 동기화 기법이 제안되고 있지만 동기화 정보를 구성하는데 많은 오버헤드가 존재한다. 본 논문에서는 불변 및 가변 종속거리를 갖는 루프상에서 효율적으로 동기화를 수행할 수 있는 새로운 동기화 기법(Synch-Free Synch-Hold)을 제시한다.

Abstract

In most application programs, loops usually comprise most of the computation in a program and are the most important source of parallelism. When loops are executed on multiprocessors, the cross iteration data dependences need to be enforced by synchronization between processors. Existing synchronization schemes have been studied mainly on the loop with constant dependence distance. When these schemes are applied to the loop with variable dependence distance, there exists lots of overhead by the use of unnecessary synchronization variables and execution of unuseful synchronization instructions. Even though there exist various variable synchronization schemes, they have a lot of run-time overhead to compute synchronization information. In this paper, we present a new synchronization scheme, Synch-Free/Synch-Hold for managing synchronization efficiently on the loop with constant and variable dependence distance.

* 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.)

接受日字: 1994年11月18日, 수정완료일: 1995年5月1日

I. 서 론

루프는 잘 정의된 구조(well-defined structure)로써 잠재적인 병렬성 획득의 가능성이 가장 많다. 또한 응용 프로그램에서 루프는 대부분의 수행시간을 차지하기 때문에 병렬성 추출의 핵심 부분이라 할 수 있다[12-14].

루프의 형태는 모든 반복(iteration)들간에 종속성이 없는 경우(doall)와 서로 다른 반복간에 종속성이 발생하는 경우(doacross)가 존재한다. 루프를 펼쳤을 때 모든 문장 각각을 인스턴스(instance)라 하며 특히, 종속의 근원이 되는 인스턴스를 source 인스턴스, 종속이 되는 인스턴스를 sink 인스턴스라 한다. Doall 형태의 루프에서는 반복간에 종속관계가 존재하지 않기 때문에 한 반복내의 데이타들을 프로세서간의 상호작용없이 병렬처리할 수 있으나 doacross 루프에서는 한 반복에서 가져온 데이타가 다른 반복에서 수정되거나, 한 반복에서 생성된 결과가 나중에 다른 반복에서 사용되는 종속관계가 존재하기 때문에 부분적으로 프로세서간의 정보 교환이 필요하게 된다. 이러한 프로세서간의 상호 정보 교환을 동기화 기법이라 하며 프로그램의 올바른 수행을 위해서 반드시 유지되어야 한다.

동기화 기법은 종속거리의 형태에 따라 불변 종속거리(constant dependence distance) 동기화 기법과 가변 종속거리(variable dependence distance) 동기화 기법으로 나눌 수 있다. 불변 종속거리 동기화 기법은 각 인스턴스간에 일정한 형태의 종속성을 갖는 루프에 대해서는 효율적으로 동기화를 수행할 수 있지만, 인스턴스간에 가변 종속거리가 존재하는 경우에는 불필요한 동기화 변수 및 동기화 명령으로 인한 오버헤드 때문에 효율적으로 동기화를 수행할 수 없다.

또한 배열 변수의 첨자식과 데이타 종속성에 대한 연구[8][11]에 의하면 데이타 종속형태에서 작은 부분만이(13.65%) 불변 종속거리를 갖는다. 즉, 86.35%가 가변 종속거리를 갖는다. 따라서 병렬처리를 위한 효율적인 동기화를 수행하기 위해서는 가변 종속거리가 존재하는 루프에 대해서도 적용할 수 있어야 한다.

가변 종속거리 루프를 동기화하기 위한 기법으로 RDC(Run-time Dependence Checking)[3]와 Synch-read/Synch-write[8] 등이 제안되었지만 동기화 정보를 실행 시에 구하기 때문에 불필요한 시간 지연이 발생하는 등 여러가지 문제점을 갖는다. 논문 [12]에서는 컴파일 시에 동기화 정보를 구하기 위한 기법을 제시했지만 종속 테이블 구성 알고리즘에 많은 시간을 소비하며, 루프 상한값의 크기가 커짐에 따라 종속 테이블 구성을 위한 메모리의 낭비가 심하게 된다.

본 논문에서는 불변 및 가변 종속거리가 동시에 존재하는 경우에도 효율적으로 수행할 수 있는 새로운 동기화 기법(Synch-Free/Synch-Hold)을 제시한다. 이 기법은 컴파일 시간에 종속 정보를 구할 수 있기 때문에 실행시의 오버헤드가 존재하지 않으며, 종속 정보를 구하기 위한 연산이 간단하기 때문에 효율적으로 동기화를 수행할 수 있다.

본 논문의 구성은 다음과 같다.

2절에서 데이타 종속성을 정의하고, 3절에서는 데이타 종속거리 형태를 기준으로 기존의 동기화 기법들의 장단점을 파악한다. 4절에서는 기존의 동기화 기법에서 발생하는 문제를 해결하기 위한 새로운 동기화 기법을 제시한다. 5절에서는 성능 평가를 위해 종속 정보를 구하기 위한 시간 복잡도(time complexity)를 이용하여 비교한다.

II. 종속성과 종속거리

하나의 프로그램이 수행되는 과정은 일련의 문장들이 정해진 순서에 따라 작업을 수행하는 것이다. 각 문장에서는 상수 또는 이미 정의된 변수들을 사용하여 새로운 값을 계산하고 그 결과를 변수에 저장한다. 또한 그 값은 다음에 수행되는 문장에 의해서 다시 사용되기도 한다.

변수의 정의와 사용관계에 따라 몇 문장들은 병렬수행이 가능하나 몇 문장들은 반드시 임의의 순서에 입각하여 수행되어야 하는 경우도 있다. 후자와 같이 반드시 순서대로 수행되어야만 하도록 만드는 성질들을 종속성이라 한다. 따라서 두 개의 서로 다른 데이타 또는 문장들이 병렬로 수행될 수 있는 가는 이들간의 종속성 유무에 따라 결정된다.

[정의 2.1] 사전식 수행 순서[16]

m차 중첩 루프내의 문장, $S_p(i_1, i_2, \ldots, i_m)$, $S_q(j_1, j_2, \ldots, j_m)$ 에서,

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

② $i_1 = j_1, \ldots, i_m = j_m$, 이고 $p < q$ 일 때

문장 S_p는 S_q보다 먼저 수행되는 것을 의미하고, $S_p \ll S_q$로 나타내며 이를 문장의 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 할 때, 다음과 같이 세가지의 종속성, 흐름

종속성(flow dependence), 반 종속성(anti dependence) 및 출력 종속성(output dependence)을 정의할 수 있다[9][10].

[정의 2.2] 데이타 종속성[9][10]

i) $S_p << S_q$ 이고 $OUT(S_p) \cap IN(S_q) \neq \varnothing$ 일 때 S_q 는 S_p에 흐름 종속이 된다고 하고 $S_p \delta S_q$ 로 표시한다.

ii) $S_p << S_q$ 이고 $IN(S_p) \cap OUT(S_q) \neq \varnothing$ 일 때 S_q 는 S_p 에 반 종속이 된다고 하고 $S_p \delta'' S_q$ 로 표시한다.

iii) $S_p << S_q$ 이고 $OUT(S_p) \cap OUT(S_q) \neq \varnothing$ 일 때 S_q 는 S_p에 출력 종속이 된다고 하고 $S_p \delta'' S_q$로 표시한다.

데이타 종속거리(dependence distance)는 동기화 기법에서 병렬처리를 위한 동기화 명령을 효율적으로 처리할 수 있는 유용한 정보로써, 데이타 종속을 이루는 두개의 인스턴스사이에 어느 정도의 종속성이 존재하는 지를 알 수 있다.

종속관계를 갖는 두 문장 S_p와 S_q가 동일한 배열 변수를 포함하며 변수의 첨자식이 각각 a×i+b, c×j+d(여기서 a, b, c, d는 정수, i, j는 루프 변수)와 같은 선형 결합 첨자(linear coupled subscript)를 갖는다면 두 문장사이의 종속거리, $D(S_p, S_q)$는 다음과 같이 정의할 수 있다.

$$D(S_p, S_q) = \begin{cases} |d-b| & \text{if } a = c \\ \textit{variable} & \text{if } a \neq c \end{cases}$$

즉, 종속거리는 종속관계를 갖는 source, sink 인스턴스간의 반복의 거리를 의미한다.

Ⅲ. 관련 연구

불변 종속거리 동기화 기법은 종속을 이루는 각 데이타의 모든 인스턴스의 종속거리가 일정한 경우에 적용하는 방법으로 병렬성을 추출하는 크기에 따라 데이타 지향 기법(data-oriented scheme : Full/Empty tag 기법[7]), 문장 지향 기법(statement-oriented scheme : Lock/Unlock, Advance/Wait[11], [7]에서 제안한 Set/Wait, Testset/Test 기법) 등으로 나눌 수 있다[5]. 이러한 불변 종속거리 동기화 기법은 가변 종속거리를 갖는 루프에 적용할 때 많은 오버헤드가 존재한다. 즉, 종속관계를 갖는 각 데이타의 모든 인스턴스들은 일정한 간격으로 종속성이 존재하지 않기 때문에 종속관계가 없는 인스턴스에도 동기화 정보를 유지하게 된다. 따라서 불필요한 동기화 명령을 수행함으로써 시간 지연이 발생하게 된다.

이에 따라 종속이 있는 인스턴스에만 동기화 정보를 전달하는 가변 종속거리 동기화 기법이 제안되었다.

RDC(Run-time Dependence Checking)[3]의 기본적인 방법은 루프내의 특정 반복이 하나 또는 그 이상의 반복에 종속이 되는가를 실행 시에 결정한다. 이에 따라 동기화가 필요한 반복에 대해서만 동기화 명령을 수행함으로써 가변 종속거리 루프를 병렬처리할 수 있다. 그러나 이 기법은 흐름 종속일 경우에만 적용 가능하고 모든 종속성 정보를 실행 시에 얻기 때문에 이전 반복의 수행에 관한 정보를 유지해야만 한다. 또한 동기화 정보를 유지하기 위하여 루프의 상한값에 해당하는 dependence source vector, R 와 synchronization vector, V 가 필요하다. 즉, 각 데이타의 모든 인스턴스에 대해 두가지 동기화 변수를 사용한다. 따라서 m차 중첩 루프의 경우, 사용되는 동기화 변수의 수는 $2\prod_{i=1}^{m} N_i$ (N_i : 각 루프의 상한값)가 된다.

논문 [8]에서는 루프의 각 데이타의 모든 인스턴스에 대한 순차 접근 순서(sequential access order)를 이용하여 데이타의 종속성 유지를 위한 동기화 기법(synch-read/synch-write)을 제시하였다. 그러나 이 기법은 임의의 인스턴스를 수행하기 전에 해당 인스턴스에 대한 쓰기-접근(write access)을 위한 접근 순서와 읽기-접근(read access)을 위한 접근 순서를 계산하여야 한다. 이러한 접근 순서를 실행 시에 계산하기 때문에 오버헤드를 초래한다. 또한, 종속성이 없는 인스턴스(이때의 접근 순서는 음수의 값을 갖는다)에 대해서도 접근 순서를 계산해야 하기 때문에 불필요한 시간 지연이 발생하게 된다.

논문 [12]에서는 종속성이 있는 인스턴스에만 동기화 정보를 전달하고 불필요한 동기화 정보를 제거하기 위한 기법으로 종속 테이블 기법을 제안하였다. 이 기법은 m차 중첩 루프의 경우, 종속 테이블을 구성하기 위해 $\prod_{i=1}^{m} N_i \times \prod_{i=1}^{m} N_i$ 크기의 행렬이 요구되기 때문에 과도한 메모리의 사용으로 인한 오버헤드가 존재한다. 또한 불필요한 동기화 명령을 제거하기 위해서는 위와 동일한 크기의 테이블을 이용해야 하기 때문에 많은 시간 지연이 발생하게 된다.

Ⅳ. 새로운 동기화 기법

본 절에서는 데이타 종속거리가 불변 또는 가변적으

로 발생하는 루프에서 기존의 동기화 기법의 오버헤드를 최소화할 수 있는 새로운 동기화 기법(Synch-Free/Synch-Hold)을 제안한다.

이 기법은 종속 정보를 컴파일 시에 구성하기 때문에 실행 시에 동기화 정보를 구하기 위한 오버헤드가 존재하지 않는다. 또한 종속관계를 갖는 인스턴스에 대해서만 동기화 정보를 유지하기 때문에 불필요한 동기화 연산을 줄일 수 있다.

1. Synch-Free/Synch-Hold 동기화 기법

불변 및 가변 종속거리에서의 동기화를 위한 기본적인 방법은 먼저 종속 데이타의 모든 인스턴스중에서 첫번째로 종속관계를 갖는 인스턴스(첫번째 source 인스턴스와 sink 인스턴스)를 구한 후 source 인스턴스 바로 뒤에 수행 완료를 나타내는 Synch-Free 동기화 명령을 삽입하고, sink 인스턴스 바로 전에 해당 source 인스턴스의 수행이 완료되기까지 인스턴스의 수행을 대기하게 하는 Synch-Hold 동기화 명령을 삽입한다. 위에서 첫번째 인스턴스의 값은 종속 방정식[6]을 이용하면 구할 수 있다. 두번째로, 첫번째 종속관계를 갖는 인스턴스 이후의 종속관계를 구하기 위하여 해당 데이타의 증가 계수를 구한다.

[정의 4.1] 증가 계수(Increment Factor)
종속관계를 갖는 두 데이타에서, 첫번째 source, sink 인스턴스 이후의 다음 종속위치는 각각 일정한 상수(ρ, ρ') 에 의해 결정되는데, 이때 이 상수값을 증가 계수라 한다.

이 증가 계수는 종속관계를 갖는 두 데이타의 첨자식이 각각 $a\times i+b$, $c\times j+d$ 와 같은 선형 결합 첨자(linear coupled subscript)에서 루프 변수의 계수(coefficient)를 이용하여 구할 수 있다.

알고리즘 4.1 증가 계수 알고리즘

```
Input : a, c      /* 루프 변수의 계수 */
Output : ρ, ρ'   /* source, sink 데이타의 증가 계수 */
Method :
        ρ = a; ρ' = c;
label : g = GCD(ρ, ρ');   /* GCD = Greatest Common
                             Divisor) */
        if (g == 1)        /* 증가 계수 */
          then
            exit;
          else
            ρ = ρ / g;
            ρ' = ρ' / g;
            goto label;
        end if
```

[정리 4.1]
불변 및 가변 종속관계를 갖는 source, sink 데이타의 종속거리는 각각 sink 데이타의 증가 계수, source 데이타의 증가 계수씩 증가한다.

〈증명〉

source, sink 데이타의 배열 첨자식을 각각 $a\times i+b$, $c\times j+d$ 라 하고 source와 sink 인스턴스간 첫번째 종속관계를 $\alpha \fallingdotseq \beta$ 로 표시한다고 하자. 그러면 항상 source, sink 인스턴스사이의 종속성은 $\alpha+n\times\rho' \fallingdotseq \beta+n\times\rho$ 임을 수학적 귀납법을 이용하여 증명하면 된다($n\geq 0$).

i) $n = 0$ 인 경우, $\alpha+n\times\rho' \fallingdotseq \beta+n\times\rho$ 로 부터
$\alpha \fallingdotseq \beta$ ——— ① /* 첫번째 종속관계 */

ii) $n = k$ 일 때 성립한다고 가정하면,
$\alpha+k\times\rho' \fallingdotseq \beta+k\times\rho$ ——————— ②

②에서, α와 β는 첫번째 종속관계를 나타내기 때문에 다음 종속관계는 $k\times\rho'$ 와 $k\times\rho$ 에 의해서 결정된다.

즉, $k\times\rho' \fallingdotseq k\times\rho$ ——— ③

$n = k+1$ 일 때

$$\begin{aligned}\alpha+(k+1)\times\rho' &= \alpha+k\times\rho'+\rho' \\ &\Rightarrow \beta+k\times\rho+\rho' \ (\because ②) \\ &\Rightarrow \beta+k\times\rho'+\rho' \ (\because ③) \\ &\Rightarrow \beta+(k+1)\times\rho' \\ &\Rightarrow \beta+(k+1)\times\rho \ (\because ③)\end{aligned}$$

$\therefore \alpha+(k+1)\times\rho' \fallingdotseq \beta+(k+1)\times\rho$ ■

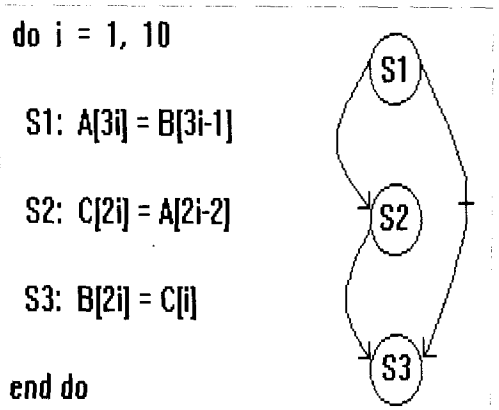
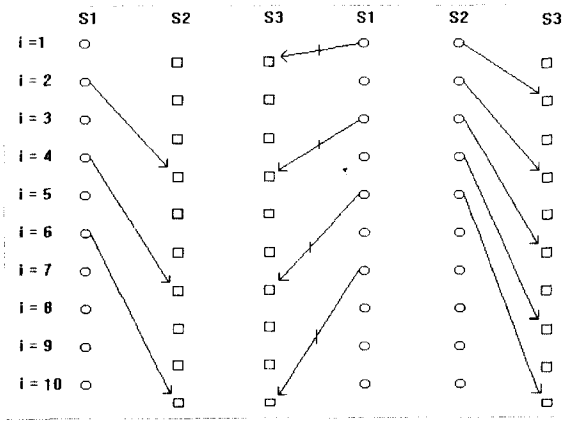


그림 1. 예제 루프 및 종속 그래프
Fig. 1. an example loop and dependencen graph.

그림 1의 루프에서는 세개의 종속관계 즉, $S1\,\delta\,S2$, $S1\,\delta^{a}S3$ 그리고 $S2\,\delta\,S3$가 존재한다. 배열 변수 A의

종속관계에서 S1은 source 데이타가 되고 S2는 sink 데이타가 되며, 첫번째 source 인스턴스의 위치는 루프 변수 i가 2일 때(A[6]), sink 인스턴스의 위치는 i가 4일 때(A[6]) 첫번째 종속관계가 성립한다. 또한 source 데이타의 증가 계수는 3이고 sink 데이타의 증가 계수는 2가 되기 때문에 〈그림 4.2〉의 (a)에서 보는 바와 같이 첫번째 종속관계 이후의 source 인스턴스의 종속거리는 2씩 증가하고 sink 인스턴스의 종속거리는 3씩 증가한다.

그림 1의 종속 그래프에서 ──〉는 흐름 종속, —|-〉는 반종속을 의미한다.



(a) A의 종속관계 (b) B의 종속관계 (c) C의 종속관계

그림 2. 각 배열 변수의 인스턴스에 대한 종속관계
Fig. 2. dependence relationships between instances of each variable.

[정리 4.1]를 이용하여 불변 및 가변 종속거리를 갖는 루프에 올바른 동기화 명령을 삽입하기 위한 알고리즘은 다음과 같다. 이 알고리즘에서 사용하는 전체문장번호(unrolled_statement_number)는 모든 루프를 펼쳤을 때 처음부터 각각의 모든 인스턴스에 붙인 번호를 의미한다.

그림 2의 배열 변수 A에 대한 모든 인스턴스간의 종속관계는 $4\delta 11$, $10\delta 20$, $16\delta 29$, 배열 변수 B에 대한 모든 인스턴스간의 종속관계는 $1\delta^a 3$, $7\delta^a 12$, $13\delta^a 21$, $19\delta^a 30$ 그리고 배열 변수 C에 대한 모든 인스턴스간의 종속관계는 $2\delta 6$, $5\delta 12$, $8\delta 18$, $11\delta 23$, $14\delta 30$의 형태(여기서 정수값은 전체문장번호를 의미)로 종속관계가 성립한다. 따라서 [알고리즘 4.2]에 의해 배열 변수 A의 전체문장번호 4,10,16 바로 뒤에 Synch-Free 명령어를 삽입하고 11,20,29 바로 전에 Synch-Hold 명령어를 삽입한다.

알고리즘 4.2 불변/가변 종속거리에 대한 동기화 알고리즘

```
* I = unrolled_statement_number of the first source instance
* J = unrolled statement_number of the first sink instance
* T = number of total statement in a loop
* ρ = increment factor of the source data
* ρ' = increment factor of the sink data
* N = upper bound of a loop
■ Insertion of Synch-Free and Synch-Hold instructions
  foreach (K=I,K'=J; K'≤N×T;)  /* N×T:전체 인스턴스 수 */
  { Insert Synch-Free instruction immediately after K;
    Insert Synch-Hold instruction immediately before K';
      K =K+T×ρ';        /* source 인스턴스의 위치 */
      K'=K'-T×ρ';}      /* sink 인스턴스의 위치 */
```

[알고리즘 4.2]에 의해 재구조화된 루프의 올바른 수행 순서를 유지하기 위한 동기화 연산은 다음과 같다.

알고리즘 4.3 불변/가변 종속거리 동기화 연산

```
* X(j) = synchronization variables /* j는 동기화 변수의 번호 */
■ Synch-Free(K,X(j))
  foreach (R=I,R'=J,j=1; R'≤N×T; R=R+T×ρ',R'=R'+T×ρ',j++)
    if(K == R)          /* source 인스턴스의 수행 여부 점검 */
    X(j) = R';          /* source 인스턴스의 수행 완료 표시 */
■ Synch-Hold(K', X(j))
  wait until (K' == X(j));   /* source 인스턴스의 완료 여부 점검 */
```

위의 Synch-Free 연산에서, 조건문은 현재 수행한 인스턴스가 source 인스턴스인지를 판별하며, 참(true)인 경우 동기화 변수에 sink 인스턴스의 수행을 위해 해당 sink 인스턴스의 전체문장번호를 대입한다. 또한 Synch-Hold 연산에서는 해당 동기화 변수의 값과 현재 수행하고자 하는 sink 인스턴스의 전체문장번호가 같을 때까지(즉, 해당 source 인스턴스의 수행이 완료될 때까지) 대기하게 한다. 동기화 변수 X(j)에 대해서 각각의 쌍(source 인스턴스, sink 인스턴스)은 하나의 동기화 변수를 공유한다.

[알고리즘 4.2]를 이용하여 〈그림 4.1〉를 재구조화시킨 루프는 〈그림 4.3〉과 같다(여기서는 배열 변수 A에 대해서만 표현).

```
반복    전체문장번호
I = 1     1        S1
          2        S2
          3        S3
I = 2     4        S1
                   Synch-Free(4,X(1))
          5        S2
          6        S3
I = 3     7        S1
          8        S2
          9        S3
I = 4     10       S1
                   Synch-Free(10,X(2))
                   Synch-Hold(11,X(1))
          11       S2
          12       S3
I = 5     13       S1
          14       S2
          15       S3
I = 6     16       S1
                   Synch-Free(16,X(3))
          17       S2
          18       S3
I = 7     19       S1
                   Synch-Hold(20,X(2))
          20       S2
          21       S3
I = 8     22       S1
          23       S2
          24       S3
I = 9     25       S1
          26       S2
          27       S3
I = 10    28       S1
                   Synch-Hold(29,X(3))
          29       S2
          30       S3
```

그림 3. 그림 4.1의 재구조화된 루프
Fig. 3. the restructured loop of fig 4.1.

그림 3에서 전체문장번호가 4인 문장 S1의 수행이 완료되었다면, 이는 source 인스턴스가 되기 때문에 동기화 변수 X(1)에 해당 sink 인스턴스의 전체문장번호인 11을 대입한다. 또한 전체문장번호가 11인 sink 인스턴스를 수행할 때는, 우선 해당 동기화 변수(X(1))의 값이 11이 될 때까지 대기하다가 동일한 값이 되면 수행을 시작한다.

또한 그림 2의 (a)에서 보는 바와 같이 배열 변수 A의 첫번째 source 인스턴스 이전의 인스턴스 즉, i=1일 때의 인스턴스들은 종속관계를 갖지 않고 마지막 source 인스턴스 이후의 인스턴스 즉, i=6일 때의 S1 이후의 인스턴스들과 i=7,8,9,10일 때의 인스턴스들은 종속관계를 갖지 않는다. 따라서 이러한 인스턴스들은 임의적으로 병렬처리할 수 있기 때문에 병렬성 가능성을 크게 하기 위해 추출할 필요가 있다.

[정의 4.2] 비-종속성(non-dependence) 인스턴스

종속관계를 갖는 첫번째 source 인스턴스와 sink 인스턴스 이전의 인스턴스들은 종속성을 갖지 않으며, 마지막 source, sink 인스턴스 이후의 인스턴스들은 루프의 상한값에 의해 종속성을 갖지 않게 되는데, 전자를 전면(front) 비-종속성 인스턴스, 후자를 후면(rear) 비-종속성 인스턴스라 한다.

이러한 비-종속성 인스턴스들은 종속관계를 갖는 첫번째 그리고 마지막 종속 인스턴스의 부분문장번호(partial_statement_number)를 이용하여 구할 수 있다. 여기서 부분문장번호는 특정 한 레벨의 루프에서, 루프를 펼쳤을 때 각각의 모든 인스턴스에 붙인 번호를 의미한다.

Φ(전면 비-종속성 인스턴스의 수) = χ-1

Ψ(후면 비-종속성 인스턴스의 수) = (N×T)-χ'

위의 식에서 χ와 χ'는 각각 첫번째 그리고 마지막 종속 인스턴스의 부분문장번호를 의미하고 N은 루프의 상한값, T는 루프내의 문장의 수를 나타낸다.

그림 2의 (a)에서 배열 변수 A에 대한 χ=4, χ'=16, N=10 그리고 T=3이 되기 때문에 Φ=4-1=3, Ψ=10×3-16=14가 된다.

m차 중첩 루프에서, 동기화 명령을 삽입하기 위한 방법은 [알고리즘 4.2]와 동일하지만 m 레벨 루프의 수행이 끝난 후 m-1 레벨 루프의 종속거리는 해당 루프의 데이타의 증가 계수에 따라 다음 종속거리가 결정된다. 그리고 m-1 레벨상에서 첫번째 source 인스턴스와 sink 인스턴스를 구하기 위해서는 [정의 4.2]의 비-종속성 인스턴스의 수가 필요하다.

```
do i1 = 1, 10
  do i2 = 1, 10
    A[3i1,2i2] = . . .
    . . . = A[2i1,i2-2]
  end do
end do
```

그림 4. 2차 중첩 루프의 예
Fig. 4. an example of 2-nested loop.

그림 4의 루프에서, 각 인스턴스간의 종속관계를 표현하면 그림 5와 같다.

그림 5에서, source 인스턴스는 i_1=2, i_2=1일 때(A[6,2]), sink 인스턴스는 i_1=3, i_2=4일 때(A[6,2]) 첫번째 종속관계가 성립한다. 따라서 m 레벨의 루프에 대응되는 source 데이타의 첨자식($2i_2$)의 증가 계수는 2이고 sink 데이타의 첨자식(i_2-2)의 증가 계수는 1이기 때문에 source 인스턴스의 종속거리는 1씩 증가하고 sink 인스턴스의 종속거리는 2씩 증가한다(m 레벨 루프(i_2)의 상한값까지). 그리고 m-1 레벨의 루프에 대응되는 source 데이타의 첨자식($3i_1$)의

증가 계수는 3이고 sink 데이타의 첨자식($2i_1$)의 증가 계수는 2가 되기 때문에 m 레벨 루프의 상한값 이후의 첫번째 source 인스턴스는 $i_1=4, i_2=1$일 때(A[12,2]), 첫번째 sink 인스턴스는 $i_1=6, i_2=4$일 때(A[12,2]) 종속관계가 성립한다.

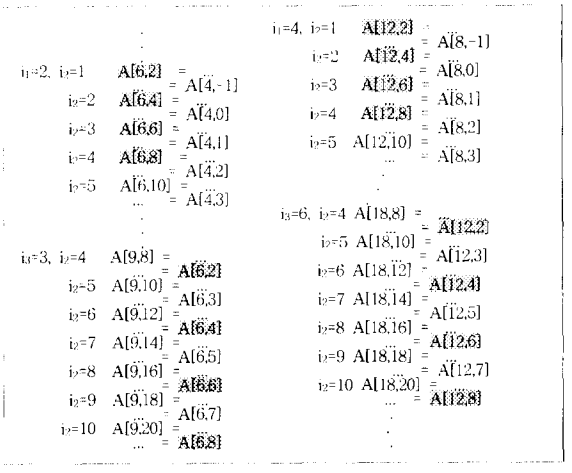


그림 5. 그림 4.4의 인스턴스들의 종속관계
Fig. 5. dependence relationships between instances of fig. 4.4.

따라서 2차 이상의 중첩 루프에서, 동기화 명령어를 삽입하기 위한 알고리즘은 현재 수행하고자 하는 인스턴스가 해당 레벨의 루프 상한값내에 존재하는지를 결정해야 한다. 즉, 다음과 같은 조건문이 필요하다.

```
S = ⌈ K' div (N_i × T) ⌉
if S ≠ I_i1
  then 루프의 상한값 벗어남
  else 루프의 상한값내에 존재
여기서, I_i(1≤i≤m)는 루프 변수의 값을 의미
```

위의 그림 5에서, 전체문장번호가 60인 sink 인스턴스(K')를 수행할 때 각 루프 변수의 값은 i1=3, $i_2=10$이 된다. 이 K'은 m 레벨 루프의 상한값을 벗어나지 않음으로 다음 전체문장번호는 [알고리즘 4.2]에 의해서 K'=64가 된다. 그러나 이때의 sink 인스턴스는 S = ⌈ 64 div (10 × 2) ⌉ = 4가 된다. 즉, 각 루프 변수의 값은 $i_1=3, i_2=11$이 되어 현재 루프 레벨(i_2)의 상한값을 벗어나게 된다. 이와 같이 범위를 벗어나는 경우, 다음 종속관계를 갖는 source, sink 인스턴스의 전체문장번호(K,K')는 다음과 같다.

```
K = K + ((ρ'_i1 - 1) × N_i × T) + Φ + Ψ + | q - p |
K' = K' + ((ρ_i1 -1) × N_i × T) + Φ + Ψ + | q - p |
단, p, q는 루프내에서 종속관계를 갖는 두 문장의 번호
```

그림 5에서, $i_1=2, i_2=5,6,7,8,9,10$ 상의 source 인스턴스들은 해당 sink 인스턴스들이 루프의 범위를 벗어나기 때문에 비-종속성 인스턴스가 되고, $i_1=3$, $i_2=1,2,3,4$ 상의 sink 인스턴스들은 해당 source 인스턴스가 존재하지 않기 때문에 비-종속성 인스턴스가 된다. 따라서 source 인스턴스에 대한 $\Phi=0$, $\Psi=13$, sink 인스턴스에 대한 $\Phi=7$, $\Psi=0$ 이 된다. 또한 하위 레벨 루프(i_1)의 source 데이타의 증가 계수(ρ_{i_1})는 3 그리고 sink 데이타의 증가 계수(ρ'_{i_1})는 2가 되고 상한값을 벗어나지 않는 마지막 source 및 sink 인스턴스의 전체문장번호는 각각 27(K), 60(K')이 되기 때문에 다음 종속관계를 갖는 source 및 sink 인스턴스의 전체문장번호는 다음과 같다.

$$K = 27 + (1\times10\times2) + 0 + 13 + 1 = 61$$

$$K' = 60 + (2\times10\times2) + 7 + 0 + 1 = 108$$

V. 성능 평가

기존의 동기화 기법과 본 논문에서 제시하는 기법를 비교하기 위하여 데이타 종속을 유지하기 위한 정보를 구성하는데 필요한 시간 복잡도(time complexity)를 이용한다. 그러나 불변 종속거리 동기화 기법은 종속거리가 가변일 때 동기화를 효율적으로 수행할 수 없기 때문에 본 논문에서는 synch-read/synch-write 기법과 종속 테이블 기법과 비교한다.

논문 [8]에서 접근 정보를 계산하기 위한 시간 복잡도(time compexity)는 다음과 같다.

◈ $O(\frac{1}{2}kn^2+kn+km)$: 쓰기-접근을 위한 시간 복잡도

◈ $O(\frac{3}{2}kn^2+3kn+2km)$: 읽기-접근을 위한 시간 복잡도

여기서 k : 해당 데이타의 참조(reference) 횟수
n : 해당 데이타의 중첩 정도
m : 루프의 중첩 정도

또한 논문 [12]에서 제시한 방법에서 종속 테이블

구성은 루프의 상한값에 좌우되기 때문에 $\pi_N \cdots \pi_N$ 행렬이 필요하고 종속 정보를 구하기 위해 $O(N^{2m})$이 요구된다.

본 논문에서 제시하는 Synch-Free/Synch-Hold에서 필요로 하는 종속성 정보는 첫번째 종속관계를 갖는 source 인스턴스, sink 인스턴스(I,J) 및 종속 데이타의 증가 계수(ρ, ρ')를 이용한다. 즉, I와 J를 구하는데 $O(m)$, ρ와 ρ'를 구하는데 $O(k)$이 된다.

즉, 각 동기화 기법에서 종속 정보를 구하기 위한 시간 복잡도는 다음과 같다.

$$O(m)+O(k) < O(\frac{1}{2}kn^2+kn+km)+$$

$$O(\frac{3}{2}kn^2+3kn+2km) < O(N^{2m})$$

따라서 본 논문에서 제시하는 동기화 기법이 기존의 동기화 기법보다 더 효율적으로 동기화를 수행할 수 있다.

VI. 결 론

동기화 기법은 문장간의 종속거리에 따라 크게 불변 종속거리 동기화 기법과 가변 종속거리 동기화 기법으로 나눌 수 있다. 그러나 불변 및 가변 종속거리를 갖는 루프를 동기화 할 때, 불변 종속거리 기법은 불필요한 동기화 명령을 수행함으로써 시간 지연이 발생하고, 가변 종속거리 기법은 종속성 유지를 위한 과도한 오버헤드가 존재한다. 이에 따라 본 논문에서는 기존의 동기화 기법에서 발생하는 문제점을 해결하기 위해서 효율적인 동기화를 수행할 수 있는 새로운 동기화 기법(Synch-Free,Synch-Hold)을 제시하였다.

이 기법은 종속 정보를 컴파일시에 구성하고, 종속관계를 갖는 인스턴스에 대해서만 동기화 정보를 유지하기 때문에 실행시의 오버헤드를 최소화할 수 있으며 종속관계를 갖지 않는 인스턴스의 동기화 연산을 수행하지 않기 때문에 불필요한 연산을 줄일 수 있다.

앞으로 연구되어야 할 과제는 제어 종속이 존재하는 루프를 효율적으로 동기화할 수 있는 동기화 기법 및 자원을 효율적으로 분배하기 위한 스케쥴링 기법 등이 연구되어야 할 과제이다.

참 고 문 헌

[1] Alliant FX/Series Architecture Manual, Alliant Computer Systems Corp. 1989.

[2] A. Aiken, A. Nicolu, "Optimal Loop Parallelization", Proc. of the SIGPLAN '88 Conference on Programming Language Design and Implementation", ACM pp.308-317 1988.

[3] C. D. Polychronopoulos, "More on Advanced Loop Optimization", CSRD Report No. 667, CSRD at Univ. of Illinois, Oct. 1987.

[4] D. J. Kuck et al., "The Effect of Program Restructuring, Algorithm Change and Architecture Choice on Program Performance", In Proc. Intl. Conf. on Paralle Processing, pp.129-138, Aug. 1984.

[5] H. M. Su, P. C. Yew, "On Data Synchronization for Multiprocessor", Proc. of the Annual Intl. symposium on Computer Architecture, pp.416-423, May 1989.

[6] H. Zima, B. Chapman, "Supercompiler for Parallel and Vector Computers", ACM Press Addison Wesly Publishing Company, 1991.

[7] S. P. Midkiff, D. A. Padua, "Compiler Algorithms for Synchronization", IEEE Tran. on Computer Vol. C-36, No. 12, pp.1485-1495, Dec. 1987.

[8] P. Tang, P. C. Yew, C. Q. Zhu, "Compiler Techniques for Data Synchronization in Nested Parallel Loops", Proc. of the ACM Intl. Conference on Supercomputing, pp.176-181, July, 1990.

[9] U. Banerjee, "Dependence Analysis for Supercomputing", Kluwer Academic Publish 1988.

[10] Z. Li, P. C. Yew, C. Q. Zhu, "An Efficient data Dependence Analysis for Parallelizing Compilers", IEEE Trans. on Parallel and Distributed Systems, Vol. 1, No. 1, pp.26-34, Jan. 1990.

[11] Z. Shen, Z. Li, P. C. Yew, "An Empirical Study on Array Subscripts and Data Dependencies", Proc. of the Intl. Conference on Parallel Processing, pp.145-152, Aug. 1989.

[12] 정기동, 정성인, "다중처리기 시스템에서 최적화된 병렬 루프 변환 기법", 정보과학회논문지 제18권 제1호, 1991. 1월

저 자 소 개

李 鑛 炯(正會員)

1988년 순천향대학교 전산학과 졸업(학사). 1991년 고려대학교 전산과학과 대학원 졸업(석사). 1992년 ~ 현재 고려대학교 전산과학과 대학원 박사과정. 주관심분야는 병렬/분산처리 시스템, 컴파일러, 프로그래밍 언어론 등임.

朴 斗 淳(正會員)

1981년 고려대학교 수학과 졸업(학사). 1983년 충남대학교 계산통계학과 졸업(석사). 1988년 고려대학교 대학원(전산학 전공) 졸업(박사). 1992년 ~ 1993년 미국 Univ. of Illinois at Urbana-Champaign CSRD 객원교수. 1985년 ~ 현재 순천향대학교 전산학과 부교수. 주관심분야는 병렬처리 컴파일러, 계산이론 프로그래밍 언어론 등임.

黃 鍾 善(正會員)

1978년 Univ. of Georgia, Dept. of Stat. & Computer Science 박사. 1978년 ~ 1980년 미국 South Carolina 주립대학교 조교수. 1980년 ~ 1981년 미국 상무성 연방 표준국 연구위원. 1981년 ~ 1982년 한국표준연구소 전자계산실장. 1986년 ~ 1989년 한국정보과학회 부회장. 1982년 ~ 현재 고려대학교 전산과학과 교수. 1995년 ~ 현재 한국정보과학회 회장. 주관심분야는 병렬처리 알고리즘, 인공지능론, 분산처리 등임.