# 다중스레드 프로그램의 디버깅을 위한 사건순서 정보

이승열* · 강문혜* · 김동국**

## 1. 서 론

현재 다중스레드를 지원하는 컴퓨터의 운영환경이 일반화됨에 따라 이를 지원하기 위한 스레드 프로그래밍 환경 또한 확대되고 있다. 특히, C와 Fortran 등과 같은 기존의 언어들도 다중스레드 프로그래밍 방법을 지원하기 위해 확장되었으며[11], 비교적 최근에 발표된 Java 등과 같은 언어들은 다중스레드를 처리하기 위한 기능들을 내장하고 있다[4,8].

이와 같은 다중스레드 프로그래밍 환경은 데스크탑 컴퓨터에서 그래픽 사용자 인터페이스가 표준화되면서 대중화되기 시작하였다. 왜냐하면, 다중스레드 시스템은 사용자로부터의 입력이나 화면 출력 등을 전담하는 스레드를 따로 가지고 있기 때문에, 사용자들에게 프로그램의 성능이 향상된 것처럼 보이기 때문이다. 또한 최근 들어, 다중프로세서를 가진 컴퓨터가 일반인을 대상으로 하는 데스크탑 시장에도 출현함에 따라 이러한 컴퓨터 자원을 활용할 수 있는 다중스레드를 기반으로 하는 응용프로그램의 개발이 일반화되고 있다.

이러한 다중스레드 프로그래밍 환경에서는 전통적인 순차적 프로그램과 달리 다중스레드의 특성으로 인하여 경합(race), 교착상태(dead lock) 등을 발생할 수 있다[1,6]. 이러한 문제들은 주로 다수의 스레드가 하나의 자원을 공동으로 사용함으로써 발생할 수 있다. 여기서 경합은 두 개 이상의 병행한 스레드가 어떤 공유메모리에 하나의 쓰기 사건을 가지고 접근할 때 발생할 수 있는 것으로 의도하지 않은 프로그램의 수행 결과를 볼일 수 있다. 또한 이러한 경합을 피하기 위해 각 스레드가 공유자원에 대한 사용권을 배타적으로 가질 수 있도록 할 수 있는데 이는 하나 이상의 스레드가 더 이상 계속할 수 없는 어떤 특정 사건(자원의 할당과 해제)을 기다리는 교착상태가 발생될 수 있다.

위에서 언급한 경합, 교착상태 등은 프로그램의 수행 결과 및 수행 자체에 심각한 영향을 줄 수 있으므로, 다중스레드 프로그램 개발과정에서 이러한 오류를 탐지하여서 수정해야 한다. 그러나 이러한 오류를 탐지하는 것은 다중스레드 프로그램의 비결정적인 수행 특성으로 인하여 순차적 프로그램의 오류탐지보다 어렵다. 왜냐하면, 순차적 프로그램은 동일한 입력으로 동일한 수행을 반복적으로 감시할 수 있어 중단점(breakpoint)을 이용하여 단계적으로 오류의 원인을 탐지할 수 있지만, 다중스레드 프로그램은 각 스레드의 수행이 비결정적이기 때문에 이러한 방식을 적용하기 어렵다.

다중스레드 프로그램의 이러한 오류들을 탐지

* 경상대학교 자연과학대학 컴퓨터과학과 박사과정
** 애스톤리눅스 대표이사

하기 위해서는 일반적으로 사건정보를 이용한다. 여기서 말하는 사건은 프로그램의 수행에 있어서 스레드들 간의 논리적 순서관계에 영향을 줄 수 있는 스레드의 생성, 종료, 동기화 등의 사건을 의미한다. 이러한 사건들과 연계된 스레드들의 수행 명령어들은 병행하게 수행되거나 또는 순차적으로 수행될 수 있다. 따라서 이러한 사건정보를 이용하여 공유메모리에 접근하는 사건들 간의 순서관계를 분석하여 오류를 탐지할 수 있다. 이러한 사건정보를 생성하는 대표적인 기법으로는 English-Hebrew Labeling[7], Task Recycling[2], Offset-Span Labeling[5], Nest-Region Labeling[3] 등이 있으며 이러한 사건정보 생성기법은 그 특성에 따라 적용되는 대상 프로그램 모델이 다르다. 또한 이러한 사건정보를 기반으로 하여 다중스레드 프로그램 개발 시에 이용될 수 있는 상용 디버깅 도구들도 개발되어 있다. 대표적인 상용 도구로는 현재 C와 Fortran 프로그램 언어를 기반으로 하여 다중스레드의 기능을 지원하는 OpenMP를 대상으로 하는 인텔(Intel)의 Thread Checker[9] 등과 다중스레드 Java를 대상으로 하는 KL 그룹의 Jprobe[10] 등을 있다.

2장에서는 다중스레드 프로그램의 특성으로 발생할 수 있는 오류인 경합과 교착상태에 대해 간단히 알아보고, 3장에서는 이러한 오류를 탐지하기 위해 필요한 사건들 간의 순서관계 정보를 생성하는 대표적인 기법들을 알아보고, 마지막 장에서 결론을 맺는다.

## 2. 다중스레드 프로그램의 오류

다중스레드 프로그램에서 스레드들은 스레드 명령(thread operation)에 의해 시작 혹은 종료되고, 시작된 스레드는 자신에게 할당된 문장들을 수행한다. 이러한 스레드들은 각기 독립적인 자료구조를 가지고 병행하게 수행될 수 있다. 또한 다수의 스레드가 어떤 연관관계를 가지면서 동작해야 할 경우에는 공유 자료구조를 이용하여 구현할 수 있으며, 이러한 공유 자료구조를 이용할 때 스레드들 간의 순서관계를 제어할 수 있도록 일반적인 다중스레드 프로그램 언어에서는 신호(post) 및 대기(wait) 명령을 제공한다. 여기서 대기 명령은 수행하는 스레드는 다른 스레드의 신호 명령이 수행될 때까지 대기해야 하지만, 신호 명령을 수행하는 스레드는 대기 명령을 수행하는 스레드의 상태와 무관하게 수행한다.

그림 1은 다중스레드 프로그램으로 여기서, T1와 T2는 스레드 블록(block)을 의미한다. 각 스레드의 블록에 포함된 문장 내의 X와 Y는 각 스레드가 접근 가능한 공유변수를 의미한다. 각 스레드들은 병행하게 수행되므로 T1의 공유변수 X와 T2의 공유변수 X는 두 스레드에 의해 동시에 접근이 가능하므로 공유변수 X에 대한 스레드 T1과 T2의 접근 순서는 여러 가지 경우로 발생할 수 있다. 만약, T1 스레드의 'X ='라는 수행문장에서 '1'라는 정수를 할당 받았다고 가정해보자. 그리고 T1 스레드의 그 다음 문장인 '= X'를 수행한

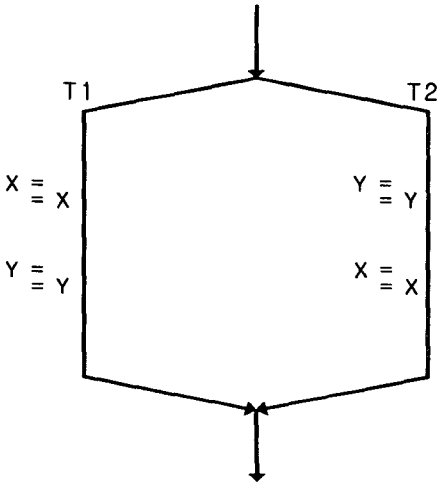



그림 1. 다중스레드 프로그램의 경합

다면, X의 값은 '1'일 것이다. 하지만 T1 스레드의 '= X'라는 문장을 수행하기 이전에 T2 스레드의 'X ='라는 문장을 수행하여 '2'라는 값을 할당하고 그 이후에 T1 스레드의 '= X'라는 문장을 수행했다면 X의 값은 '2'일 것이다. 그러므로 공유변수 X의 값은 각 스레드의 접근 순서에 따라 그 값이 비결정적으로 나타날 수 있으므로 이러한 공유변수를 참조하는 각 스레드의 수행 문장들은 프로그래머가 의도하지 않은 결과 값을 볼일 수 있다. 경합이라 위의 예에서 보인 것처럼 동일 공유변수에 두 개 이상의 병행한 스레드들 적어도 하나의 쓰기 사건을 가지고 접근할 때 발생할 수 있다. 그림 1에서는 공유변수 X에 대해 스레드 T1, T2가 경합을 보이고 있다. 마찬가지로, 공유변수 Y에 대해서도 스레드 T1과 T2에서 경합을 보이고 있다.

그림 1에서와 같은 문제를 해결하기 위해 프로그래머가 임의의 스레드가 공유변수에 어떤 값을 저장하고 이를 참조하여 다른 연산을 수행할 때까지 공유변수의 값이 유지되는 것을 보장할 수 있도록 하고 그 이외의 비결정적인 공유변수의 접근은 허락하도록 동기화를 구현하다고 해보자. 그림 2와 같이 스레드 T1, T2에서 접근하는 공유변수 X와 Y에 대해 각각 쓰기 사건과 읽기 사건을 원자적(atomic)으로 처리하기 위해 관련되는 영역을 임계영역(critical section)으로 지정하여 각 공유변수에 대해 각 스레드가 상호 배타적으로 접근할 수 있도록 동기화 명령을 이용하여 구현할 수 있다. 그림 2에서 공유변수 X와 Y에 대해 각각의 동기화 명령을 사용하여 구현했다고 가정하고, '[ ]', '( )'와 같은 기호로 각기 다른 임계영역(critical section)을 표현하였다.

그림 2와 같이 구현된 다중스레드 프로그램을 실행시켜 그림 3와 같은 수행 구조가 발생했다고 가정해 보자. 그림 3의 수행 A는 스레드 T1이 먼

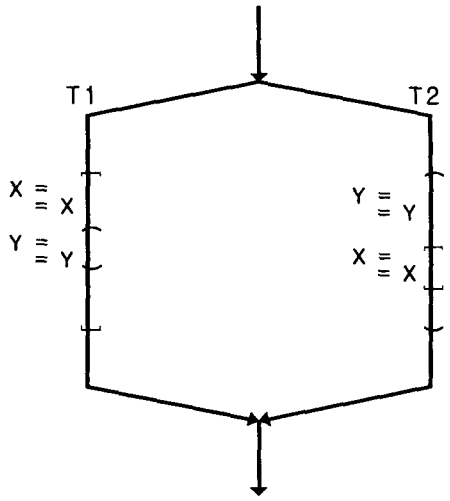



그림 2. 다중스레드 프로그램의 동기화

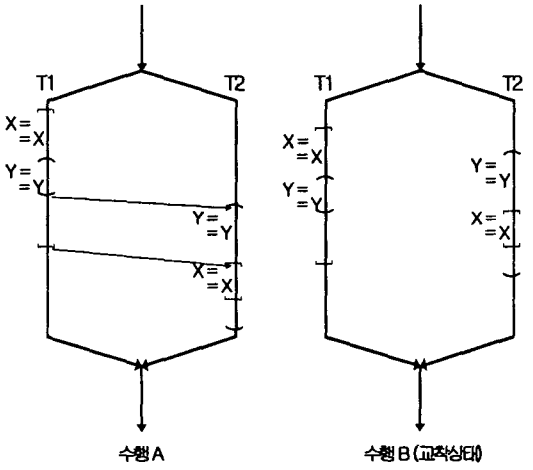



그림 3. 다중스레드 프로그램의 교착상태

저 공유변수 X에 대한 배타적 사용권을 확보하여 X에 대한 연산 명령을 처리한 후, 공유변수 Y에 대한 배타적 사용권을 또한 확보하여 Y에 대한 연산을 처리한 후에 Y에 대한 사용권을 T2 스레드에게 양보한다. T2가 공유변수 Y에 대한 사용권을 확보하여 Y에 대한 연산을 처리한다. 또한 스레드 T1이 공유변수 X에 대한 사용권을 스레드 T2에게 전달하고, 이를 받은 스레드 T2는 공유변수 X에 대한 연산을 처리한다. 따라서 그림

3의 수행 A는 프로그래머의 의도대로 두 스레드가 공유변수에 대한 배타적 접근을 하면서 정상적으로 수행된다.

그러나 그림 3의 수행 B와 같은 수행 형태에서는 교착상태라는 심각한 문제가 발생한다. 즉스레드 T1이 공유변수 X에 대한 배타적 사용권을 먼저 확보하고 X에 대한 연산을 처리하고 있을 때, 스레드 T2가 공유변수 Y에 대한 배타적 사용권을 확보하였다. 즉, T1 스레드가 공유변수 X에 대한 사용권을 가지고 있으면서 공유변수 Y에 대한 사용권을 요구하고 있는 상황이고 또한 스레드 T2는 공유변수 Y에 대한 사용권을 가지고 있으면서 공유변수 X에 대한 사용권을 요구하고 있는 상태이다. 따라서 스레드 T1과 T2를 자신이 확보한 공유변수에 대한 사용권을 가지고 있으면서 또 다른 공유변수에 대한 사용권을 요구하고 있기 때문에 서로가 발생할 수 없는 사건을 무한정 기다리는 교착상태가 발생한다.

위에서 언급한 다중스레드 프로그램의 오류인 경합, 교착상태 등은 다중스레드 프로그램의 비결정적인 수행 특성으로 인하여 발생할 수 있는 것으로 프로그램의 결과 및 수행에 심각한 영향을 줄 수 있으므로 이를 탐지하여 수정하는 것은 중요하다. 그러나 복잡한 다중스레드 프로그램에서는 이러한 오류를 탐지하는 것이 기존의 순차적 프로그램에서와 같이 수행의 재반복을 통하여 오류를 탐지하는 것은 어렵다. 왜냐하면, 그림 3의 수행 예에서와 같이 다중스레드의 비결정적이 수행 특성으로 인하여 다양한 수행 형태가 발생할 수 있기 때문에 이러한 모든 경우의 수행을 고려하여 각각의 수행을 분석하는 것은 비현실적이 시간과 공간의 비용이 요구된다. 따라서 공유변수 등과 같은 공유자원에 대해 접근하는 사건들의 순서정보를 분석하므로써 이러한 문제를 해결할 수 있다. 이를 위해서는 먼저, 다중스레드 프로그램에서 사건들 간의 논리적 순서관계 정보를 생성할 수 있는 기법이 필요하다. 따라서 3장에서는 이러한 정보를 생성시키는 대표적인 기법들을 소개한다.

## 3. 사건정보 생성기법

다중스레드 프로그램에서 비결정적인 수행 특성으로 인해 발생할 수 있는 오류인 경합, 교착상태 등을 탐지하기 위해서는 공유메모리에 접근하는 사건들 간의 순서관계를 분석하여 오류의 원인을 탐지할 수 있다. 그러므로 사건들 간의 순서관계 정보를 생성하는 기법은 다중스레드 프로그램의 오류를 탐지하는데 있어 중요한 자료로 이용된다. 본 장에서는 이와 같은 사건들의 순서관계 정보를 생성하는 대표적인 기법인 English-Hebrew Labeling, Task Recycling, Offset-Span Labeling, Nest-Region Labeling을 소개한다. 앞으로 소개하는 사건정보 생성기법은 루프 기반의 구조화된 다중스레드 프로그램에 효율적으로 적용될 수 있도록 개발된 것이다. 특히 Nest-Region Labeling은 동기화가 없는 구조화된 내포 다중스레드 프로그램에서 높은 효율성을 가지고 있으며, English-Hebrew Labeling은 Java와 같은 비구조적인 다중스레드 프로그램에서도 적용될 수 있는 특성을 가진다.

### 3.1 English-Hebrew Labeling

English-Hebrew (EH) 레이블링은 English Label E와 Hebrew Label H의 두가지 레이블로 구성된다. English 레이블은 그림 4에서와 같이 왼쪽에서 오른쪽으로 선순서화(preorder)에 의해 생성되는 것으로, 각 스레드의 레이블 값은 자식 스레드와 오른쪽의 형제 스레드에게 할당되는 레

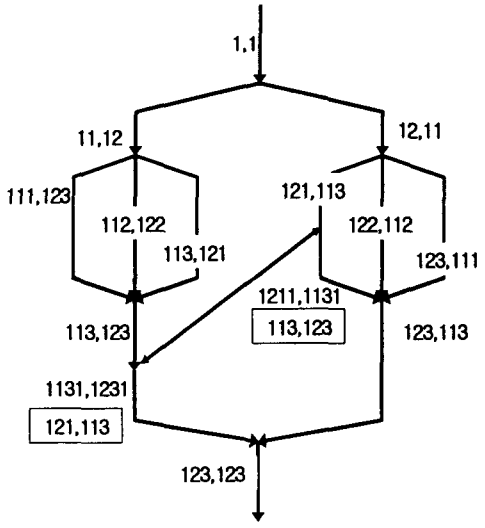



그림 4. English-Hebrew Labeling

이블 값보다 작은 값이 할당된다. 그러나 레이블은 수행 중에 생성되어야 하기 때문에 수행 중에 모든 레이블을 탐색할 수 없다. 그러므로 레이블은 일련의 숫자들의 열이고, 사전적인 순서(lexicographically)에 따른 순서화가 된다. 만약 새로운 스레드가 발생하면. 부모 스레드의 레이블 정보에 각 스레드 자신의 루프 지수가 첨부(append)되어 각 스레드의 레이블이 생성된다. 또한 동기화 명령에 의한 스레드는 부모 스레드의 English 레이블 값에 '1'를 첨부하고 동기화된 스레드의 정보를 부가적으로 유지하고 있어야 한다. 그리고 생성된 스레드들이 합류시키는 사건이 발생하면, 합류전 스레드들의 값들 중에서 가장 큰 값을 선택하여 레이블 값이 할당된다.

Hebrew 레이블은 English 레이블에서와 유사하게 오른쪽에서 왼쪽으로 대칭적으로 생성된다. 차이점은 Hebrew 레이블의 마지막 요소의 값이 해당 루프의 스레드의 수에 자신의 루프 지수를 빼고 그 값에 '1'를 더하는 것이다. EH 레이블은 스레드 생성 또는 동기화 사건마다 부모 스레드의 레이블에 새로운 값을 첨부하기 때문에 레이블의 길이는 스레드 생성사건과 동기화 사건의 수에 비례하여 증가된다.

이러한 방법으로 생성된 레이블을 각각 label_i와 label_j라 할때, 아래의 조건 중에 하나만을 만족하면 두 사건은 병행한 사건이다.

E(label_i)<E(label_j) and H(label_i)>H(label_j)
또는
E(label_i)>E(label_j) and H(label_i)<H(label_j)

예를 들어, 그림 4에서 레이블 '111,123'와 레이블 '112.122'을 위의 조건을 비교하여 판단하면 사전적 순서로 '111'<'112' 그리고 '123'>'122'하므로 병행한 사건으로 판단한다. 그러나 스레드 '11,12'와 스레드 '111,123'의 경우에는 위의 두 조건 중에 어느 것도 만족하지 못하므로 순서화 되어 있다고 판단한다. 이와 같은 방법으로 다중스레드 프로그램에서 공유메모리에 접근하는 사건들의 정보로 경합, 교착상태 등을 탐지하는 자료로 이용할 수 있다.

### 3.2 Task Recycling

Task Recycling (TR)에서의 레이블은 각 스레드에 대한 유일한 스레드 식별자 (thread identifier)로 구성된다. 스레드 식별자는 스레드 번호 (thread number: t)와 버전 번호 (version number: v)로 구성되어 있다. 스레드 번호는 재사용될 수 있다. 즉, 하나 이상의 순차적 스레드에 대해서는 동일한 스레드 번호를 할당할 수 있다는 것이다. 그리고 병행한 스레드에 대해서는 항상 다른 스레드 번호를 할당한다. 스레드 식별자에서 버전 번호는 동일 스레드 번호를 갖는 스레드들을 구분하기 위해 사용된다. 순차적 스레드에 대해 스레드 번호 't'가 할당될 때 마다 버전 번호 'v'는 '1'이 증가한다.

이러한 사건정보는 병행 수행되는 스레드을 위한 부모 벡터 (parent vector)를 유지한다. 부모 벡터에서 유지되는 요소의 수는 병행한 스레드의 수와 동일하다. 임의의 스레드 'T'에 대한 부모 벡터의 't'번째 요소는 스레드 번호 't'를 갖는 스레드 'T'의 조상 스레드에 할당된 버전 번호 중 최대값을 포함한다.

임의의 스레드 'T'가 다음과 같은 조건을 만족하면 스레드 식별자 'tv'를 가지는 스레드와 병행하다.

parent_T[t] < v

따라서 두 스레드간의 병행성 검사는 상수적인 비용(배열 참조) 만을 요구한다. EH에서와 같이 병행하게 수행되는 스레드에 대해서만 부모 벡터를 필요로 하며, 두 스레드간의 병행성 결정은 스레드의 식별자만 으로도 가능하다. 그러나 부모 벡터는 병행 수행하는 모든 스레드들을 위한 요소를 모두 가지고 있어야 하는 부담을 가지고 있다. 또한 생성되는 스레드의 식별자를 부여하기 위해 공유자료 구조를 이용해야 하기 때문에 다중스레드 프로그램의 병행적 수행에서 스레드 번호의 순차적 할당에 의해 정보 생성시 수행 시간의 증가를 보이게 된다.

그림 5는 스레드 식별자의 할당 예를 보여주고 있으며 각 스레드 식별자 아래에 위치한 값은 해당 스레드의 부모 벡터의 구성 값을 의미한다. 그림 5에서 알 수 있듯이 스레드 '1,3'은 스레드 '2,1'과 병행하다. 왜냐하면, 스레드 '1,3'의 부모 벡터에 있는 두 번째 요소가 '1'보다 작기 때문이다. 반재의 경우에도 스레드 '2,1'의 첫 번째 요소에 있는 값이 '3'보다 작기 때문에 병행하다고 판단한다. 그러나 스레드 '1,2'와 '3,1'의 경우를 보면, 스레드 '3,1'의 부모 벡터의 첫 번째 요소가 '1'보다 크기 때문에 순서화 되어 있다고 판단한다.

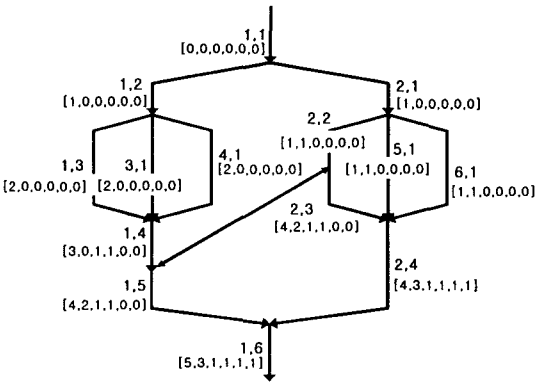



그림 5. Task Recycling

### 3.3 Offset-Span Labeling

Offset-Span (OS) 레이블링은 동기화가 없는 다중스레드 프로그램을 대상으로 하여, EH 레이블의 개념을 기반으로 레이블링에 필요한 기억 공간의 감소를 목적으로 한다. OS 레이블에서 레이블의 길이는 스레드에 관련되는 생성과 종료 사건의 쌍이 나타나는 내포 수준에 비례한다. EH의 경우에는 스레드의 생성에 관련된 경로상에 나타난 모든 생성 사건의 수에 비례한다. 임의의 한 스레드에서 OS 레이블은 일련의 레이블 열로 존재한다. 레이블 열은 스레드의 생성에서는 하나의 레이블이 증가되고, 종료 또는 합류 사건에서는 하나가 삭제된다.

하나의 OS 레이블은, 동일 부모 스레드에서 생성된 스레드들을 구분하기 위해 Offset과 동일 생성 사건에 의해 생성된 스레드의 수를 표시하는 Span으로 구성되며, [o, s]로 표시한다. 따라서, 하나의 레이블 길이는 상수 크기만을 필요로 한다. 임의의 스레드에 대한 OS 레이블은 부모 스레드의 레이블을 기반으로 구성한다. 임의의 스레드 'T'라 하고, 부모 스레드를 'P'라 하면 OS 레이블의 구성은 부모 스레드의 레이블 값에 부모가 생성한 총 스레드의 수를 's'라 하고 그 중에 해당되는 각 스레드들을 왼쪽에서 오른쪽으로 내림차순

으로 'o'의 값을 할당한다. 단, 할당되는 값은 '0'에서부터 시작된다. 그리고 스레드 합류시에는 합류 이전 스레드의 부모 스레드의 레이블 값 중에 가장 오른쪽의 [o, s] 레이블 값을 [o+s, s]와 같이 구성하여 합류 사건이후의 스레드에게 할당한다. 이러한 과정에 의해, OS 레이블링에서 마지막 스레드의 OS 레이블은 항상 [1,1]을 갖는다. 그림 6에서 위에서 언급한 방법으로 생성된 OS 레이블의 예를 보인다.

OS 레이블링에서는, 생성 스레드에서 부모 스레드의 레이블에 자신의 레이블 값을 첨가시키기 때문에 레이블 길이는 내포 수준에 비례하는 길이를 유지해야 한다. OS 레이블에서는 Offset 의 값은 왼쪽에서 오른쪽으로 사전적인 순서를 가지고 있기 때문에 임의의 두 스레드 간의 병행성은 레이블에서 동일 내포 수준의 레이블 요소에 대한 검사에 의해 결정된다. 동일 내포 수준의 OS 레이블 [o_i, s_i]를 갖는 스레드 'Ti'와 [o_j, s_j]를 갖는 스레드 'Tj'에 대해 다음 식을 만족하면, 두 스레드는 병행하다.

$$o\_i \bmod s\_i \neq o\_j \bmod s\_j$$

예를 들어, 그림 6에서 [0,1][1,2]와 [0,1][2,2]의 경우에 '1 mod 2 ≠ 2 mod 2'를 만족하므로 두 스레드는 병행하다. 그러나 [0,1][0,2][0,3]과 [0,1][2,2]의 경우에는 '0 mod 2 = 2 mod 2'이기 때문에, 순서화된 스레드로 결정한다.

### 3.4 Nest-Region Labeling

동기화 없는 내포 다중스레드 프로그램을 대상으로 하는 Nest-Region (NR) 레이블링에서, 각 스레드는 이전에 발생한 스레드의 일련의 루푸 지수열인 지수 역사 (index history)에 의해 스레드의 종류와 수를 인식할 수 있다. 지수 역사는 각 스레드에서 지수 역사의 마지막 요소에 첨가

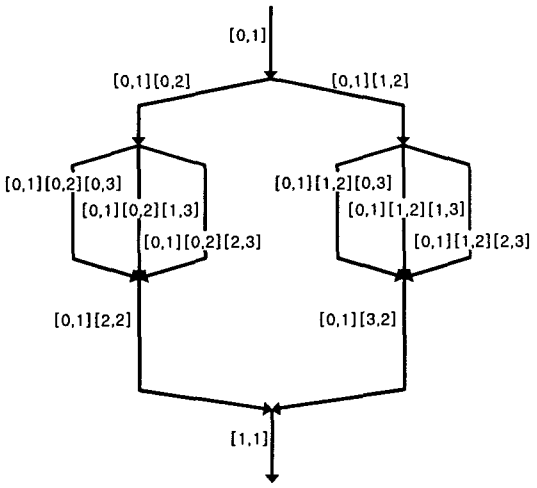



그림 6. Offset-Span Labeling

또는 삭제에 의해 구성되는 것으로, 수직적, 수평적 특성에 의해 두 개의 서수에 의해서 지수 역사의 사전적인 순서를 표시할 수 있다. NR 레이블링은 이 특성에 기반하여 내포 구역 (nest region)을 정의한다. 한 프로그램의 수행에서의 내포 구역은 프로그램의 최대 병행성을 의미한다.

NR 레이블링에서의 레이블 구조는 TID & OH를 구성하는 두가지 요소이다. 여기서 TID는 스레드 식별자이고, OH는 TID의 순서화된 리스트이다. TID는 [a,[b,c]로 표시되는 세 개의 정수로 구성된다. TID에서 '<b,c>'은 내포구역이라 하며, 'a'는 합류계수(join counter)라 한다. 스레드의 내포구역의 임의의 수행에서 나타나는 모든 스레드들에 대한 수 공간 (number space)에서의 간격을 의미한다. 두 스레드들의 논리적인 병행성을 결정하기 위한 비교에서 두 스레드의 내포구역의 중복 여부를 검사하는 과정을 수행한다. 임의의 두 스레드 Ti와 Tj에 대해, 내포구역의 중복은 다음 식에 의해 결정된다.

$$NR(Ti) \triangle NR(Tj) \equiv (bi \leq cj \wedge bj \leq ci)$$
$$NR(Ti) \diamondsuit NR(Tj) \equiv (ci < bj \vee cj < bi)$$

그림 7에서는 보이고 있지 않지만, 내포 구역만의 중복에 대한 검사는 동일 수준에서 하나의 연속적인 루프을 가지는 다중 방향 내포[3] 다중스레드 프로그램의 경우에는 적용하지 못한다. 따라서 이 경우의 검사과정은 두 스레드의 합류계수의 비교를 포함한다. 스레드의 합류계수는 프로그램의 최초 스레드에서 임의의 스레드까지의 순차적 경로에서 발생된 합류 사건에 '1'을 더한 값이다. 왜냐하면, NR 레이블링은 형식적인 일관성을 위해 다중스레드 프로그램을 하나의 병행 루프로 간주하여 프로그램의 최초 스레드를 합류 스레드로 가정하기 때문이다.

병행성 결정과정에서 두 스레드의 합류계수가 다르면, 현재 스레드의 OH에 있는 TID 중 한 항목을 이전 스레드의 TID와 비교 되어진다. OH는 합류계수 값에 의해 순서화된 TID의 리스트이다. NR 레이블에 의한 병행성 검사는 다음 과정에 의해 수행된다.

1. 두 스레드의 내포 구역비교,
   - 중복이면 순서화된 스레드로 결정
   - 중복이 아니면, 합류 계수를 비교
2. 동일한 합류 계수이면, 병행 스레드로 결정
3. 다른 합류 계수이면, 현재 스레드의 OH에 있는 스레드와 비교
   - OH에서 비교 스레드의 합류 계수보다 큰 합류 계수중 가장 작은 합류 계수를 가지는 스레드와 비교
   - 중복이면, 순서화된 스레드로 결정
   - 중복이 아니면, 병행 스레드로 결정

예를 들어, 그림 7에서 최초의 스레드의 내포 구역이 <1,50>이므로 이후 모든 생성된 스레드의 내포 구역의 값과 중복되므로 순서화된 스레드로 결정된다. 위의 검사과정에 따라 그림 7에서 보이는 레이블 값을 검사하면, 임의의 두 스레드들 간의 논리적 순서관계를 판단할 수 있다.

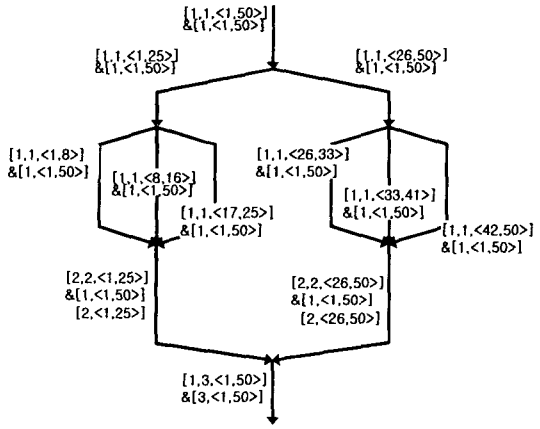



그림 7. Nest-Region Labeling

본 장에서 소개한 사건정보 생성기법은 레이블 생성을 위해 사용되는 구조체의 사용 형태에 따라 두 가지로 분류할 수 있다. 이러한 분류는 구조체의 공유 여부에 의한 것이다. NR, OS, EH 는 공유하는 구조체가 없는 생성기법으로, 새로운 스레드의 레이블을 생성할 때에는 그 스레드를 생성시킨 스레드의 레이블의 값에 의해서만 결정된다. 그러나 TR은 병행한 스레드 간의 논리적 순서를 결정하기 위해 부모 벡터를 스레드들이 공유한다. 이러한 구조체의 공유는 병행 수행하는 스레드들의 수행을 순서화 시키는 상태를 유발할 수 있다.

## 4. 결 론

다중스레드 프로그램의 비결정적인 수행 특성으로 인하여 발생할 수 있는 오류인 경합, 교착상태 등을 탐지하기 위해서는 감시 대상이 되는 공유메모리에 접근하는 사건들 간의 순서관계 정보가 필요하다. 이러한 사건정보를 생성하는 기법들은 프로그램의 수행 중에 발생하는 스레드의 생성, 종료, 동기화 등 논리적 순서 관계에 영향을

줄 수 있는 사건들을 분석하여 그들 간의 논리적 순서관계를 표시한다. 이러한 기법들의 성능은 레이블 길이와 순서관계 결정에 필요한 비용에 의해 결정된다.

EH 레이블링 기법은 Java와 같은 스레드 수행 구조를 가지는 프로그램에서도 적용하여 경합, 교착상태 등을 탐지하는데 이용될 수 있지만, 웹 서비스 응용 프로그램과 같이 클라이언트가 요청한 서비스를 처리하기 위해 계속적으로 스레드을 생성되는 수행 모델에서는 사건정보를 생성하는데 비현실적인 시간, 공간 비용이 요구되므로, 이러한 문제점을 해결할 수 있는 다중스레드 Java 프로그램을 위한 효율적인 사건정보 생성기법의 개발이 필요할 것으로 생각된다.

## 참 고 문 헌


[1] Deitel H. M., *Operating Systems,* Second Edition, Addison-Wesley, 1994.

[2] 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, March 1990.

[3] Jun, Yong-Kee, and Kern Koh, "*On-the-fly Detection of Access Anomalies in Nested Parallel Loops,*" Proc. of the ACM/ONR Workshop on Parallel and Distributed Debugging, pp.107-117, ACM, San Diego, California, May 1993.

[4] Lea, Doug, *Concurrent Programming in Java,* Addison-Wesley, 1997.

[5] 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.

[6] Netzer R. H. B., and B. P. Miller, "*What Are Race Conditions? Some Issues and Formalizations,*" ACM Letters on Programming Language and Systems, 1(1): 74-88, March 1992.

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

[8] Oaks, Scott, and Henry Wong, *Java Threads,* Second Edition, O'Reilly, 1999.

[9] Intel Co., Intel Thread Checker Relesase Notes, 2002.

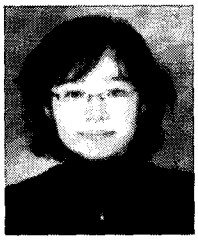
[10] KL Group, Getting Started with JProbe.

[11] OpenMP Architecture Review Board, OpenMP Fortran Application Program Interface, Version 2.0, Nov. 2000.



이 승 렬

• 2000년 경상대학교 교육대학원 석사 졸업
• 2001년~현재 경상대학교 컴퓨터과학과 박사과정
• 관심분야 : 운영체제, 다중스레드 프로그램 디버깅
• E-mail : lsy@gsnu.ac.kr



강 문 혜

- 2001년 경상대학교 컴퓨터과학과 학사졸업
- 2003년 경상대학교 컴퓨터과학과 석사졸업
- 2003년~현재 경상대학교 컴퓨터과학과 박사과정
- 관심분야 : 운영체제, 병렬프로그램 디버깅
- E-mail : turtle@race.gsnu.ac.kr

김 동 국

- 1997년 경상대학교 전산학과 공학박사졸업
- 1999년 한국전자통신연구원(ETRI)
- 2000년 (주)대신정보통신 선임연구원
- 2000년~현재 (주)애스톤리눅스 대표이사
- 관심분야 : 운영체제, 병렬 프로그램 디버깅,
- E-mail : dgkim@astonlinux.com