1차원 배열의 다중첨자를 갖는 루프의 병렬화를 위한 동기화 기법

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

A Synchronization Method for Parallelizing Nested Do Loop with one dimensional variable

Hyun-Ho Park*, Sung-Dae Youn**
*Dept of Computer and Information, Pu-kyong National University
**Dept of Computer Science, Pu-Kyong National University

요 약

일반적인 응용 프로그램에서 루프는 대부분의 수행시간을 차지하기 때문에 병렬성 추출의 핵심 부분이라 할 수 있다. 병렬성이 많은 구조는 루프 구조이며, 루프를 병렬로 처리하기 위해 각 반복간에 존재하는 데이터의 종속은 프로세서간의 동기화가 필요하다. 본 논문에서는 다중첨자를 갖는 1차원 배열의 루프의 병렬화를 위해 다수 개의 동일한 종속값을 이용하여 종속함수를 생성하고 이를 이용하여 종속관계가 성립하지 않는 비종속 구간(Non-dependence part)을 구한다. 그리고 동일한 값을 가지는 복수개의 종속값 간의 동기화는 외부루프 분할 기법을 이용하여 적은 횟수의 첨자가 외부에 위치 하도록 하여 간소화 한 후 단일 첨자를 갖는 루프에 동기화를 수행하는 기법을 제시한다.

1. 서론

병렬처리(parallel processing)란 다수의 프로세서들이 여러 개의 프로그램들 또는 한 프로그램의 분할된 부분들을 분담하여 동시에 처리하는 기술을 말한다. 최근 컴퓨터 사용 기술의 급속한 발전으로 인해 이를 이용한 서비스의 질적, 양적인 향상에 대한 사용자의 요구가 날로 증대됨에 따라 순차 프로그램을 보다 효율적으로 처리하기 위한 병렬 처리에 대한 연구가 많이 이루어져 왔다. 루프를 병렬로 처리하기 위한 연구 또한 활발히 이루어져 왔다. 또한 응용 프로그램에서 루프는 대부분의 수행시간을 차지하기 때문에 병렬성 추출의 핵심 부분이라 할 수 있다.[2,5]

루프의 형태는 doall과 doacross가 있다. 모든 반복들간에 종속성이 없는 경우를 doall이라 하고 서로 다른 반복간에 종속성이 발생하는 경우를 doacross라 한다.[3,4] 루프를 펼쳤을 때 모든 각각의 문장을 인스턴스라 하고 종속의 근원이되는 인스턴스를 source 인스턴스, 종속이 되는 인스턴스를 sink 인스턴스라 한다. doall 형태는 종속관계가 존재하기 않기 때문에 한 반복내의 데이터들을 프로세서간의 상호 작용없이 병렬처리할 수 있으나 doacross루프에서는 한 반복에서 가져온 데이터가 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.

본 논문에서는 불변 및 가변 종속거리가 동시에 존재하는 경우에도 적용 가능한 새로운 동기화 기법을 제시한다. 본 논문의 구성은 2장에서 기존의 연구를 소개하고, 3장에서는 제안한 동기화 기법과 동기화 알

고리즘을 제안하고, 4장에서는 모의 실험을 통한 제안 알고리즘의 타당성을 평가한 후, 5장에서 결론 및 향후 과제를 제시한다.

2. 관련 연구

2.1 Diophantine 방정식

1차 Diophantine 방정식 $ax+by=c$ ($a\neq0$ 또는 $b\neq0$)의 정수 해가 존재하기 위한 필요충분 조건은 $GCD(a,b) \mid c$ 이다. $d=GCD(a,b)$ 이고, $d \mid c$ 일 때 (x_0, y_0)를 Diophantine 방정식의 초기해라고 하면, 식(1)에 의해서 모든 정수해 (x,y)를 구할 수 있다.

$$x=x_0+\frac{b}{d}t,\ y=y_0-\frac{a}{d}t$$

($a,b,c,d,t \in Z$, where Z : 정수) (1)

2.3 기존의 연구

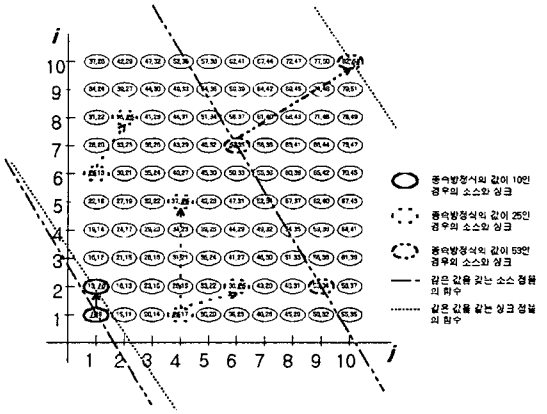
기존의 동기화 기법에는 종속거리가 일정한 경우, 즉 불변종속거리에 적용되어지는 동기화 기법과 가변 종속거리에서의 동기화 기법으로 나눌 수 있다. 또한 가변 병렬성을 추출하는 크기에 따라 데이터 지향 기법(data oriented scheme : Full/Empty tag 기법[1])과 문장 지향 기법(statement oriented scheme : Lock/unlock, Advance/Wait[1,3], Set/Wait, Testset/Test 기법[3])등으로 나눌 수 있다. 불변 종속거리 동기화 기법은 가변 종속거리를 갖는 루프에 적용할 때 많은 오버헤드가 존재하고, 기존의 가변 종속거리 동기화 기법은 단일첨자를 갖는 루프에 적합하다.

3. 제안한 동기화 기법과 알고리즘

```
Do  i   = 1, 10
   Do  j  = 1, 10
      A(3i + 5j + 2)  =  .....
                ....    =  A(2i + 3j + 3)
   End do
End do
```

(그림 1) 다중첨자를 갖는 1차원 배열 중첩루프

제안할 동기화 기법이 적용될 루프의 예를 살펴 보면 그림 1과 같고, 문장 $S1, S2$에서 변수 A에 대해 흐름종속이 발생하는 것을 알 수 있다. 종속성에 위배되지 않으면서 동기화를 하기 위해서는 변수 A의 첨자식을 이용하여야 한다. 그림 2에서 타원 안의 첫 값은 source문장 첨자식에 각각의 i,j값을 넣은 종속값이고, 두 번째 값은 sink문장의 값이다. 그림 2에서 알 수 있듯이 동일한 종속값을 발생하는 첨자값이 여러개 존재하는 것을 알 수가 있고, 이런 특성 때문에 기존의 동기화 기법으로는 동기화가 어렵다. 이 경우의 동기화와 병렬성을 높이기 위해서 그림 1의 루프에서 source, sink의 첨자식을 이용하여 non-dependence[4]구간을 추출하고, 나머지 공간에 대해서만 동기화를 한다. 그림 2는 루프공간에서 종속값이 10, 25, 53일 때 종속값과 종속관계를 보이고 있다. 예제 루프의 경우 종속값이 i축으로는 첨자식의 i계수 만큼, j축에 대해서 j계수 만큼 증가하는 것을 알 수가 있고, 이를 이용하여 동일한 종속값에 대한 함수를 구할 수가 있다. 또한 루프첨자 i,j의 lower bound value와 upper bound value를 알 수가 있고, 첨자식을 이용하여 source와 sink의 최소 종속값과 최대 종속값을 구할 수가 있다. 최대 종속값을 지나는 함수 이후의 문장들은 종속관계를 갖지 않는다. 따라서 이러한 문장들은 임의적으로 병렬로 처리할 수 있기 때문에 병렬성을 높이기 위해서 추출해야 한다.



(그림 2) 예제 루프에서 종속관계의 예

종속값이 여러개 존재할 때, 동기화를 위해 루프 분할 기법을 이용한다. 즉 외부루프는 순차적으로 실행하고 내부를 병렬로 처리한다. 루프 분할을 최소화 하기 위해서 루프첨자의 횟수를 비교하여 적은 횟수의 첨자가 외부에 위치하도록 루프교환을 한 후 루프 분할을 수행한다. 예를 들어 외부루프의 횟수가 n번이고, 내부루프의 횟수가 n^2이라면, 루프의 분할을 하기전의 시간복잡도는 $O(n^3)$이고, 외부루프를 분할하면 n^2횟수를 갖는 단일 루프가 n개가 생긴다. 이때의 시간복잡도는 또한 $O(n^3)$이지만, n^2횟수를 갖는

단일 루프의 경우 첨자식이 다중 첨자식이 아닌 내부 루프의 첨자식만을 갖기 때문에 쉽게 동기화가 가능하고, 병렬로 처리가 가능하다. 그러므로 실제 시간복잡도는 $O(n^3)$보다 작게 된다. Non-depend ence 부분을 구하는 알고리즘에서 고려되어야 할 부분은 종속관계를 갖는 첫 번째 source 인스턴스와 sink 인스턴스 이전의 인스턴스들은 종속성을 갖지 않으며, 마지막 source, sink 인스턴스 이후의 인스턴스들은 루프의 상한 값에 의해 종속성을 갖지 않는다.[3] 다중 첨자를 갖는 루프의 경우 동일한 종속값을 발생하는 i, j의 값이 여러개 존재함으로 이들 같은 값을 지나는 함수를 생성하고, source와 sink의 마지막 종속값을 비교하여 후면 비종속 부분을 구하는 알고리즘은 알고리즘4.1과 같다. 전면 비종속 부분과 이 알고리즘에 의해서 제거되지 않은 후면 비종속 부분은 분할된 루프의 동기화에서 고려한다.

[알고리즘 4.1] 후면 비종속 구간을 구하는 알고리즘

```
i - 루프에서 첨자 i의 Lower Bound Value
j - 루프에서 첨자 j의 Lower Bound Value
I - 루프에서 첨자 i의 Upper Bound Value
J - 루프에서 첨자 j의 Upper Bound Value
S - Source,  K - Sink
dep_value - i, j 값에 따른 종속값

if (dep_value_IJS < dep_value_IJK) {
   call dioph1(dep_value_IJS)
// 마지막 종속값에 대한 sink의 i, j를 diophantine 방정식을 이용하여 구함
     β = (ci + dj) / c
   if ( d/c < 0 )            // 기울기 비교
      J = (-ci + cβ) / d
   else
      j = (-di + cβ) / c
else
   call dioph2(dep_value_IJK)
// 마지막 종속성에 대한 source의 i, j를 diophantine 방정식을 이용하여 구함
     α = (ai + bj) / a
   if ( b/a < 0 )            // 기울기 비교
      J = (-ai + aα) / b
   else
      j = (-bi + aα) / a    }
```

그림 3은 그림 1의 예제 루프에서 후면 비종속 부분을 표시한 것이다. 종속성이 존재하는 영역은 동일한 종속값이 여러개 존재하므로 이를 처리하기 위해 내부루프는 동기화 기법을 이용하여 병렬로 수행하고 루프 분할을 통해 외부루프는 순차로 처리한다. 루프 분할에 앞서 순차 처리 부분을 줄이기 위해 내부 루프의 상한 값과 외부 루프의 상한 값을 비교하여 작은 값을 가지는 루프 상한값을 외부루프로 변환하는 루프 교환을 수행한다. 내부 루프의 동기화에서 전면 비종속과 후면 비종속이 발생하는데 이를 제거 함으로써 보다 병렬성을 높일 수 있다. 루프교환 및 분할 알고리즘 및 분할된 내부 루프의 동기화 기법은 알고리즘4.2, 알고리즘 4.3, 알고리즘 4.4과 같다.

[알고리즘 4.2] 루프교환 및 분할 알고리즘

```
i - 루프에서 첨자 i의 Lower Bound Value
j - 루프에서 첨자 j의 Lower Bound Value
I - 루프에서 첨자 i의 Upper Bound Value
J - 루프에서 첨자 j의 Upper Bound Value
Rv - 후면 비종속 함수

Init_j = j - 1
Last_j = Rv의호출값  if Rv가존재할때
         J - 1       otherwise
T' = ai + bj + c     // 루프 분할 후 소스 첨자식의 초기 상수 값
T" = xi + yj + z     // 루프 분할 후 싱크 첨자식의 초기 상수 값

if J > I {
  Do m = i, I
  {
    Do k = Init_j, Last_j
    { A(bk + T') = ···
                 ··· = A(yk + T") }
    T' += a
    T" += x
  }
 }
else
  Do m = j, J
  {
    Do k = Init_i, Last_i
    { A(bk + T') = ···
                 ··· = A(yk + T") }
    T' += a
    T" += x
  }
 }
```

[알고리즘 4.3] 내부루프의 동기화 알고리즘

```
// Diopantine 방정식(1)을 이용하여 종속관계를 갖는 첫 j를 구함.
x0 : source에서 첫 종속관계의 j 값
y0 : sink에서 첫 종속관계의 j 값
a : source 첨자의 계수,  b : sink 첨자의 계수
J : 첨자 j의 upper bound value,  N : 문장의 개수
d = GCD(a, b)
```

for (k = x_0, l = y_0 ; l <= J×N ; k+= $\frac{b}{d} \times N$, l+= $\frac{a}{d} \times N$)

{ Insert Unlock instruction immediately after k
Insert lock instruction immediately before l }

[알고리즘 4.4] 동기화 연산

X(m) = synchronization variables // m는 동기화 변수의 번호
Unlock(k, X(m))

for (K = x_0, L = y_0; L <= J×N ; k+= $\frac{b}{d} \times N$, l+= $\frac{a}{d} \times N$)

if (k == K) // source 인스턴스의 수행 여부 점검
X(m) = L // source 인스턴스의 수행 완료 표시

Lock (K, X(j))
wait until (l == X(j)) // source 인스턴스의 완료 여부

그림 4는 그림 1의 예제 루프를 실제 제안한 알고리즘에 적용했을 때의 분할된 루프들을 보여준다. 분할된 루프내에서는 동일한 종속값이 없고 단일 첨자를 가지기 때문에 쉽게 동기화를 할 수가 있다. 또한 알고리즘에서 제거하지 못한 전면 비종속과 후면 비종속을 찾아 낼 수 있으므로 보다 효율적인 병렬 처리가 가능하다.

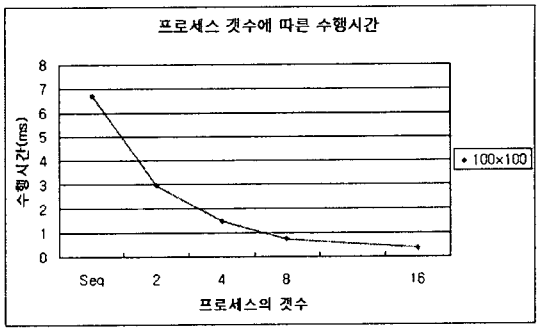
```
Do  j  = 0, 8
    A(5j + 10)  =  .....
          ....  =  A(3j + 8)
End do

Do  j  = 0, 8
    A(5j + 13)  =  .....
          ....  =  A(3j + 10)
End do
        ⋮
Do  j  = 0, 4
    A(5j + 37)  =  .....
          ....  =  A(3j + 26)
End do
```

(그림 4) 외부루프 분할과 첨자식이 변경된 루프

4. 모의 실험을 통한 제안 알고리즘 타당성 평가

3장에서 제안한 기법을 100×100 루프에 적용하여 프로세스의 개수를 2, 4, 8, 16로 변화시 수행 시간을 측정하였다. 종속함수 $i = \frac{-5j+501}{3}$ 이후의 루프 영역은 비종속 영역으로 따로 추출하여 병렬로 처리한다. 나머지 부분은 루프 교환과 분할, 내부동기화를 적용하여 수행 하였다.



(그림 5) 모의 실험 결과

그림 5은 프로세스 개수에 따른 모의 실험 결과이다. 순차처리가 6.8ms 정도의 시간이 걸린데 비해 프로세스 개수가 2, 4, 8, 16일때는 각각 2.9, 1.5, 0.7, 0.4ms의 시간이 필요하였다.

5. 결론

본 논문에서는 다중 첨자를 가지는 루프의 동기화 기법을 제안하였다. 다중 첨자를 가지는 루프는 단일 첨자를 가지는 루프에 비해 동일한 종속성을 유발하는 i, j값이 다양하게 존재하므로 기존의 동기화 기법으로는 동기화가 어렵고, source와 sink 문장간의 동기화 이외에도 sink와 source 문장간의 동기화가 필요하다. 본 연구에서 제안한 기법은 같은 종속값을 가지는 i, j값을 이용하여 종속함수를 만들고, 효율성 증진을 위해, 루프내의 비종속 영역을 우선 제거한다. 복수개의 동일한 첨자값에 대한 동기화는 외부루프 분할을 이용하여 해결하고, 간소화된 루프의 동기화를 수행하는 다중첨자를 가지는 루프에서의 동기화 기법을 제안하였다.

향후 연구로써 제안한 기법의 복수 배열을 가지는 다차원 첨자 루프에의 적용, 첨자식을 이용한 최적화된 동기화 기법을 연구하는 것이다.

참고문헌

[1] 이광형, 황종선, 박두순, "중첩 루프의 병렬화를 위한 새로운 동기화 기법" Journal of KISS. Vol. 22, No. 11, January 1995

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

[3] 박현호, 김영만, 배은호, "중첩 루프의 병렬화를 위한 동기화 기법" proc of 15th KIPS Spring Conference. 2001