공유메모리 프로그램의 디버깅을 위한 프로그램 재구성

김 영 철*

1. 서 론

병렬 프로그램(Parallel program)은 순차적 프로그램에 비해 내재된 특성 때문에 오류를 수정하기가 어렵다. 특히, 공유메모리(shared memory)를 사용하는 병렬 프로그램은 수행 중에 자료경합(data race) 또는 경합(race)이라는 오류를 발생시킨다. 경합은 병행하게 수행되는 스레드(thread)들 간에 실행하는 명령 중에 동일 공유변수에 적어도 하나의 쓰기 접근을 가지고, 적절한 동기화 없이 접근 할 때 발생한다. 프로그램의 비결정적(non-deterministic) 수행을 야기하는 경합을 탐지하는 것은 병렬 프로그램의 디버깅에서 중요하다.

병렬 프로그램에서 경합을 탐지하기 위한 기법으로는, 탐지 정보를 수집하고 분석하는 시기에 따라서 수행전, 수행후, 수행중 분석 기법인 정적 분석(static analysis) 기법[4,8]은 모든 탐지 작업을 수행 전에 하기 때문에, 비실제적인 경합을 너무 많이 포함한다는 단점이 있다. 수행후 분석 기법인 사후 분석(post-mortem analysis) 기법[9, 17]은 프로그램 수행 중에 추적 내용(trace log)을 저장해 두었다가 수행 후에 분석하는 기법으로, 실제 나타날 수 있는 경합만 탐지하지만, 추적 내용 저장에 따른 과중한 기억 장소 부담이 생기는 단점이 있다. 끝으로 모든 탐지 작업을 수행 중에 하는 수행중 분석(on-the-fly analysis) 기법[6,9, 11,13,22]은 수행후 분석 기법만큼 많은 경합을 탐지하지는 못하지만 경합이 존재한다면 각 공유변수에 대해 적어도 하나는 탐지를 보장하는 장점이 있으며, 순서적 동기화(ordered synchronization)가 있는 병렬프로그램에서는 최초 경합을 탐지[22]할 수 있다.

수행중 경합 탐지 기법은 공유변수에 접근하는 스레드들 간의 논리적 병행 여부를 결정하기 위해서, 원시 프로그램에는 경합 탐지에 필요한 부가적인 코드(instrumental code)를 추가한다. 그리고 실행 중에는 공유변수에 대한 스레드의 접근 역사(access history)를 유지하게 된다. 이때 필요한 정보를 생성하고 저장하기 위해 필요로 하는 공간은 최대 병렬성에 의존적으로 증가한다. 더구나 동기화가 있는 병렬 프로그램인 경우는 동기화 정보까지 유지해야 하므로 기억 공간 문제는 더욱 심각해진다.

그래서 경합을 탐지할 때 기억공간을 줄일 수 있는 방안으로 순차적 모니터링을 제안되어져 왔다. 그러나, 순차적 모니터링은 적용대상이 제한적이라 임의적 동기화가 있는 병렬 프로그램에 대해서는 해결 방안이 되지 못했지만, [13]에서 임의적 동기화가 있는 병렬 프로그램의 경합 탐지에서도 기억 공간의 부담을 줄일 수 있는 방안을 제시하고 있다.

* 진주국제대학교 컴퓨터미디어학과 교수

본 고에서는 임의적 동기화가 있는 병렬 프로그램에서 효율적인 경합 탐지 방안으로 제시되었던 프로그램을 재구성하는 기법을 단일사건을 가지고 기술한다. 먼저 병렬 프로그램의 유형을 병렬화 구조와 동기화 구조를 살펴본 후, 수행중 경합 탐지 기법의 효율과 효과에 대해 기술한다. 그리고, 실용적 경합 탐지를 위한 기존의 연구들과 공간 효율적인 탐지 기법을 기술하고, 마지막으로 병렬 프로그램을 재구성하는 기법을 단일사건을 가지고 설명한다.

2. 병렬 프로그램의 유형

지금까지 가장 일반적인 병렬 컴퓨터 구조는 공유메모리(shared memory)를 가지는 다중 프로세서(multiprocessor)구조와 메시지전달(message passing)에 의한 다중 프로세스 구조이다. 여기에서는 전자의 경우에 대해서만 설명한다.

공유메모리 구조를 사용하는 병렬 프로그램의 유형을 소개하기 위해 여기서는 PCF(Parallel Computing Forum)의 Parallel Fortran[20]과 OpenMP[19]에서 나타나는 병렬루프, 병렬섹션 그리고 내포 병렬성 등의 병렬화 구조와 순서적 동기화 그리고 임의적 동기화를 가진 프로그램 등의 동기화 구조에 대해 PCF Fortran을 대상으로 설명한다.

2.1 병렬화 구조

대부분 복잡한 수치 연산이 요구되는 공학이나 과학 프로그램들은 루프 처리에 많은 시간을 소비한다. 그래서 여러 프로세서들에서 수행되도록 하는 병렬구조가 필요로 하는 경우가 많다. 이런 병렬 구조는 PCF Fortran에서는 병렬루프, 병렬 섹션 그리고 두 가지 구조가 혼합된 내포 병렬 구조 등으로 나타난다.

먼저 병렬루프 구조는 생성 및 합류를 PARALLEL DO와 END PARALLEL DO로 표현된다. PARALLEL DO 문장에 의해 다중 스레드들이 생성되면, 대응하는 END PARALLEL DO 문에 의해 스레드들이 합류된다. 그림 1은 이러한 병렬루프를 포함하는 병렬프로그램[1,14,20]의 예를 보이고 있다.

병렬섹션 구조는 생성 및 합류를 PARALLEL SECTIONS과 END PARALLEL SECTIONS으로 표현된다. PARALLEL SECTIONS 문장에 의해 다중 스레드들이 생성되면, 대응하는 END PARALLEL SECTIONS 문에 의해 스레드들이 합류된다. 병렬섹션 안의 SECTION 문에 의해 각 스레드들의 수행이 이루어진다. 그림 2는 이러한

```
PARALLEL DO I = 1, 3
   . . .
   IF . . . = X
   . . .
   IF . . . X = . . .
   . . .
END PARALLEL DO
```

그림 1. 병렬 루프의 예

```
PARALLEL SECTIONS
  SECTION
     . . .
     IF . . . = X
     . . .

  SECTION
     . . .
     IF . . . X = . . .
     . . .
END PARALLEL SECTIONS
```

그림 2. 병렬 섹션의 예

병렬섹션을 포함하는 병렬 프로그램의 예를 보이고 있다.

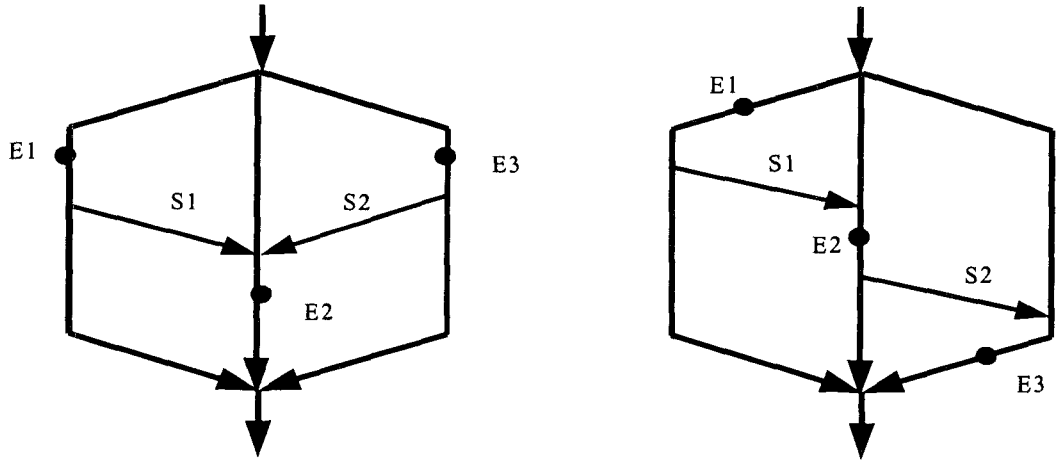
마지막으로 내포 병렬성 구조로 병렬루프 안에 또 다른 병렬 루프가 존재하는 구조, 또는 병렬섹션 안에 병렬섹션이 존재하는 내포 병렬 구조가 있다.

프로그램 수행에서 각 스레드들간의 병행 관계는 그림 3에서와 같이 POEG(partial order execution graph)이라는 비순환 그래프를 이용하여 각 스레드 정보로 판단할 수 있다. POEG에서 정점(vertex)은 스레드의 생성이나 종료명령이며, 임의 정점에서 시작하는 간선(arc)은 그 정점으로부터 수행되는 스레드 블록 또는 대응되는 두 동기화 명령들간의 수행 순서를 의미하는 동기화 간선을 표시한다. 일반적으로 병렬구조를 가진 프로그램이 수행될 때, 처음에는 마치 순차 프로그램과 같이 하나의 스레드가 수행을 시작한다. 이런 수행은 병렬 구조가 나타날 때까지 순차적으로 수행되며, 병렬 구조가 나타나면 프로그램은 그 병렬 구조 내의 코드 블록을 수행하기 위해 추가적인 스레드를 생성시켜 수행된다.

2.2 동기화 구조

병렬 구조 내에서는 여러 형태의 동기화 명령들이 존재할 수 있다. 일반적으로 사용 가능한 동기화 명령으로는 동기식(synchronous)과 비동기식(asynchronous)이 있다. 임의의 두 스레드가 장벽(barrier) 등과 같은 동기식 명령에 의해 동기화되면, 동기식 명령을 먼저 수행 한 스레드는 나중에 다른 한 스레드가 대응되는 동기식 명령을 수행할 때까지 기다리게 된다. 만일 신호/대기(signal/wait) 혹은 잠금/해제(lock/unlock) 형태의 비동기식 명령에 의해 동기화 되면, 대기 명령을 수행하는 스레드는 다른 한 스레드가 신호 명령을 수행 할 때까지 기다려야 되지만, 신호 명령을 수행하는 스레드는 대기 스레드의 상태와 무관하게 계속 진행하게 된다. 따라서, 동기식 동기화는 2개의 서로 다른 비동기식 동기화들의 조합으로 볼 수 있다. 이러한 스레드 명령과 동기화 명령을 통칭하여 병렬수행(parallel operation)이라 한다.

동기화에는 순서적(ordered) 동기화와 임의적(random) 동기화가 존재한다. 순서적 동기를 가



(a) 임의적 동기화 (b) 순서적 동기화

그림 3. 동기화 모델

진 병렬 프로그램은 프로그램의 매 수행 시마다 동기화 명령들이 항상 동일한 순서로 발생되는 것으로 병렬 프로그램의 결정성을 보장하는 형태를 가진다. 이와는 달리 임의적 동기를 가진 프로그램은 병렬프로그램의 수행시 동기화 명령들의 순서가 변경될 수 있는 형태이며, 이와 같은 경우는 프로그램에서 발생하는 접근들간의 순서화 및 병행성을 판단하기 어렵고, 동기화 명령들이 항상 동일한 순서로 발생된다는 것을 보장하지 못하기 때문에 비결정성을 유발한다.

그림 3(a)에서 보면 이는 임의적 동기화를 가지는 경우이다. 여기서 보면, 동기화 명령 S1이 먼저 발생되면 E1과 E3은 순서화 되지만, E2와 E3은 서로 병행하다. 하지만 S2가 먼저 발생하면 E2와 E3은 순서화 되지만, E1과 E3은 서로 병행해지기 때문에, 이들 접근들 간의 순서화 및 병행성을 판단하는데 어려움이 존재한다. 따라서 이러한 임의적 동기화는 그 사용이 현실적으로 극히 제한 적이다. 이와는 달리 순서적 동기는 그림 3(b)와 같이 두 동기화 명령 S1, S2 는 항상 동일한 순서로 발생되기 때문에 프로그램이 결정성을 보장한다.

3. 수행중 경합 탐지

대규모 병렬 프로그램에서의 경합을 탐지하는 과정은 대량의 병행성 정보와 탐지 정보를 위한 기억장소를 필요로 하며, 경합을 결정하기 위한 프로토콜은 수행하는 시간을 필요로 한다. 이러한 대량의 기억공간 비용과 경합 탐지 시간은 탐지 기법의 적용에 영향을 준다.

수행중 경합 탐지는 Nudler와 Rudolph[18]에 의해 처음으로 제안되었으며, 수행 중에 공유 변수와 접근 사건들의 종류, 그리고 각 스레드들의 병행성에 관한 정보를 얻기 위해 프로그램의 원시 코드에 부가적인 코드를 삽입한 후에 모니터링 한다. 이들 중에 병행성 정보는 접근 사건들이 발생한 스레드들의 논리적 병행 관계를 결정하기 위해 필요로 한다. 이러한 논리적 병행성은 스레드들의 논리적 발생순서를 표시하는 스레드 레이블에 의해 결정되는데, 이 스레드 레이블은 프로그램 수행에서 유일한 값을 가지기 때문에, 생성되는 모든 다른 스레드들과 구별할 수 있는 유일한 정보이다.

이러한 병렬프로그램의 각 스레드 레이블을 생성하기 위한 기법으로는 OS[15], EH[6], TR[6], NR[10], 그리고 SNR[24] 레이블링 등이 있다. 이들 레이블링 기법들의 공통점은 기억 공간 복잡도(space complexity) 및 수행 시간 복잡도(time complexity)를 모두 고려하여, 어떤 스레드의 레이블은 그 스레드의 조상 스레드의 정보를 유지해야 한다는 것이다. 그래서 수행중 경합 탐지는 프로그램 수행 중에 부모 스레드의 레이블에 의해 현재 스레드의 레이블을 생성하고, 생성된 레이블의 값을 비교함으로써, 스레드들 사이의 병행성을 결정할 수 있다. 따라서 최악의 경우에 모든 병행한 스레드에 대한 값을 유지해야 한다.

수행중에 경합을 탐지하기 위한 프로토콜은 병렬 프로그램의 각 스레드별로 고유하게 할당되는 레이블링 정보를 이용하여 이들의 병행성을 조사하여 탐지하는 데, 이러한 탐지 프로토콜은 지금까지 두 가지 측면에서 발전되어 왔다. 즉, 경합 검증(race verification) 기법[6,15]과 최초경합 탐지 기법[9,12,22]이다.

경합 검증 기법에서는 만약 대상으로 하는 병렬 프로그램에서 경합이 존재한다면 적어도 하나 이상의 경합이 존재한 다면 적어도 하나 이상의 경합은 보고된다는 것을 보장하지만, 처음으로 탐지된 경합이 그 병렬 프로그램에서 처음으로 발생한 경합이라는 것을 보장하지 못한다. Mellor-Crummey의 탐지 프로토콜[15]에서는 접근역사

의 크기를 상수 크기로 유지하며, 접근역사내의 가장 최근의 쓰기 접근과 읽기 접근 중에서, 가장 오른쪽에서 발생한 읽기 접근과 가장 왼쪽에서 발생한 읽기 접근만을 선택하여 유지함으로써, 수행중 적어도 하나의 경합을 탐지할 수 있은 기법이다. 이 프로토콜은 레이블링 기법 중에서 left-of 관계[15]를 만족하는 레이블링 기법[10,15]에만 적용될 수 있다. 이러한 기법은 기억 공간의 절약을 위해서 프로그램 모니터링 중에 수집한 정보를 삭제할 수 있는 데, 이러한 삭제는 접근 역사에 각 스레드에서 경합에 참여하는 두 개의 읽기와 하나의 쓰기 접근만을 저장한다. 이 때문에 각 스레드의 처음으로 발생하는 접근에 의한 경합을 보고하지 못하는 경우가 발생할 수 있다는 단점이 존재한다.

하지만 최초경합 탐지 기법[9,12,22]에서는 수행중 최초로 발생한 경합을 탐지하는 기법이다. 기존의 기법들에 적용된다는 단점이 아울러 존재한다. 확장성을 가지고 있는 최초경합 탐지 기법[12]은 프로그램 수행 중에 각 접근들과 비교하여 최초경합을 탐지하는 기법이다. 이는 최대 2개의 접근으로 구성된 접근역사가 각 스레드들에 분산되어 있고, 이를 이용하여 수행되기 때문에 기존의 공유 변수들의 접근에서 야기되는 병목현상을 감소시키는 장점을 가진다. 따라서 스레드들의 수가 증가하더라도 서로 분산되어 있는 접근들을 비교하기 때문에 시간 복잡도 면에서는 유리한 기법이다. 하지만 스레드들 간의 동기가 없는 비내포 병렬 프로그램만 대상으로 하기 때문에 그 적용 범위가 한정된다고 할 수 있다. 이 기법과는 달리 동기가 있는 내포 병렬성을 갖는 프로그램에서 최초경합을 탐지할 수 있는 기법[9,22]은 최초경합에 관련된 키(key) 접근 개념을 사용하여 좀 더 일반성이 있는 방법을 제공한다.

3.1 탐지의 효과

병렬 프로그램의 경합을 수행 중에 탐지하기 위해 사용되는 병행성 정보인 스레드 레이블은 접근이 나타나는 스레드의 논리적인 수행 순서를 표시하는 값이다. 스레드 레이블에 의한 기존의 수행 중 경합 탐지는 스레드내 특정 경합의 탐지에 의미가 있다.

특정 경합 탐지는 프로토콜에 의해 결정된다. 기존의 접근역사 삭제를 적용하는 프로토콜[15]은 각 스레드에서 나타나는 읽기 접근 중에서 마지막으로 나타나는 읽기 접근만을 고려하기 때문에, 특정 수행에서 주어진 입력에 대한 경합의 유무를 검정하는 효과를 제공한다. 그러나, 이 기법에 의해 탐지된 경합을 구성하는 접근은 이전에 수행된 접근과의 종속성[7]을 고려하지 않기 때문에 탐지된 경합의 종속성을 결정할 수 없다. 경합의 종속성은 병렬 프로그램의 수행에서 특정 경합의 제거에 따라 해당 공유 변수에 대한 접근 순서가 변하기 때문에, 이후에 나타나는 공유 변수에 대한 접근이 나타나는 공유 변수에 대한 접근이 나타나지 않게 할 수 있다. 이 경우에 일반적인 경합 검증 프로토콜을 적용하면 나타나지 않은 경합을 탐지하기 위해서 탐지과정을 반복적으로 수행하는 부담을 가지게 된다. 탐지된 경합의 제거는 경합을 구성하는 접근의 결정적인 수행을 유지하는 것으로, 공유 변수를 삭제하거나, 접근을 동기화 시킨다. 그러나, 동기화 명령의 경우에는 프로그램의 수행 순서를 변화시키는 요소로서 다른 경합을 유발할 수 있다.

Jun과 McDowell의 연구에서는 이후에 나타나는 경합의 발생에 영향을 줄 수 있는 특정 경합을 탐지하는 프로토콜[9]을 제시하였다. 이 프로토콜에 의한 경합 탐지는 경합의 존재 효과와 함께 임의의 내포 수준에서 발생되는 최초경합을 탐지

하는 효과를 가지고 있다. 최초경합의 제거는 이후에 나타나는 경합의 발생에 영향을 주어 나타나지 않게 할 수 있다.

탐지의 효과는 병렬 프로그램의 오류수정 과정에서 실용성에 영향을 주는 요소이다. 그러나, 탐지의 효과는 수행중 경합 탐지의 결과적인 요소로 수행중 경합 탐지에 대한 실용성에는 직접적인 영향이 크지 않다. 수행중 경합 탐지에서 나타나는 실용성을 결정하는 주요한 요소는 탐지 정보 생성 및 유지를 위한 기억공간의 크기와 병행성 정보를 생성하고, 경합을 결정하는 시간이다. 특히 대규모 병렬 프로그램에 대한 경합 탐지과정에서 필요한 기억 공간은 대규모 병렬 시스템에서조차도 비실용적인 크기를 나타낸다.

3.2 탐지의 효율

수행중 경합 탐지에 필요한 비용은 기억 공간과 경합 탐지 시간의 두 가지로서, 경합 탐지의 효율성에 주요한 요소이다. 이들 두 가지 요소의 통합적인 고려를 통해, 기법의 실용적 구현을 제공 할 수 있으며, 경합의 효과를 구분할 수 있게 한다. 개념적으로, 기억 공간 비용은 모든 공유 변수들에 대한 접근역사를 유지하기 위한 것과 병행 수행하는 스레드들의 레이블들을 위한 것으로 구성된다. 그리고, 경합 탐지에 필요한 시간적 비용은 접근 발생할 때마다 경합을 결정하고, 접근 역사를 유지하는 것과 각 스레드의 레이블 생성에 필요한 것이다.

이러한 비용 요소에서, 실제적으로 수행중 경합 탐지의 효율성에 직접적인 영향을 주는 것은 접근 사건에서의 시간 복잡도와 접근역사의 유지에 필요한 기억 공간 복잡도이다. 이는, 실제 병렬 프로그램에서 접근 사건의 복잡도가 공유 변수의 수와 접근이 발생하는 병행 스레드의 수에 영향을 받기 때문이다. 그러므로, 수행중 경합 탐지에 필요한 기억 공간과 시간은 다음 요소에 의해 결정된다[23].

이 요소들에 의한 기억 공간 복잡도와 시간 복잡도에 의해 기법의 효율성이 결정된다. 전체적으로 기억 장소 복잡도는, 모든 공유변수들의 접근역사와 병행 수행하는 모든 스레드들의 레이블을 위해 기억 공간으로 구성된다. 두 기억 공간의 복잡도는 Jun의 연구[23]에서와 같이 계산된다. 접근역사의 복잡도를 결정하는 요소 중에 공유 변수의 수는 프로그램의 고정적인 요소로, 복잡도 계산에서는 상수적인 영향을 준다. 그리고, 접근 역사의 항목 수는 프로토콜에 의해 결정되는 요소로, 상수적인 크기를 갖는 프로토콜[15]에 의해 기억 공간 복잡도에 상수적인 영향을 준다. 각 항목의 크기는 사용된 레이블의 크기에 의해 결정되는 요소로, 레이블의 복잡도 계산에서 레이블의 크기는 Junr과 Koh에서 상수적인 크기를 갖는 레이블[10]을 제안하였다. 그리고, 레이블링을 위한 부가적인 기억 장소는 일반적으로 상수의 크기이고, 덧셈의 요소이다. 그러나, 최대 병렬성은 생성되는 스레드의 수에 의해 결정되기 때문에 동적인 특성을 가지며, 일반적인 병렬 프로그램의 응용인 대규모 과학 기술 계산용 병렬 프로그램에서 최대 병렬성은 수백만 이상의 크기를 갖기 때문에, 다른 요소에 비해 상대적으로 일반적인 경우에 기억 장소 복잡도에 영향을 주는 것은 최대 병렬성으로, 이 요소의 제거는 전체 기억 장소 복잡도의 효율성에 중요한 요소가 된다.

수행중 경합 탐지에서, 시간 복잡도는 스레드들의 생성 또는 종료 시에 레이블을 생성하는 시간과 각 공유 변수에 대한 접근의 발생 시에, 각 변수를 위한 접근 역사의 검사와 유지를 위한 시간으로 구성된다. 두 가지 시간의 복잡도는 Jun의

연구[23]에서와 같이 계산된다. 시간 복잡도의 계산에서, 레이블 생성 시간 복잡도는 레이블의 크기와 공유 자료 구조에 대한 경합 시간에 의해 결정되는 것으로, 공유 자료에 대한 경합 시간은 레이블 생성 시에 공유 자료를 참조하는 병행 스레드들에 의해서만 유효한 요소이며[6], 레이블 생성 시에 공유 자료를 참조하지 않는 Jun과 Koh [10], Nudler과 Rudolph[18] 그리고 Jun[23]의 연구에서는 의미가 없다. 따라서, 레이블 생성 시간 복잡도는 레이블의 크기에 의해 결정된다고 할 수 있다. 기존의 레이블링 기법에서 레이블의 크기는 내포 수준에 의존한다[10,23]. 일반적으로, 내포수준은 다른 요소에 비해 상대적으로 매우 작은 값이다. 접근역사 검사와 유지 시간 복잡도는 접근역사내 항목 수와 레이블 비교 시간의 함수에 의해 계산된다. 여기서, 접근역사내 항목 수는 상수적인 크기를 가지며, 레이블 비교 시간은 현재 레이블의 크기에 의해 결정된다. 기존의 기법에서, 시간 복잡도는 내포 수준에 의해 결정되는 것이다. 일반적인 대규모 병렬 프로그램에서 내포 수준은 최대 병렬성에 비해 무시할 정도의 값이다.

이와 같이 병렬 프로그램의 수행중 경합 탐지 기법의 효율성은 적용된 레이블과 프로토콜에 대해 최대 병렬성 요소를 제거하면 실용적인 측면의 적용이 가능하다. 최대 병렬성은 스레드가 병행으로 생성되지 않는다면 제거된다. 내포 루프병렬 프로그램에서 스레드가 결정적으로 생성되면, 스레드들에 대한 병렬성을 제거할 수 있다. 루프들의 결정적인 수행은 루프 지수 순서에 의한 수행이다.

수행중 경합 탐지 기법의 실용성은 경합탐지 도구의 실용화에 지대한 영향을 준다. 수행중 탐지 기법에 의해 탐지된 경합은 적용된 레이블과 프로토콜에 따라 다른 효과를 제공한다. 탐지의 결과에 의한 경합의 효과에 따라서 병렬 프로그램의 오류 수정에 영향을 줄 수 있다. 수행중 경합 탐지 기법은 수행중에 프로그램을 모니터링하여 병행성 정보를 생성하고, 이 정보에 의해 경합의 발생을 탐지, 보고하는 기법이다. 수행중 경합 탐지 기법은 기본적으로 병행한 스레드들에 대한 병행 모니터링 비용을 필요로 한다. 병행 모니터링 비용은, 일반적인 대규모 과학 기술용 병렬 프로그램에 대한 수행중 경합 탐지 기법의 실용성에 영향을 주는 요소이다. 특히, 병행 모니터링에 의한 기억 공간 비용은 수행중 경합 탐지에 대한 비실용적인 비용을 필요로 한다. 병행 모니터링에 대한 비용은 병렬 프로그램의 수행에서 최대 병렬성을 제거하면 실용적인 크기로 유지된다.

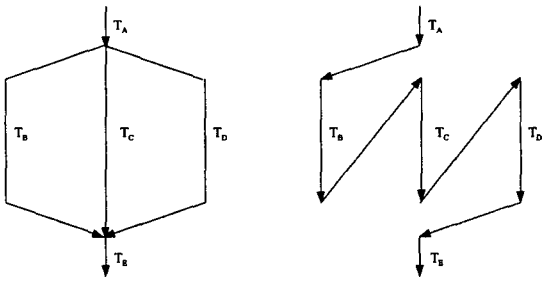
4. 실용적 경합 탐지 기법

병렬 프로그램의 수행중 경합 탐지는 프로그램의 최대 병렬성에 의존적인 기억 공간 비용을 필요로 한다. 이러한 기억 공간 비용은 수행중 경험 탐지의 실용성에 영향을 주는 주요 요소이다. 기존의 병행 모니터링에 의한 기억 공간 문제점을 개선하기 위한 방법으로 병렬성에 무관한 순차적 모니터링(sequential monitoring)을 위한 순차적 수행 제어에 대해 설명한다.

시간 효율성을 가지는 기존의 경합 탐지 기법을 위한 프로그램 모니터링은 병행 모니터링을 필요로 한다. 병행 모니터링은 병행 수행하는 모든 병행 스레드들을 동시에 모니터링하기 때문에 프로그램의 최대 병렬성에 의존적인 기억 공간 비용을 가진다. 기억 공간의 비효율성은 수행중 탐지 기법의 실용성에 부정적인 영향을 줄 수 있다. 이러한 병행 모니터링의 기억 공간의 비효율성을 제거하기 위해서는 프로그램의 스레드들이

결정적인 순서로 수행되어야 한다. 결정적인 순서에 의한 모니터링은 한번에 하나의 스레드만을 모니터링하기 때문에 병렬성과는 무관한 기억 공간 비용을 갖는다. 결정적인 수행 순서는 기억 공간의 효율성뿐만 아니라, 동일 입력에 대한 다수의 수행에서 스레드 또는 접근사건들이 동일한 수행 순서의 재생산(replay)을 제공하는 특성이 있다.

그림 4는 프로그램의 병렬 수행과 순차 수행을 POEG의 형태로 나타낸 것이다. 그림 4(a)는 병렬 수행을 나타낸 것으로 TB, TC, TD는 병행하게 수행할 수 있다. 이러한 프로그램에 대한 순차 수행을 나타낸 것이 그림 4(b)이다. 순차수행은 병렬 수행과는 달리 스레드들을 순차적으로 수행한다. Fig. 그림 4(b)의 수행순서는 TB가 수행된 이후에 TC 그리고 TD가 순차적으로 수행한다. 이러한 프로그램 수행은 병렬 프로그램의 병행 수행에서 나타나는 한 특정한 경우라 할 수 있다. 그리고 순차 수행에서도 각 스레드들의 논리적 병렬성은 병렬 수행에서와 동일하게 유지된다. 그러므로, 병렬 수행에서 나타날 수 있는 경합을 순차 수행에서도 탐지할 수 있는 것이다. 순차 수행 제어는 상수적 기억 공간과 시간 비용만을 필요로 하기 때문에 경합 탐지의 효율성에 부가적인 영향을 주지 않는다. 최대 병렬성을 나타내지 않는 순차 수행은 수행중 경합 탐지에서 기억공간의 효율성을 증대시키는 요소이다. 순차 수행에서는 한번에 하나의 스레드에 대해서만 경합 탐지 과정을 수행하기 때문에, 경합 탐지 과정에 유지되는 정보는 하나의 자료 구조만을 필요로 한다.



(a) 병렬 모니터링 (b) 순차적 모니터링

그림 4. 프로그램 수행 모델

5. 공간 효율적 경합 탐지

수행중 경합 탐지 기법은 병렬 프로그램에서 경합을 탐지하기 위해 원시 코드에 경합 발생을 모니터링할 부가 코드를 추가한다. 추가된 코드는 병렬 프로그램 수행과 동시에 공유변수에 대한 모니터링을 한다. 이렇게 수행 중에 공유변수를 감시하는 것을 모니터링이라 하며, 방법으로는 병렬 모니터링과 순차 모니터링 두 가지가 있다.

병렬 모니터링은 병행 수행되는 스레드 수만큼 모니터링 코드가 병행적으로 탐지를 위한 논리적 병행성을 체크하기 위해 수행 중에 공유변수에 대한 접근 정보와 레이블을 정보 유지를 위한 기억공간을 필요로 하게 된다. 따라서 병렬 모니터링에서는 각 모니터링 코드가 동시에 병행적으로 수행하므로 필요로 하는 기억 공간이 최대 병렬성에 비례하여 증가하게 된다. 또한 동기화를 가진 병렬 프로그램에서는 요구되는 공간 복잡도는 비현실적이다. 순차 모니터링은 병행 수행되는 스레드 수와는 무관하게 한 순간에 오직 하나의 모니터링 코드가 수행된다. 이 기법은 병렬 프로그램 수행에서 각 스레드가 순차적으로 수행하는 것이 모든 가능한 병렬 수행의 한 경우라 할 수 있기 때문에 가능하다. 순차 모니터링은 각 스레드가 순차적으로 수행되도록 하여 하나의 스레드씩 순차적으로 모니터링 함으로써 경합을 탐지한다. 따라서 병렬모니터링과 달리 한 순간에 오직 한 스레드에 대한 공유변수 접근 정보와 레이블 정보 유지를 위한 기억 공간만을 필요함으로 공간 복잡

도를 상당히 줄일 수 있다.

기존의 기법들을 비교하면 병렬 모니터링을 위한 Task-Recycling[6], Clock-Tree[2], Nest-Region[10]과 순차모니터링을 위한 Access-Set [3], SNR[23]이다. 기존의 기법들의 효율성은다음과 같은 인수들에 의해 평가된다.

V : 공유변수의 수
N : 내포깊이
T : 최대 병렬성

표 1에서 보는 것과 같이 스레드간의 동기화가 존재하지 않는 경우는 병렬 모니터링에서 프로그램의 최대 병렬성인 T 인자에 기인한 공간 복잡도를 요구하지만 순차 모니터링은 T 인자와 무관한 공간 복잡도를 보인다. 스레드간 동기화가 존재하는 경우에서도 병렬 모니터링에서 프로그램의 최대 병렬성인T 인자에 기인한 공간 복잡도를 요구한다. 순차 모니터링은 적용되지 못하지만 만약 적용시킬 수가 있다면 T 인자와 무관한 공간 복잡도를 보일 것이다. 그러나, 순차적 모니터링 기법은 임의적 동기화를 가지는 병렬 프로그램에는 교착상태를 방생시키는 문제점을 가지고있다. 그래서 효율적인 공간 복잡도를 가지는 경합 탐지를 위해 임의적 동기화를 가지는 병렬 프로그램에서 순차적 모니터링 기법을 적용할 수 있도록 프로그램을 재구성하는 기법이 제안되어져 왔다.

6. 병렬 프로그램 재구성 기법

임의적 동기화를 가지는 병렬 프로그램의 수행중에 경합을 탐지할 때, 최초경합 탐지에 도움을 주고, 최대 병렬성과 무관한 기억 공간 효율적인 순차적 모니터링이 가능한 형태로 재구성하는 기법을 소개하면 다음과 같다.

동기화가 있는 병렬 프로그램에서도 기존의 순차적 모니터링으로 경합을 탐지할 수 있다면, 공간 복잡도를 현저히 감소시킬 수 있다. 예를 들어, 모니터링 대상이 되는 공유변수의 수를 V라 하고, 최대 병렬성을 T라 할 때 TR 기법[6]의 경우 최악의 공간 복잡도는 $O(VT+T^2)$이다. 그러나 순차적인 모니터링을 하게 되면 최악의 공간 복잡도는 $O(VT)$가 된다. 경합을 탐지하는 기존의 수행중 기법은 임의적 사건 동기화가 있는 병렬 프로그램에서의 최초경합 탐지를 보장하지 못한다. 그러나 재구성을 하면 순서적 동기화에서 최초경합을 탐지하는 알고리즘[20]을 사용하여 최초경합을 탐지해 낼 수 있다. 디버깅에 있어 최초경합의 의미는 중요하다. 왜냐하면 최초경합의 제거에 의해 다른 여러 경합들도 자동적으로 제거될 수도 있기 때문이다.

6.1 단일 사건의 재구성

동기화가 있는 병렬 프로그램의 수행에서, 스

표 1. 모니터링 기법에 대한 최악의 공간 복잡도

	Technique	Inter-thread Coordination	
		No	Yes
Parallel	Task-Recycling	$O(VT+T^2)$	$O(VT+T^2)$
	Clock Tree	$O(V+NT)$	$O(VT+T^2)$
	Nest-Region (NR)	$O(V+NT)$	NA
Sequential	Access-Set	$O(VN)$	NA
	SNR	$O(V+N)$	NA

레드들은 병렬 명령에 의해 생성 및 종료된다. 동일한 생성명령(fork operation)에 의해 생성된 스레드들은 대응하는 합류명령(join operation)이 수행될 때까지 병렬로 수행한다. 이러한 생성과 합류 명령을 스레드 명령이라 한다. 그리고, post와 wait 명령은 사건 동기화를 나타낸다. post 명령은 사건의 발생을 나타내고, wait 명령에 의해 동기화가 된다.

병렬 프로그램은 병렬 루프(parallel loop)와 병렬 섹션(parallel section) 등의 병렬 형태를 가지고 있는데, 여기에서는 병렬 프로그램은 사건 동기화가 있는 병렬 루프와 병렬 섹션에 대해서 하나의 사건 변수(event variable)에 의한 동기화를 갖는 병렬 프로그램의 재구성하는 기법을 설명한다. 사건 변수는 두 가지 상태(clear, posted)를 가지며, 초기 값은 항상 clear이고 post 명령에 의해 posted 상태가 된다. posted 상태가 된 값은 wait 명령에 의해 비교되어 지는데 wait 명령은 사건 변수 값이 post 명령을 통해 posted 상태가 될 때까지 해당 스레드의 실행을 지연하게 된다. 반면에 post 명령은 수행후 즉시 다음 실행을 계속한다.

임의적 동기화가 있는 병렬 프로그램에서 순차적 모니터링을 할 때의 문제점은 wait 명령에 의해 교착상태에 빠진다는 점이다. 그래서 교착상태에 빠지지 않으면서 공간 효율적인 순차적 모니터링이 가능하도록 부가 코드가 추가된 원시 병렬 프로그램을 재구성하는 것이다.

단일 사건의 재구성은 병렬 스레드의 각 스레드의 첫 번째 wait 문을 기준으로 병렬 프로그램을 분리시킨다. 그림 5는 재구성하기 전의 병렬 프로그램을 POEG으로 나타낸 것이며, 여기에서 B_i는 동적블록을 나타낸다. 이것을 그림 6과 같이 병렬 프로그램의 시맨틱과 병행성을 검사하기 위한 부가적인 코드의 수행 순서는 그대로 유지하면서 재구성한다.

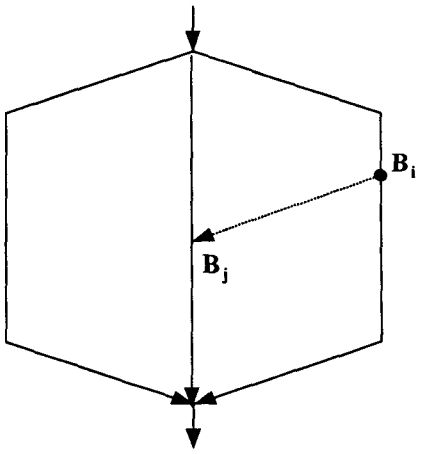


그림 5. 재구성 이전의 POEG

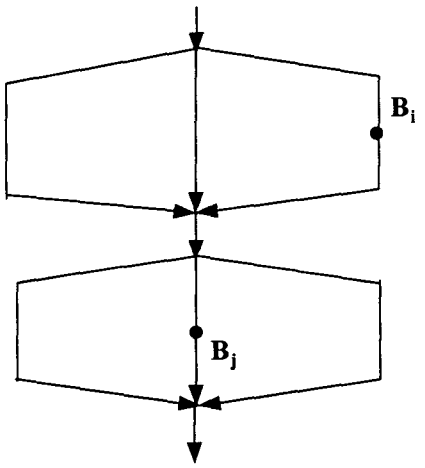


그림 6. 재구성 후의 POEG

7. 결 론

수행중 경합 탐지 기법은 공유변수에 접근하는 스레드들 간의 논리적 병행 여부를 결정하기 위해서, 원시 프로그램에는 경합 탐지에 필요한 부가적인 코드를 추가한다. 그리고 실행 중에는 공유변수에 접근하는 각 스레드들을 모니터링하고, 공유변수에 대한 스레드들의 접근역사를 유지하게 된다. 이때 필요한 정보를 생성하고 저장하기 위해 요구되는 공간은 최대 병렬성에 의존적으로

증가한다. 더구나 동기화가 있는 병렬 프로그램인 경우는 동기화 정보까지 유지해야 하므로 기억 공간 문제는 더욱 심각하다.

그래서 수행중 기법에서는 경합을 탐지할 때 기억 공간을 줄일 수 있는 방안으로 순차적 모니터링이 제안되어져 왔고, 동기화를 가지는 병렬 프로그램에서 공간 효율적인 수행중 경합 탐지를 위한 순차적 모니터링이 가능하도록 원시 프로그램을 재구성하는 기법을 소개했다.

향후 연구 과제로서는 다중 사건을 재구성할 때 필요한 사건 변수에 의존적인 루프를 감소시키는 방안과 함께 임의적 동기화를 가지는 병렬 프로그램을 위한 레이블링 기법을 제시하는 것이다. 이렇게 함으로써 현재 사용되는 대부분의 병렬 프로그램에 대한 최초경합 탐지를 지원할 수 있다.

참 고 문 헌

[1] Allen, J. R. and Ken Kennedy, "*A Parallel Programming Environment*," Software, IEEE, July 1987.

[2] Audenaert, K., "*Clock Trees: Logical Clocks for Programs with Nested Parallelism*," Tr. on Software Engineering, 23(10): 646-658, IEEE, Oct. 1997.

[3] Audenaert, K., and L. Levrouw, "*Space Efficient Data Race Detection for Parallel Programs with Series-Parallel Task Graphs*," 3rd Euromicro Workshop on Parallel and Distributed Processing, pp. 508-515, 1995.

[4] Callahan, D. Kennedy K., Subhlok, J., "*Analysis of Event Synchronization in a Parallel Programming Tool*," 2nd Symp. on Principles and Practice of Parallel Programming, pp. 21-30, ACM, March 1990.

[5] Dagum, L., and R. Menon, "*OpenMP: An Industry-Standard API for Shared- Memory Programming*," Computational Science and Engineering, Vol. 5(1), pp. 46-55, IEEE, Jan.-Mar. 1998.

[6] Dinning, A., and E. schonberg, "*An Empirical Comparison of Monitoring Algorithms for Access Anomaly Detection*," 2nd Symp. on Principles and Practice of Parallel Programming, pp. 1-10, ACM, Mar. 1990.

[7] Emrath. P. A., and D. A. Padua, "*Automatic Detection of Non- determinacy in Parallel Programs*," 1st Workshop on Parallel and Distributed Debugging, pp. 89-99, ACM, May 1988.

[8] Grunwald, D., H. Srinivasan, "*Efficient Computation of Precedence Information in Parallel Programs*," 6th Workshop on Languages and Compilers for Parallel Computing, pp. 602-616, Springer-Verlag, Aug. 1993.

[9] Jun, Y., C. E. McDowell, "*On-the-fly Detection of the First Races in Porgrams with Nested Parallelism*," 2nd Int'l Conf. on Parallel and Distributed Processing Techniques and Applications, pp. 1549-1560, CSREA, Aug. 1996.

[10] Jun, Y., and K. Koh, "*On-the-fly Detection of Access Anomalies in Nested Parallel Loops*," 3rd Workshop on Parallel and Distributed Debugging, pp. 107-117, 1993.

[11] Kim, D., and Y. Jun, "*An Effective Tool for Debugging Races in Parallel Programs*," 3rd Int'l Conf. on Parallel and Distributed Processing Techniques and Applications, pp. 117-126, CSREA, June 1997.

[12] Kim, J., and Y. Jun, "*On-the-fly Detection of the First Races by Summary Report Method*," Proc. of the 4th KISS Youngnam Branch Conference, 4(1): 63-73, KISS, Feb. 1997.

[13] Kim, Y., and Y. Jun, "*Restructuring Parallel Programs for On-the-fly Race Detection*," 5th Int'l. Conf. Parallel Computing Technologies (PaCT-99), pp. 446-452, RAS, Sept. 1999.

[14] Loveman, D. B., "*High Performance Fortran*," Parallel and Distributed Technology, 1(1): 25-

42, IEEE, Feb. 1993.

[15] Mellor-Crummey, J. M., "*On-the-fly Detection of Data Races for Programs with Nested Fork-Join Parallelism*," Supercomputing '91, pp. 24-33, ACM/IEEE, Nov. 1991.

[16] Netzer, R. H. B., S. Ghosh, "*Efficient Race Condition Detection for Shared-Memory Programs with Post/Wait Synchronization*," Int'l. Conf. on Parallel Processing, pp. II-242-246, Penn. State Univ., Aug. 1992.

[17] Netzer, R. H. B., B. P. Miller, "*Improving the Accuracy of Data Race Detection*," 3rd Symp. on Principles and Practice of Parallel Programming, pp. 133-144, ACM, April 1991.

[18] Nudler, I., and L. Rudolph, "*Tool for the Efficient Development of Efficient Parallel Programs*," 1st Israeli Conf. on Computer System Engineering, 1998.

[19] *OpenMP Fortran Application Program Interface,* OpenMP Architecture Review Board, Oct. 1997.

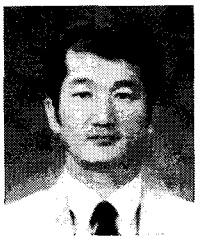
[20] Park, H., and Y. Jun, "*Two-Pass On-the-fly Detection of the First Races in Shared-Memory Parallel Programs*," Proc. of the SIGMETRICS Symposium of the Parallel and Distributed Tools, ACM, pp. 158, Aug. 1998.

[21] Park, H., Y. Jun, "*Detecting the First Races in Parallel Programs with Ordered Synchronization*," 6th Intl. Conf. on Parallel and Distributed Systems, pp. 201-208, IEEE, Dec. 1998.

[22] *Parallel Computing Forum*, PCF Parallel Fortran Extensions, Vol. 10(3), ACM, July 1991.

[23] 전용기, "*병렬 프로그램의 접근 이상 탐지*", 박사학위논문, 서울대학교, 1993. 8.

[24] 정춘화, 김정시, 전용기, "*병렬 프로그램의 경합 수정을 위한 순차적 감시 도구*", 한국정보과학회 병렬처리시스템연구회 추계학술발표회 논문집, 8(3): 25-32, 한국정보과학회, 1997. 10.



김 영 철

- 1990년 경상대학교 전산통계학과 학사졸업
- 1992년 경상대학교 전자계산학과 공학석사졸업
- 2000년 경상대학교 전자계산학과 공학박사졸업
- 1992년~현재 진주국제대학교 교수
- 관심분야 : 분산 및 병렬처리, 운영체제, 멀티미디어 운영체제