論文 92－29B－10－1

MIMD 시스템에서의 효율적인 중복 동기화명령어 제거 기법

(Effective Elimination Methods of Redundant Synchronization Instructions in MIMD Systems)

金 秉 洙,* 黃 鍾 善,** 朴 斗 淳***

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

要 約

본 논문에서는 기존의 동기화 기법들과는 달리 종속의 성질에 따라 적절한 동기화명령어를 문장 사이에 삽입하는 동기화 기법에 관하여 논하였으며, 가변 종속에서의 동기화 기법도 아울러 논하였다. 또한 중복 동기화명령어들의 효율적인 제거 방법을 개발하여 기존의 CPG 방법과 비교, 월등한 성능향상을 보였다.

Abstract

This paper presents an effective synchronization algorithm. It is different from the existing synchronization methods by inserting appropriate synchronization instructions between statements according to different kinds of data dependencies. The overhead caused by too many synchronization instructions in a loop can be a critical problem. Synchronization optimization is a method which discriminates and eliminates the redundant synchronization instructions in a loop. In this paper, a new synchronization optimization algorithm is developed, and performance analysis using simulation on the UNIX operating system is carried out.

I. 서 론

최근 들어 처리해야 할 자료의 양이 많아짐에 따라, 이러한 자료들을 병렬처리하기 위한 하드웨어 기술[6, 8]들이 급속도로 발전해 왔다. 이에 따라 병렬처리 하드웨어에 적합한 소프트웨어 기술도 많이 연구되고 있고, 이중에도 특히 기존의 순차 프로그램을 병렬 처리 시스템에서 직접 실행시키는 방법들이 많이 연구되고 있다.[2, 3, 11, 12]

주어진 순차 프로그램을 병렬화 하는데 있어서 가장 핵심 부분은 루프이다. 그러나 순차 프로그램내에는 이와 같은 방법들을 적용할 수 없는 루프가 상당히 많이 존재할 수 있으며,또한 적용할 수 있다 하더라도 문장 재구성(restructuring) 기법에 소요되는 오버헤드(overhead)가 상당히 커질 수 있어 비 효율적인 경우가 많다.

동기화 (synchronization) 기법은 이러한 결점들을 보완하는 방법으로, 이 논문에서는 주로 공유 메모리(shared memory)를 가지는 밀착결합 MIMD 시스템 내에서의 동기화 기법에 대하여 논하고자 한다. MIMD 시스템들이 이용하는 동기화 기법으로는 semaphore,

*正會員, 建陽大學校 電子計算學科
(Dept. of Computer Science, Keonyang Univ.)
**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 : 1992年 2月 25日

monitor, message passing[4] 등이 있으며, 여기에서는 공유 메모리 시스템에서 널리 쓰이는 semaphore 형태의 동기화에 대하여 논하겠다.

실제로 MIMD 시스템에서 구현되고 있는 동기화 명령어 기법은 여러가지가 있다.[1, 4, 5, 9, 10, 12] 그러나, 이들은 각 종속(dependence)의 거리(distance)가 불변이라고 가정하고 있으며, 흐름(flow)종속의 경우에 대하여만 주로 논하고 있다. 따라서 본 논문에서는 반(anti)종속에 대하여도 적용할 수 있는 동기화 기법을 제안하였으며, 종속 거리가 가변인 경우에도 고려하였다. 본 논문에서는 이미 제안된 여러 동기화 기법과 유사한 "busy waiting" 형식의 구현 방식을 쓰는 Release & Wait 라는 동기화 명령어를 제안하고자 한다.

또한, 단일 루프내의 중복되는(redundant) 동기화 제거 기법에 대하여 기존의 방법들[10, 12]보다 훨씬 효율이 좋은 알고리듬을 제시하며, 가변 종속거리에 대한 중복 동기화 제거에 대하여도 논하고자 한다.

Ⅱ. 단일 루프내의 동기화 기법

1. 제안된 동기화 명령어 쌍(tuple)

공유(shared) 메모리를 이용하는 MIMD 시스템내의 기존의 동기화 명령어들은 주로 semaphore 를 근간으로 한 busy waiting 형식의 구현 방법을 쓰고 있다. 이런 동기화 기법을 사용하기 위해서는 먼저 자료 종속성(data dependency) 관계를 알아야 한다. 자료 종속성 관계에는 흐름종속, 반종속, 출력종속(output dependence) 등이 있으며 종속관계를 가지는 문장과 문장 사이의 거리를 종속거리(d)라 한다.[9, 11]

기존에 연구된 대부분의 동기화 명령어들은 각 문장간의 종속 거리가 불변이며 흐름종속만을 가정하고 있다. [9]에서는 복수지정문(multiple statement assignment)에 의한 출력종속의 경우에 대한 해결책으로 확장된 Full /Empty 기법을 제시하였으나, 배열의 첨자식이 I±b(I는 변수, b는 정수상수)인 불변종속거리에 국한되는 단점이 있다.

본 논문에서는 배열 첨자가 a*I±b 형태의 diophantine 방정식의 가변종속거리를 가지는 경우에 대하여 동기화 기법을 적용할 수 있도록 설계하였으며, 반종속을 가지는 문장들에 대하여도 동기화를 효율적으로 수행할 수 있도록 하였다. 또한 이들에게서 존재할 지도 모르는 중복된(redundant) 동기화 제거 기법에 대하여 논의하였다.

본 논문에서 정의한 동기화 명령어 쌍(Release, Wait)은 다음과 같다.

[정의 1] 동기화명령어 Release&Wait 는 다음과 같이 정의한다.

```
Release(v, a*i+b, γ) :
        if(v[a*i+b][γ]. value == i)
                v[a*i+b][γ]. key = TRUE;
Wait(v, p*j+q, γ) :
        while(v[p*j+q][γ]. key == FALSE);

struct {
    int data ;
    int value ;
    boolean key ;
} v[MAX][2] ;
```

여기서 v는 배열 변수, MAX 는 루프반복의 최대값, a, b, p, q는 정수 상수, i, j는 루프반복값, data는 해당 배열의 값, key는 TRUE(=1)과 FALSE(=0) 값만을 가지는 부울(bool)형 정수변수, value는 두 문장간의 출력종속관계에 쓰이는 루프 반복값이다. 종속의 성질을 구별하기 위한 첨자 (γ=1이면 흐름종속, γ=0이면 반종속)의 필요성에 의하여 배열 v는 2차원으로 확장될 수 있으며, 단일 종속 성질만 가질 경우 생략될 수 있다.

Release 명령어는 배열 v[a*i+b]를 가지는 문장을 수행한 후, 해당 배열 v[a*i+b]가 잠금(lock) 상태에서 해제(release)되었음을 알리기 위하여 v[a*i+b]. key=1로 지정(set)한다. 임의의 배열변수 v와 배열첨자 p*j+q를 가지는 Wait 명령어는, Release 명령어가 같은 변수 v와 첨자 a*i+b (a*i+b=p*j+q)를 가지는 문장을 수행할 때까지 계속 기다리면서 문장 수행을 못하게(block) 한다.

[정의 2] 종속 거리 d를 가지는 변수 v를 포함하는 문장 S_1에서 S_2로의 종속 관계가 존재하며, 변수 v에 대한 동기화명령어 쌍을($R_{v,d}$, $W_{v,d}$)라 하자. 이때 S_1을 $R_{v,d}$의 source 문장, S_2를 $W_{v,d}$의 sink 문장이라 하고, S_1=source($R_{v,d}$), S_2=sink($W_{v,d}$)라 정의한다. 또한 S_1 및 S_2의 해당 배열변수를 각각 v[a*i+b]과 v[p*i+q]라 한다.

본 논문에서는 종속거리 d를 명확히 할 필요가 없을 경우에는 변수 v에 대한 동기화명령어 쌍을 (R_v, W_v)로 줄여서 표기하기로 한다. 알고리듬 1에서는 위의 정의들을 이용하여 동기화명령어 쌍 (Release, Wait)을 루프내의 문장들 사이에 적절히 삽입하는 방법을 보여주고 있다.

〈알고리듬 1〉

```
for(루프내의 각각의 종속거리 d>0을 가지는 배열 v)
{
    v[a*i+b]. data = …
    source(Rv) 바로 뒤에 Release 동기화명령어
    삽입;
        …
    sink(Wv) 바로 앞에 Wait 동기화명령어 삽입;
        … = v[p*i+q]. data
}
```

이제, 흐름종속, 반종속 및 Diophantine 첨자식의 가변종속이 혼재하는 그림 1.(a)에 대하여 알고리듬 1을 적용하면 그림 1.(b)와 같다.

```
S1 : a[3 *i-1]=d[i-3]
S2 : b[i]=c[i-5]
S3 : c[i]=a[2*i-1]
S4 : d[i]=b[i+3]
```

(a)

```
    Wait(d, i-3, 1)
S1 : a[3*i]=d[i-3]
    Release(a, 3*i-1, 1)
    Wait(b, i, 0)
    Wait(c, i-5, 1)
S2 : b[i]=c[i-5]
    Wait(a, i-2, 1)
S3 : c[i]=a[2*i-1]
    Release(c, i, 1)
S4 : d[i]=b[i+3]
    Release(d, i, 1)
    Release(b, i+3, 0)
```

(b)

그림 1. 동기화명령어(Release, Wait)의 적용 예
Fig. 1. An Application of (Release, Wait) Synchronization Instructions.

Ⅲ. 중복 동기화명령어 제거 기법

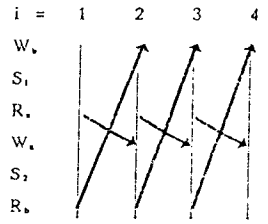
앞절에서는 Release, Wait 동기화 명령어를 각 문장간에 존재하는 종속 종류에 따라 어떻게 유효 적절하게 삽입하는 가에 대하여 논하였다. 이와 같은 동기화 명령어 삽입 방법은 비교적 작은 수의 단순한 문장들로 구성된 루프에서는 별 문제가 없지만, 루프 크기가 커지거나 문장들의 종속 관계가 복잡할 수록 동기화와 관련된 오버헤드는 증가하며, 이는 루프의 수행 효율을 저하시키는 주요한 요인이 된다.

만일 루프내의 임의의 동기화 명령어가 없어도 그 루프의 수행에 아무런 영향이 없다면 그 명령어는 불필요하게 된다. 즉, 중복된 동기화 명령어는 다른 동기화 명령어들의 조합(combination)에 의하여, 그 중복된 명령어의 source 문과 sink 문을 내재적으로(implicitly) 동기화시킬 수 있음을 의미한다. 중복된 동기화 명령어는 제거될 수 있으며, 동기화에 필요한 명령어 집합들의 수가 감소함에 따라 전체 루프의 수행 효율은 증가하게 된다. 여기에서는 어떠한 동기화 명령어들이 중복되어 존재하는 가를 규명한 다음, 기존의 동기화 제거 방법들보다 효율성이 뛰어난 기법에 대하여 논하고자 한다.

기존 방법[9, 10] 에서는 순차 루프내의 문장들 사이에 필요한 동기화 명령어를 삽입시킨 후, 루프의 수행순서 및 동기화를 제어하는 Controlled Path Graph(CPG)라는 일종의 종속그래프를 정의하였다. CPG는 루프내의 문장들 중에서 자료종속거리가 최대인 값 Δ_{max}를 찾아서 $\Delta_{max}+1$ 만큼의 루프 횟수에 대한 각 문장간의 자료종속 관계를 동기화 명령어들을 이용하여 표현한 종속 그래프이다. 그림2(b)에서는 그림2(a)에 대한 CPG를 앞에서 정의한 Release & Wait 동기화명령어를 이용하여 보여주고 있다.

```
    Wait(b, i-1)
S1 : a[i]=b[i-1];
    Release(a, i)
    Wait(a, i-1)
S2 : b[i]=a[i-1];
    Release(b, i-1)
```

(a)



(b)

그림 2. CPG 구성
Fig. 2. Construction of CPG.

그림에서 R과 W는 Release와 Wait를 의미하며, 각 아래첨자(R_a, W_b, …)는 해당 변수를 의미한다. R과 W의 쌍에 대하여 같은 배열변수에 대한 종속거리만큼의 방향성 그래프를 R에서 W로 표시하며, 편의상 i=1, 2, 3에서 시작하는 방향그래프만 표시하였다. 프로그램은 각 루프 i에 대하여 ($1 \le i \le \Delta_{max}+1$) 위에서 아래로 ($W_b$에서 R_b까지) 순차적으로 수행되며, 수행 도중에 만나는 R 명령어에 의하여 방향 그래프를 따라 대응하는 W로 제어를 바꿀 수 있다.

이와 같은 성질을 가지는 CPG 내의 각 원소들에 대한 동기화 명령어 및 문장들로 구성된 인접행렬을 생성한 다음, 그 인접행렬의 이행성 행렬(transitive closure)을 구하여 각 동기화 명령어들의 중복성 여부를 조사한다. 즉, 종속거리 d를 가지는 임의의 배열변수 A에 대한 동기화명령어 쌍 (R_a, W_a)에 대하여 그 명령어들이 없다고 가정한 다음의 나머지 원소들에 대한 이행성 행렬을 구한다. 만일 d의 source 문장과 d의 sink 문장간의 경로가 존재한다면 동기화명령어 쌍 (R_a, W_a)는 제거될 수 있다.

그러나, 이와 같은 방법은 문장간에 존재하는 종속 거리중 최대값(Δ_{max})을 취하여 $\Delta_{max}+1$ 만큼의 루프 횟수에 대한 CPG를 만들고 있으며, Δ_{max}가 비교적 작은 경우에 대하여 만들어진 CPG에 대하여만 유효하다. 또한, 일단 $\Delta_{max}+1$의 루프 크기에 대한 CPG를 구한 다음 루프내에 존재하는 동기화명령어들의 중복성 여부를 모든 종속에 대하여 조사하여야 하며, 이의 계산 비용은 종속의 종류가 많아질 수록 상당히 커지게 된다.

1. 제안된 기법

기존의 방법들은 앞에서 제시한 바와 같이 여러가지의 문제점들을 가지고 있다. 따라서 본 논문에서는 인접행렬의 크기를 최대한도로 줄이고 중복성의 가능성이 있는 동기화명령어들만 조사함으로써 기존의 방법보다 훨씬 시간 복잡도가 작으며, 반종속 및 가변거리의 경우로 확장 가능한 새로운 알고리듬을 제시하고자 한다.

[정의 3] 같은 루프반복내의 문장 S_i가 문장 S_j보다 순차적으로 먼저 수행된다고 할때, S_j의 등급은 S_i보다 크다고 하며, $level(S_j) > level(S_i)$ 라고 표기한다.

[정의 4] 동기화명령어 쌍 (R_v, W_v)를 제거한 후 $Source(R_v)$에서 $Sink(W_v)$에 이르는 경로가 존재할 때 이 경로를 (R_v, W_v)의 대체 경로(alternate path)라 하며, 둘 이상의 동기화명령어 쌍들로 대체 경로가 구성되어 있을 때 이를 조합(combined)대체 경로라 한다.

이제, 각각의 종속 거리에 대한 동기화명령어 쌍 (R_v, W_v)의 R_v에서 W_v로의 연결을 고려해 보자. 정의 3에 의하여 R_v의 등급이 i_r이라면 R_v의 source 문장은 하나 위인 i_r-1 등급에 있으며, W_v의 등급을 i_w라고 할 때 W_v의 sink 문장의 등급 i_w+1 이다. 만일 (R_v, W_v)가 중복된다면, R_v의 source 문장에서 W_v의 sink 문장으로의 제어의 흐름이 (R_v, W_v) 없이도 가능하다는 것을 의미한다. 그림 1의 경우, 두 배열 a, b의 첨자 및 거리값 d(=1)가 같으면 a의 동기화명령어쌍은 중복되므로 제거될 수 있다고 하였다.[7] 본 논문에서는 이러한 특수한 경우 이외에도 배열 변수들의 첨자 및 거리값들이 일반적으로 서로 다른 경우에 대하여 논하고자 한다.

(정리 1)

종속 거리 d를 가지는 임의의 동기화명령어 쌍 (R_v, W_v)가 중복됨을 보이기 위한 조건은 다음과 같다.

i) $source(R_v)$로부터 $sink(W_v)$에 이르는 대체 경로의 유무를 보이기 위해서는, 각 종속 거리 d_i가 d보다 같거나 작은 동기화명령어 쌍들로 구성된 집합이 필요하다.

ii) 그 집합중 적어도 하나의 동기화명령어 쌍 (R_i, W_i)의 등급 $level(R_i)$와 $level(W_i)$는 $level(R_v) \le level(R_i)$ 및 $level(W_v) \ge level(W_i)$를 동시에 만족시켜야 한다.

(증명)

i) 만일 임의의 종속거리 d_i에 대하여, $d_i > d$이면 루프 반복 $1+d_i >$ 루프 반복 $1+d$이 되므로, d의 source 문에서 d_i의 경로를 따라서 d의 sink 문에 결코 도달 할 수 없다. 따라서, $d_i \le d$ 이어야 한다.

ii) 만일 i)의 집합내의 어떤 동기화명령어 쌍도 그러한 조건을 만족하지 않는다면 그 집합내의 모든 동기화명령어 쌍은 $level(R_v) > level(R_i)$ 이거나 $level(W_v) < level(W_i)$의 조건만을 만족한다. $level(R_v) > level(R_i)$의 경우에는 $source(R_v)$ 로부터 어떠한 동기화명령어쌍 (R_i, W_i)의 경로를 이용할 수 없으며, $level(W_v) < level(W_i)$의 경우는 동기화명령어쌍 (R_i, W_i)의 경로 이용후에 결코 $sink(W_v)$ 문에 도달할 수 없다. 따라서, i)의 집합내의 적어도 하나의 동기화명령어 쌍 (R_i, W_i)의 등급 $level(R_i)$와 $level(W_i)$는 $level(R_v) \le level(R_i)$ 및 $level(W_v) \ge level(W_i)$를 동시에 만족시켜야 한다.

위의 정리 1에 의하여 모든 동기화명령어 쌍의 중복성 조사에 대하여 최대 종속 거리 $\Delta_{max}+1$ 만큼의 인접 행렬을 매번 구할 필요가 없으며, 제거하고자 하는 동기화명령어 쌍의 종속 거리 d_i+1 에 대한 인접 행렬만을 구하면 된다. 또한, 기존의 CPG에서처럼 모든 동기화명령어 쌍들에 대한 중복성 여부를 조사하기 위하여 매번 인접행렬 및 이행성 행렬을 구하는 대신에, 정리 1.(ii)를 만족시키는 경우에만 고려하면 된다. 지금까지 논의한 내용을 알고리듬으로 나타내면 다음과 같다.

〈알고리듬 2〉

```
주어진 루프로부터 CPG 및 인접 행렬을 구한다;
for (자료종속 di를 가지는 각 동기화명령어 쌍
      (Ri, Wi)) {
  dj ≤ di, (j≠i)를 만족하는 자료종속 dj를 가지는
동기화명령어 쌍 (Rj, Wj)의 집합 S(R, W)를 구한다;
  for (각각의 (Rj, Wj) ∈ S(R, W))
    if ((level(Rj) ≥ level(Ri) && (level(Wj) ≤
        level(Wi)) {
      find = YES;
      break;
    }
  if (find) {
    S(R, W) ∪ (source(Ri), sink(Wi))에 대한
    인접 행렬 및 이행성 행렬을 구한다;
    if (Source(Ri)로부터 sink(Wi)로의 경로가
        존재)
      (Ri, Wi)을 CPG 로부터 제거한다;
    }
}
```

기존의 CPG에 의한 동기화명령어 제거 알고리듬의 수행시간은 이행성 행렬의 계산*전체 동기화명령어 쌍에 비례하는 시간 복잡도(time complexity)를 가진다. 즉, $\Delta_{max}+1$의 루프 횟수에 대한 CPG의 이행성 행렬을 각각의 동기화명령어 쌍에 대하여 반드시 구하여야 한다. 이러한 CPG의 이행성 행렬의 시간복잡도는 $O(D^*((\Delta_{max}+1)^*S)^3+D^2)$이며, 루프내에 n개의 조사해야 할 동기화명령어가 존재한다면 전체의 시간복잡도는 $O(n^*(D^*((\Delta_{max}+1)^*S)^3+D^2))$가 된다. 따라서 Δ_{max}가 커짐에 따라 이행성 행렬의 계산에 소요되는 시간은 상당히 부담스럽게 된다(여기서 D는 자료종속의 갯수, S는 루프내의 문장수이다.)

본 논문에서 제시하는 방법은 기존의 이행성 행렬을 구성하는 원소의 수를 최대한 감소시키며, 전체 동기화명령어에 대하여 일일이 조사하는 것이 아니라 정리 1에서의 조건을 만족하는 쌍만 조사함으로써 전체 수행시간을 현저하게 감소시킨다. 따라서 전체 알고리듬의 시간 복잡도는 $O((\Delta'^*S)^3)$이 된다.

(여기서 $\Delta'^3 = \sum_{\Delta_i \in L} (\Delta_i+1)^3$, L은 정리1(ii)를 만족하는 종속거리 Δ_i들의 집합이다)

따라서, CPG 알고리듬과 새로운 기법과의 속도비율은 $\rho \simeq n^*(\Delta_{max}+1)^3/\Delta'^3 \gg 1$의 근사치를 갖는다.

표 1은 CPG 및 알고리듬 2를 C 언어로 구현하여 UNIX 운영체제하에서 수행한 결과이며, 시뮬레이션을 위하여 인공적으로 합성한 5개의 루프 문장들에 대한 속도비율을 보여주고 있다.

표 1. CPG 및 알고리듬 2의 속도 비율
Table. 1. The speed ratio between CPG and algorithm 2.

Δ_{max}	ST #	SI #	RSI #	TE #	ρ
3	10	20	5	120	2.23
10	4	8	4	132	2.87
17	10	20	5	480	45.83
22	7	14	2	483	20.69
25	15	30	12	1170	12.57

ST # : 문장 갯수
SI # : 동기화명령어 갯수
RSI # : 중복된 동기화명령어 갯수
TE # : 인접행렬의 원소수

2. 가변 종속 동기화명령어 제거 기법

앞에서는 루프내의 문장들의 종속 거리가 일정한 경우만 고려하여 CPG를 구하였다. 따라서 이행성 행렬의 원소의 범위는 문장간에 존재하는 최대 종속 거리 $\Delta_{max}+1$을 초과하지 않으며, 각 동기화 명령어쌍의 중복성 여부를 판별하기도 비교적 용이하다고 할 수 있다. 그러나, 문장간의 종속거리가 가변일 경우에는 루프의 반복 횟수가 증가할 수록 종속거리가 커지기 때문에, 동기화 명령어의 중복성 여부를 판별하기가 쉽지 않다. 가변종속거리의 경우, 기존의 CPG 방법에서의 Δ_{max} 값은 루프의 최대 반복 횟수 n에 의존적이다. 따라서, 종속성 판별 알고리듬에 대한 수행 시간의 대부분을 차지하는 이행성 행렬 계산에 소요되는 시간은 일반적으로 n이 증가할 경우 상당히 커질 것이며, 이는 매우 비효율적이라 할 수 있다. 또한, 종속 거리는 루프 반복 i의 값에 따라 가변적이며, 따라서 각각 다른 모든 종속 거리에 대하여 동기화 명령어쌍의 중복성 여부를 조사해야 한다. 왜냐하면, 임의의 종속 거리 d_k (k=1, 2…)에 대하여 중복성을 만족하지 못 할 경우, 비록 d_k를 제외한 모든 종속 거리에 대한 중복성을 만족시킨다 하더라도 그러한 조건의 동기화 명령어쌍은 제거할 수 없기 때문이다.

본 논문에서는 배열첨자가 Diophantine 방정식의 형태인 가변종속거리의 경우, 루프 반복 횟수가 증가할때 종속거리가 일정하게 증가하는 사실을 이용하여, 이행성 행렬에서 요구하는 원소의 수를 최대한 축소시킴으로써 효율적인 동기화명령어의 종속성 판별 알고리듬을 제시하고자 한다. 먼저, 루프 반복이 증가함에 따라 가변종속거리가 등차수열의 형태로 일정하게 증가함을 다음의 정리 2에서 보인다.

(정리 2)

임의의 Diophantine 방정식 $a^*i+b = c^*j+d$ (i, j, b, d : 정수, a, c : 자연수)에서, 루프 반복 범위를 만족하는 i, j의 초기값을 (i_0, j_0)라 하자. 그러면 가변종속거리 d의 초기값은 $d_0 = j_0-i_0$이고, 일반적으로 $d_k = d_0+(a-c)/\gcd(a, c)^*k$, $(k \geq 1)$이다. (여기서 gcd는 최대공약수이다.)

(증명)

루프 반복 범위를 만족하는 (i, j) 쌍의 집합을 $\{(i_m, j_m) \mid m = 0, 1, 2, \cdots\}$라 하자. 이때, k번째의 쌍 (i_k, j_k)와 k−1 번째의 쌍 (i_{k-1}, j_{k-1})은

$a^*i_k-c^*j_k+(b-d) = 0$ …… (1)

$a^*i_{k-1}-c^*j_{k-1}+(b-d) = 0$ …… (2)를 만족하며, 식 (1)에서 식 (2)를 빼면, $a^*(i_k-i_{k-1}) = c^*(j_k-j_{k-1})$ …… (3)이 된다.

$d_{ik} = i_k-i_{k-1}$, $d_{jk} = j_k-j_{k-1}$ …… (4)라 할때, 식 (3)을 만족하는 d_{ik} 및 d_{jk}의 최소값은 각각 $d_{ik} = c/\gcd(a, c)$, $d_{jk} = a/\gcd(a, c)$ …… (5)이다.

마지막으로, 식 (4)와 (5)에서

$$
\begin{aligned}
d_k &= j_k - i_k = a/\gcd(a,c) + j_{k-1} - c/\gcd(a,c) \\
&\quad - i_{k-1} \\
&= (a/\gcd(a,c) - c/\gcd(a,c)) + a/\gcd(a,c) \\
&\quad + j_{k-1} - c/\gcd(a,c) - i_{k-1} \\
&= \cdots \\
&= (a/\gcd(a,c) - c/\gcd(a,c)) \quad + \cdots + \\
&\quad (a/\gcd(a,c) - c/\gcd(a,c) + j_0 - i_0 \\
&\qquad \underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}_{k} \\
&= (a/\gcd(a,c) - c/\gcd(a,c)) * k + (j_0 - i_0) \\
&= (a-c)/\gcd(a,c) * k + d_0
\end{aligned}
$$

이제, d_0가 0인 특별한 경우와 그렇지 않은 일반적인 경우로 나누어서 살펴보기로 한다.

1) $d_0 = 0$인 경우

$d_0 = 0$인 경우는 정리 2에서 $d_k = (a-c)/\gcd(a, c)*k$가 된다. 이제,그림3(a)와 3(b)와 같이일정 종속 거리를 가지는 배열 b에 관한 문장들을 포함하는 경우를 고려해 보자.그림3(c)에서는 정리 1을 적용하여, 가변종속을 가지는 변수 a의 첫번째 가변종속거리 d_1 (=3)에 대한 동기화명령어쌍 (R_a, W_a)은 일정 종속 거리 3을 가지는 변수 b에 의하여 제거될 수 있음을 알 수 있다. 또한 정리 2에 의하여 배열 b에 대한 가변 종속 거리는 d_1의 배수만큼 일정하게 증가하므로, 배열 b의 동기화명령어쌍 (R_b, W_b)는 d_1에 대한 동기화명령어쌍 (R_a, W_a)를 대체할 뿐만 아니라 모든 d_k에 대한 동기화명령어쌍들을 대체할 수 있음을 알 수 있다.

그림3(d)는 배열 a의 d_3의 가변 종속거리에 대한 (R_a, W_a)의 경로를 동기화명령어 쌍 (R_b, W_b)가 어떻게 대체할 수 있는가를 보여준다. (R_a에서 W_a에 이르는 실선을 R_b에서 W_b로 이르는 세개의 점선으로 대체함을 알 수 있다.)

이제, 정의 5, 6과 정리 3에 의하여 가변 종속 거리를 가지는 동기화명령어쌍을 제거하기 위한 조건을 구할 수 있다.

[정의 5]

가변 종속 $d = \{ d_k \mid k \geq 1 \}$을 가지는 변수 v 에 대한 동기화명령어 쌍은 다음과 같이 정의한다.

$(R_{v,d}, W_{v,d}) = \{ (R_{v,d}, W_{v,d}) \mid k \geq 1 \}$

[정의 6]

동기화명령어 쌍 $(R_{v,d}, W_{v,d})$의 조합 대체 경로에서 요구되는 동기화명령어 쌍들 중에서 경로의 처음에 위치하는 동기화명령어를 start(v, d, i_s) 이라 하며, 마지막에 위치하는 동기화명령어를 end(v, d, i_e)라 한다. (i_s 및 i_e는 임의의 루프 반복값이며, $i_e = i_s + d$이다.)

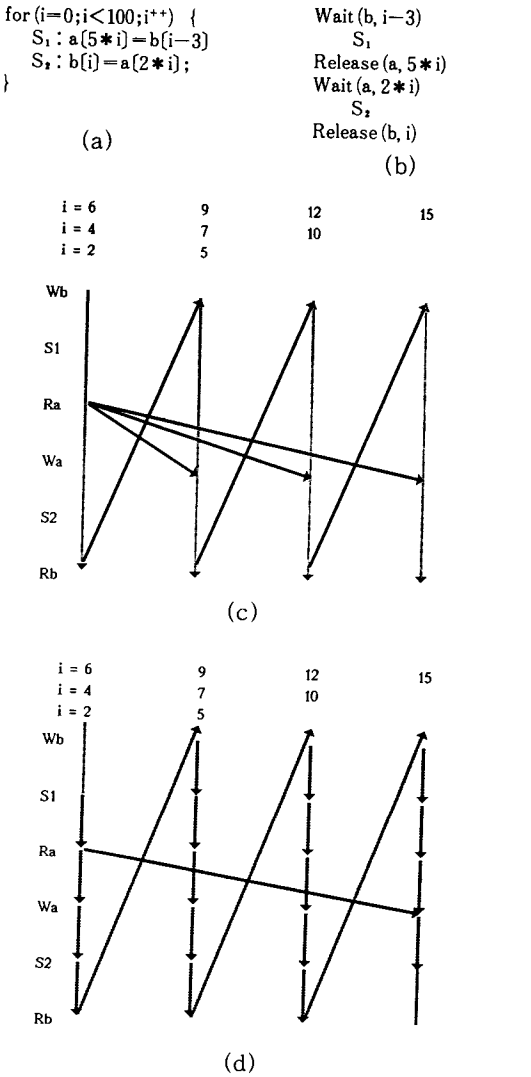


그림 3. d_0=0인 가변종속거리인 경우의 대체경로
Fig. 3. The alternate path for variable dependence distances in case of d_0=0

(정리 3)

정리 2에서의 가변 종속 거리 d_k가 $d_k = (a-c)/\gcd(a, c)*k$, ($k \geq 1$) 일때, 변수 v의 가변 종속 거리 d_k에 대한 모든 동기화명령어 쌍 (R_{v,d_k}, W_{v,d_k})을 제거하기 위한 조건은 다음과 같다.

i) source(R_{v,d_1})에서 sink(W_{v,d_1})까지, 동기화명령어 쌍($R_{v,d}$, $W_{v,d}$)에 대한 대체 경로가 존재한다.

ii) i)의 대체 경로중 적어도 하나는 level(start(v, d_1, i_1)) level(end(v, d_1, j_1)) 를 만족하여야 한다.

((i_1, j_1)은 정리 2에서의 가변 종속 거리 d_1에 대한 루프 반복값이다.)

(증명)

동기화명령어 쌍(R_{v,d_1}, W_{v,d_1})에 대하여 대체 경로가 반드시 존재하여야 하기 때문에 i)은 당연하다. 그리고, CPG 의 성질에 의하여

level(start(v, d_1, i_k)) ≥ level(end(v, d_1, i_k+d_c)),

level(start(v, d_1, i_k+d_c)) ≥ level(end(v, d_1, i_k+2*d_c)),

...

level(start(v, d_1, $i_k+(k-1)*d_c$)) ≥ level(end(v, d_1, i_k+d_k)) 이다. 또한,

level(start(v, d_1, i_k)) = level(start(v, d_1, i_k+d_c))

= ...

= level(start(v, d_1, $i_k+(k-1)*d_c$)),

level(end(v, d_1, i_k+d_c)) = level(end(v, d_1, i_k+2*d_c))

= ...

= level(end(v, d_1, i_k+d_k)) 이므로,

level(start(v, d_1, i_k+j*d_c)) ≥ level(end(v, d_1, i_k+j*d_c)), ($j \geq 1$)가 성립한다.

따라서, start(v, d_1, i_k)에서 end(v, d_1, i_k+d_k)에 이르는 대체 경로가 존재하며, 이는 동기화명령어 쌍 (R_{v,d_k}, W_{v,d_k})에 대한 대체 경로가 존재한다는 것을 의미 한다.

2) $d_0 \neq 0$인 경우

변수 a에 관한 가변 종속 거리 d_0가 0이 아닌 경우의 d_k의 일반식은 정리 2에서 보듯이 $d_k = d_0+(a-c)/\gcd(a, c)*k$, ($k \geq 1$)이다.

그림 4에서는 루프내의 변수 a에 대한 가변 종속 거리 및 이의 대체 경로를 보여주고 있다. 그림 4로 미루어 보아, 주어진 루프의 범위를 만족하는 모든 d_k에 대한 동기화명령어 쌍이 전부 중복되기 위해서는 최소한 다음 2가지 조건들을 만족하여야 함을 알 수 있다.

i) 가변 거리 d_0 -- $i=i_0$ 부터 $i=j_0$ 까지의 루프 반복 -- 에 대하여 대체 경로가 존재하여야 한다.

ii) (d_1-d_0) -- $i=j_0$ 부터 $i=j_1$ 까지의 루프 반복 -- 의 거리에 대한 대체 경로가 존재하여야 한다.

이를 종합하여 $d_0 \neq 0$인 경우의 동기화명령어 쌍($R_{v,d}$, $W_{r,d}$)의 중복성 여부를 판별하기 위한 조건을 구하면 정리 4와 같다.

(정리 4)

정리 2에서의 가변 종속 거리 d_k가 $d_k = d_0+(a-c)/\gcd(a, c)*k$, ($k \geq 1$) 일때, 변수 v의 가변 종속 거리 d_k에 대한 모든 동기화명령어 쌍 ($R_{v,d}$, $W_{v,d}$)을 제거하기 위한 조건은 다음과 같다.

i) source($R_{v,d}$)에서 sink($R_{v,d}$)까지의 동기화명령

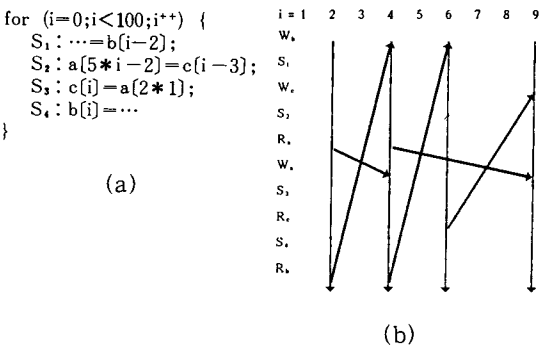


그림 4. 변수 a에 대한 가변 종속 거리 및 이의 대체 경로
Fig. 4. A variable dependence distance and its alternate path for variable a.

어 쌍 ($R_{v,d}$, $W_{v,d}$)에 대한 대체 경로가 존재한다.

ii) sink(end(v, d_0, i_1+d_0)) 부터 sink(W_{v,d_1}) 까지의 대체 경로가 존재한다.

iii) sink(end(v, d_0, i_1+d_0))의 start 동기화명령어를 R' 이라고 할때, ii)의 대체 경로중 적어도 하나는 level(R') ≥ level(end(v, d_1,j_1))를 만족한다. ((i_1, j_1)은 정리 2에서의 가변 종속 거리 d_1에 대한 루프 반복값이다.)

(증명)

그림 5와 같이 모든 동기화명령어 쌍 (R_{v,d_0}, W_{v,d_0})의 경로에 대한 start 동기화명령어의 루프 반복을 i_k가 되도록 평행 이동한 상태를 고려하기로 한다.

i) 동기화명령어 쌍 ($R_{v,d}$, $W_{v,d}$)에 대하여 대체 경로가 반드시 존재하여야 하기 때문에 당연하다.

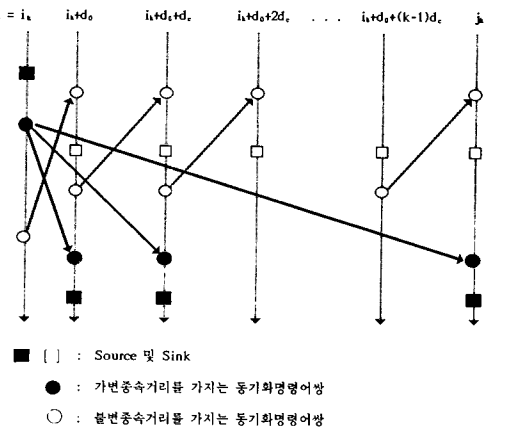


그림 5. 루프반복 i_k에 대한 각 동기화명령어쌍
Fig. 5. Each synchronization instructions for the loop iteration i_k

ii) level(sink(end(v, d_0, i_k+d_0))) level(end(v, d_0, i_k+d_0)) 이므로 sink(end(v, d_0, i_1+d_0)) 부터 sink(W_{v_1, d_1}) 까지의 대체 경로가 존재한다면 i)에 의하여 start(v, d_1, i_k)부터 end(v, d_1, $i_k+d_0+d_c$)까지의 경로가 존재한다. 즉, 이 사실은 동기화명령어 쌍(R_{v, d_1}, W_{v, d_1})에 대한 대체 경로가 존재함을 의미한다.

iii) 정리 3과 같은 증명 방식을 따르면 sink($W_{v, d}$)에서 sink(W_{v, d_k})에 이르는 대체 경로가 존재함을 알 수 있으며, 이 사실과 i), ii)에 의하여 동기화명령어 쌍(R_{v, d_k}, W_{v, d_k})에 대한 대체 경로가 존재함을 알 수 있다.

Ⅵ. 결 론

본 논문에서는 밀착결합(strongly-coupled) 구조의 MIMD 시스템에서의 문장 수준에서의 동기화를 위한 Release&Wait 동기화명령어를 제안하였다. 문장의 종속이 흐름종속뿐만 아니라 반종속이 혼재된 경우에도 적용할 수 있도록 하였으며, 과도한 동기화 명령어의 사용으로 발생할 수 있는 오버헤드(overhead)를 최소화하기 위하여 효율적인 중복 동기화명령어 제거 방법을 제시하였다.

앞으로는, 완전한 동기최적화를 위해서 문장내의 제어종속을 가지는 경우에 대한 동기화 및 일반적인 중첩루프로의 확장이 있어야 할 것이다.

參 考 文 獻

[1] Callahan, D., K. Kennedy and J. Subhlok, "Analysis of event synchronization in a parallel programming tool," ACM SIGPLAN, 1990.

[2] Chu, C. and D. Carver, "An analysis of recurrence relations in fortran doloops for vector processing," Proceedings of the Fifth International Parallel Processing Symposium, May 1991.

[3] Cytron, R. G., "Compile-time scheduling and optimazation for asynchronous machines," Ph. D. dissertation, Univ. of Illinois at Urbana-Champaign, Oct. 1984.

[4] Dinning, A., "A aurvey of synchronization methods for parallel computers," *IEEE Computer*, July 1989.

[5] Emrath, P. and D. Padua, "Automatic detection of nondeterminacy in parallel programs," ACM SIGPLAN and SIGOPS Workshop on Parallel and Distributed Debugging, May 1988.

[6] Gottlieb, A. el all., "The NYU ultracomputer designing a MIND, shared memory parallel computer," *IEEE Transactions on Computers*, vol. C-32, no. 2, pp. 276-290, February 1983.

[7] Li, Z. and W. Abu-Sufah, "On reducing data synchronization in multiprocessed loops," *IEEE Trans. Comput.*, vol C-36, pp. 105-109. Jan. 1987.

[8] Malony, A., J. Larson, and D. Reed, "Tracing application program execution on the cray X-MP and cray 2," Tech. Rep., Univ. of Illinois at Urbana-Champaign, CSRD Rpt. no. 985, November 1990.

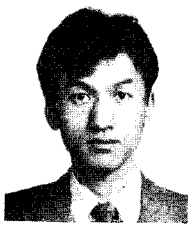
[9] Midkiff, S. P. and D. A. Padua, "Compiler algorithms for synchronization," *IEEE Trans. on Computers*, vol 36, no. 12, Dec. 1987.

[10] Midkiff, S. P. and D. A. Padua, "A comparison of four synchronization optimization techniques," *Proc. of the ICPP*. vol II, pp. 9-16, 1991.

[11] Polychronopoulos, C. D., "Compiler optimazation for enhancing parallelism and their impact on architecture design," *IEEE Transactions on Computers*, vol. 37, no. 8, August 1988.

[12] Wolfe, M., "Multiprocessor synchronization for concurrent loops," *IEEE Software*, pp. 34-42, January 1988.

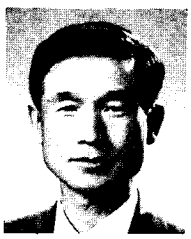
著 者 紹 介



金 秉 洙(正會員)

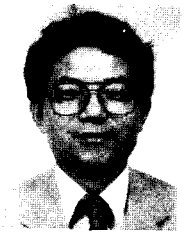
1981年 서울대학교 수학과 졸업(학사). 1987年 미국 오클라호마 주립대 전산학과 졸업(석사) 1991年 고려대학교 대학원(전산학 전공) 박사과정 수료. 1987年～1991年 한국과학기술원 시스템공학연구소 근무. 1991年～현재 건양대학교 전자계산학과 전임강사. 주관심분야는 병렬처리, 컴파일러, 알고리듬 등임.

●



黃 鍾 善 (正會員)

1978年 Univ. of Georgia, Dep. of Stat. & Computer Science 박사 1978年～1980年 미국 South Carolina 주립대학 조교수. 1980年～1981年 미국 상무성 연방 표준국 연구위원. 1981～1982年 한국표준연구소 전자계산실장. 1986年～1989年 한국정보과학회 부회장. 1982年～현재 고려대학교 전산과학과 교수. 주관심분야는 병렬처리 알고리듬, 인공지능론, 소프트웨어 분석론 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). 1985年～현재 순천향대학교 전산학과 부교수. 1992年～현재 미국 Univ. of Illinois at Urbana Champaign 에서 Post Doc. 주관심분야는 병렬처리 컴파일러 연구, 프로그래밍 언어론, 계산이론등임.