OpenMP 프로그램을 위한 경합탐지 도구의 분석

김영주,° 강문혜, 전용기
경상대학교 컴퓨터과학과
{yjkim,turtle,jun}@race.gsnu.ac.kr

An Analysis of Race Detection Tool for OpenMP Programs

Young-Joo Kim,° Moon-Hye Kang, and Yong-Kee Jun
Dept. of Computer Science, Gyeongsang National University

요 약

공유메모리 기반의 OpenMP 프로그램에서 발생하는 경합은 의도하지 않은 비결정적 수행 결과를 초래하므로 효과적으로 경합을 탐지하는 도구가 필요하다. 본 연구는 OpenMP 프로그램의 경합탐지를 위한 Intel 사의 Thread Checker를 내포병렬성의 여부와 접근사건들의 분포 형태를 기준으로 개발한 커널프로그램 집합을 이용하여 분석한 결과로서, 스레드들을 순서적으로 수행하면서 내포된 스레드를 부모 스레드와 동일한 스레드로 간주하고 적어도 하나의 읽기와 쓰기 접근사건들을 유지하면서 수행중에 경합을 탐지하는 도구임을 보인다. 이 도구는 접근사건의 발생 시에 이전 접근사건들과의 경합 여부를 검사한 후에 그 접근사건의 유지 여부를 결정하므로, 논리적 병행성 관계를 반영하지 못하는 내포된 스레드가 존재하지 않으면 경합의 존재를 검증한다.

1. 서 론

공유메모리를 기반으로 하는 산업표준 프로그램 모델인 OpenMP[8] 프로그램에서 병행하게 수행되는 스레드들이 공유변수에 대해 적절한 동기화 없이 적어도 하나 이상의 쓰기 사건으로 접근할 때 발생되는 오류를 경합[7]이라 한다. 이러한 경합은 동일한 입력에 대하여 프로그램의 매 수행 시에 의도하지 않은 비결정적 결과를 초래하므로 효과적으로 경합을 탐지하는 도구가 필요하다.

본 연구는 OpenMP 프로그램의 경합탐지를 위한 Intel 사의 Thread Checker[2, 3, 4, 9]를 내포병렬성의 여부와 접근사건들의 분포 형태를 기준으로 개발한 커널프로그램 집합을 이용하여 분석한 결과로서, 스레드들을 순서적으로 수행하면서 내포된 스레드를 부모 스레드와 동일한 스레드로 간주하고 적어도 하나의 읽기와 쓰기 접근사건들을 유지하면서 수행중에 경합을 탐지하는 도구임을 보인다. Intel Thread Checker는 OpenMP 프로그램에서 경합을 탐지하는 KAI Assure[5]를 재 설계하여 동적 분석 기법을 적용한 도구이다. 동적 분석 기법[9]은 프로그램에 입력되는 데이터에 의존적이고 프로그램의 수행과 분석을 동시에 하여 프로그램의 수행중에 발생할 수 있는 경합을 탐지하는 기법으로써 프로그램의 크기가 크고 구조가 복잡할 때 유용하다.

이러한 프로그램은 수행시간이 길고 많은 양의 메모리를 요구하며, 분석목적에 부합하는 프로그램을 선정하기 어려우므로 분석하고자 하는 도구의 입력 데이터로는 적합하지 않다. 따라서 도구분석을 위해 고려해야 할 사항들을 포함한 커널프로그램 집합으로 경합탐지 도구인 Intel Thread Checker를 분석한다. 커널프로그램은 내포병렬성의 여부와 읽기/쓰기 접근사건들의 발생순서를 조합하여 작성한다. 이렇게 해서 분석된 결과는 접근사건의 발생 시에 이전 접근사건들과의 경합 여부를 검사한 후에 그 접근사건의 유지 여부를 결정하므로, 논리적 병행성 관계를 반영하지 못하는 내포된 스레드가 존재하지 않으면 경합의 존재를 검증한다.

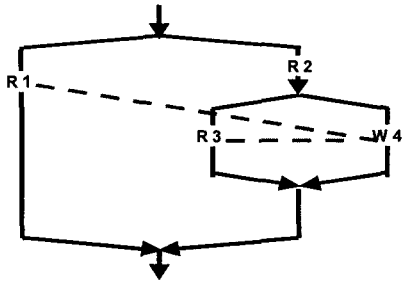
본 도구는 두 가지 기준으로 분석한다. 첫 번째 기준은 프로그램의 수행중 경합탐지 여부이고, 두 번째 기준은 내포병렬성을 가진 프로그램에서 경합검증 여부이다. 프로그램의 수행중 경합탐지 여부는 입력되는 데이터에 따라 경합으로 보고되는 접근사건들의 쌍을 분석하고, 내포병렬성을 가진 프로그램에서 경합검증 여부는 커널프로그램의 집합을 이용하여 분석한다.

2절에서는 OpenMP 프로그램에서의 경합을 소개하고, 3절에서는 경합탐지 도구인 Intel Thread Checker의 수행환경과 실험적으로 분석한 것을 소개한다. 4절에서는 결론 및 향후과제를 보인다.

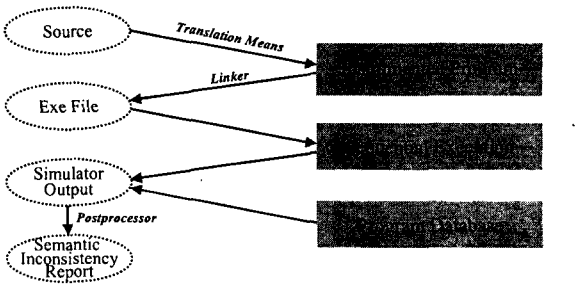
2. OpenMP 프로그램의 경합

OpenMP[8]는 공유메모리 병렬프로그램을 위한 산업표준 프로그램 모델로서, 표준 C/C++와 Fortran 77/90을 확장하는 지시어(directives)와 라이브러리들의 집합이다. OpenMP에서 제공하는 구조로는 병렬화 구조, 작업공유 구조, 데이터 환경 구조, 동기화 구조 등이 있다. 병렬화 지시어는 'PARALLEL'이 있고, 작업공유 지시어는 'DO'와 'SECTION'이 있고, 데이터환경을 위한 지시어는 데이터의 접근범위를 지정하는 'PRIVATE'과 'SHARED'가 있으며, 동기화를 위한 지시어는 'CRITICAL'과 'BARRIER'가 있다.

(그림 1)은 방향성이 있는 비순환 그래프인 POEG[1]으로 OpenMP 병렬프로그램의 수행을 나타낸 것이다. 여기서 정점(vertex)은 병렬스레드의 생성/합류의 스레드 명령을 나타내고, 정점들 사이의 간선(arc)은 스레드 명령에 의해 수행되는 논리적 스레드들이다. 이런 스레드에 발생하는 R와 W는 읽기와 쓰기 접근사건을 의미하고, 접근사건에 있는 숫자는 임의의 발생순서를 의미한다. POEG는 병렬프로그램의 수행 시에 나타나는 스레드들간의 부분적인 순서를 나타내므로, 병렬스레드의 발생후(happened before) 관계[6]를 알 수 있다. 두 스레드들 사이에서 발생후 관계를 만족하면, 두 스레드들간에



<그림 1> POEG에서의 경합



<그림 2> Intel Thread Checker 실행 환경

경로가 존재하며 그 스레드들은 순서적 (ordered) 이라고 하고, 그렇지 않으면 병행 (concurrent)하다고 한다. 경합은 병행하는 스레드들 사이에서 하나의 공유변수에 적어도 하나의 쓰기 접근사건을 가지는 접근사건의 쌍에 의해 발생한다. <그림 1>에서 점선은 경합을 의미하고, 경합의 표시는 R1-W4와 R3-W4로 나타낸다.

3. 경합탐지 도구

본 절에서는 OpenMP 프로그램의 경합탐지를 위한 Intel 사의 Thread Checker를 내포병렬성의 여부와 접근사건들의 분포 형태를 기준으로 개발한 커널프로그램 집합을 이용하여 분석한 결과로서, 스레드들을 순서적으로 수행하면서 내포된 스레드를 부모 스레드와 동일한 스레드로 간주하고 적어도 하나의 읽기와 쓰기 접근사건들을 유지하면서 수행중에 경합을 탐지하는 도구임을 보인다. 이 도구를 분석하기 이전에 도구의 수행환경에 대해 살펴보고, 커널프로그램 집합으로 도구의 경합탐지 원리에 대해 분석한다.

3.1 수행 환경

Intel Thread Checker의 수행환경을 시스템 환경과 실행환경으로 나누어 살펴본다. 먼저, 시스템 환경은 인텔 펜티움-IV 컴퓨터에서 윈도우 2000을 기반으로 하여 OpenMP API 2.0[8]을 지원하고, 감시 및 성능 분석을 위한 프로그램으로 변형할 수 있는 Intel C++/Fortran Compiler[2,4]와 프로그램에서 스레드들의 병행적 수행을 순차적으로 수행하면서 분석하여 경합을 탐지하는 도구인 Intel Thread Checker[2, 3, 4] 및 OpenMP 프로그램의 성능을 분석하는 도구인 Thread Profiler[2, 3]를 포함하고 있는 Intel VTune Performance Analyzer[3, 4]로 구성되어 있다.

다음으로, 실행 환경은 <그림 2>와 같다. 대상으로 하는 프로그램인 OpenMP 프로그램을 Intel C++/Fortran Complier가 경합을 탐지하기 위한 프로그램으로 변형한다. 변형된 프로그램의 스레드들을 Intel Thread Checker가 순서적으로 수행시키면서 프로그램의 변형 시에 경합에 필요한 정보들을 저장한 Program DataBase를 참조하여 경합을 탐지하고 그 결과를 Intel VTune Performenance Analyzer가 사용자에게 <그림 4>처럼 보고한다.

3.2 실험분석

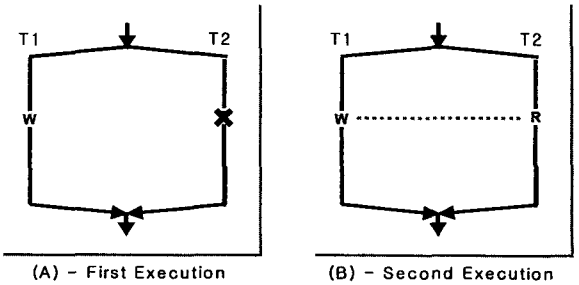
커널프로그램의 집합을 이용하여 Intel Thread Checker가 프로그램의 수행중에 경합을 탐지하는 기법임을 보이고, 커널프로그램에서 탐지되는 경합의 유형을 분석하여 경합탐지 기법의 원리를 살펴본다.

3.2.1 수행중 경합탐지

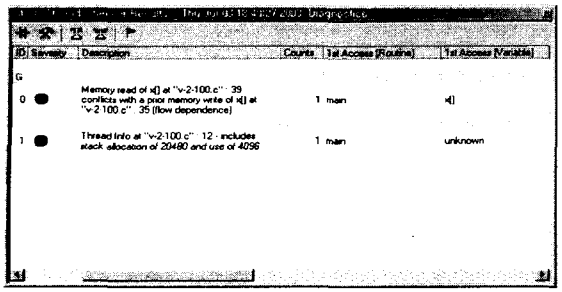
<그림 3>은 Intel Thread Checker가 수행중에 경합을 탐지하는 도구임을 보인다. <그림 3>의 (A)와 (B)는 동일한 OpenMP 프로그램에서 입력되는 데이터 값에 따라서 스레드들에 발생하는 접근사건들의 양상이 달라짐을 보인다. <그림 3>-(A)는 T2 스레드에서 입력되는 데이터에 의해서 읽기 접근사건이 발생하지 않으므로 T1 스레드에서 발생한 쓰기 접근사건과 경합을 이루지 않지만, <그림 3>-(B)는 T2 스레드에서 읽기 접근사건이 발생하므로 T1 스레드에서 발생한 쓰기 접근사건과 경합을 이룬다. 이처럼 입력되는 데이터에 의해 발생하지 않은 접근사건들에 대해 경합 여부를 검사하지 않고 수행중에 발생하는 매 접근사건과 이전 접근사건들과의 논리적 병행성 여부를 판단하여 경합 여부를 결정하므로 Intel Thread Checker는 수행중 경합탐지 도구이다. <그림 4>는 <그림 3>-(B)의 경우에 해당하는 OpenMP 프로그램에서 공유변수 X에 대해서 발생한 경합을 보고한 것이다. Intel VTune Performance Analyzer에 의해 보고된 두 개의 리스트 중에서 첫 번째 리스트에 있는 것이 것이 경합이고 두 번째 리스트는 프로그램에 대한 설명이다.

3.2.2 경합탐지 기법

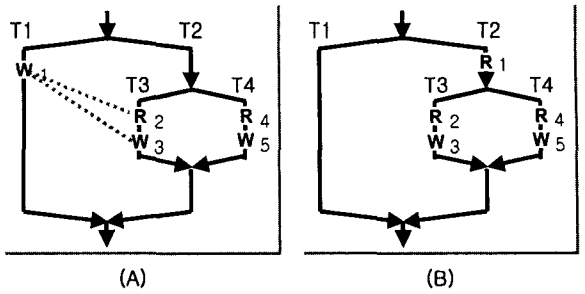
수행중 경합탐지 도구인 Intel Thread Checker의 경합탐지



<그림 3> Intel Thread Checker의 수행중 경합탐지



<그림 4> VTune Performance Analyzer에서 경합보고

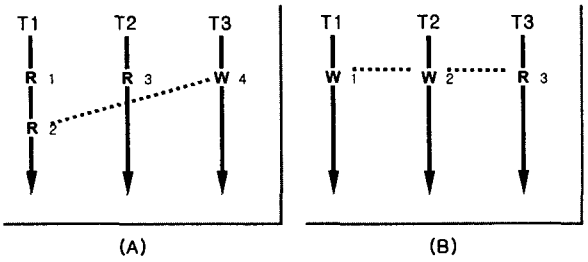


<그림 6> 내포병렬성이 존재하는 모델에서 경합탐지

기법의 원리를 분석하기 위해서 커널프로그램의 집합을 이용한다. 커널프로그램은 내포병렬성이 존재하지 않은 프로그램 모델과 내포병렬성이 존재하는 프로그램 모델로 분류하여 각 스레드에 접근사건들의 순서를 임의로 발생시켜 경합이 탐지되는 양상을 분석하는데 이용한다. <그림 5>와 <그림 6>은 커널프로그램에서 탐지된 경합들을 POEG으로 나타낸 것이다.

<그림 5>는 내포병렬성이 존재하지 않은 프로그램 모델에서 프로그램의 수행중에 탐지된 경합들을 보인 것이다. <그림 5>-(A)에서 T1 스레드의 R2와 T3 스레드의 W4만이 경합이다. 왜냐하면, 이 도구는 R2 발생시에 이전 접근사건인 R1과 순서화 관계에 있으므로 R1을 삭제하고, R3 발생시에 이전 접근사건인 R2와 병행성 관계에 있으므로 R3을 삭제하기 때문이다. 따라서 W4 발생시에 이전 접근사건인 R2와 논리적 병행성 여부를 판단하여 경합으로 보고된다. 그리고 <그림 5>-(B)에서 W1-R3은 경합으로 보고되지 않고, W1-W2와 W2-R3이 경합으로 보고된다. 왜냐하면, W2 발생시에 이전 접근사건인 W1과 논리적 병행성 여부를 판단하여 W1와 W2를 경합으로 보고한 후에, 접근사건 유지정책에 의해 W2는 W1과 논리적 병행성 관계이므로 W1이 삭제되기 때문이다. 따라서 R3 발생시에 W1은 W2에 의해 삭제되어 W2와 R3이 경합으로 보고된다.

<그림 6>은 내포병렬성을 가진 프로그램 모델에서 프로그램의 수행중에 탐지된 경합을 보인 것이다. 경합을 탐지하는 원리는 내포병렬성이 존재하지 않은 프로그램에서와 동일하지만 내포된 스레드에 존재하는 접근사건들에 대해서 논리적 병행성 관계를 여부를 판별하지 못한다. 왜냐하면, Intel Thread Checker가 내포된 스레드들을 부모 스레드와 동일한 스레드로 인식하기 때문이다. 예를 들어, <그림 6>-(A)에서 R2-W5를 경합으로 보고하지 못하는 이유는 T3와 T4는 병행한 스레드이지만 Intel Thread Checker가 동일한 스레드로 인식하므로 두 스레드는 순서적 관계에 있다고 판별되기 때문이다. 이런 이유로 내포병렬성을 가진 프로그램 모델에서 경합이 존재하더라도 <그림 6>-(B)처럼 경합을 탐지 못할 수도 있다.



<그림 5> 내포병렬성이 존재하지 않는 모델에서 경합탐지

4. 결론

본 논문의 Intel Thread Checker의 실험적 분석 결과를 프로그램 모델과 경합탐지 기법 측면에서 정리하면 다음과 같다. 첫 번째, Intel Thread Checker는 내포병렬성을 가진 프로그램 모델이 존재하지 않으면 경합 존재 여부를 검증할 수 있다. 두 번째, 접근사건이 발생하면 이전 접근사건들과 경합 여부를 검사한 후에 병행한 스레드들에서 적어도 하나의 읽기와 쓰기 접근사건들을 유지한다. 앞으로 Intel Thread Checker는 OpenMP 프로그램의 작성시에 프로그래머에게 유용한 도구가 될 것이다.

[참고문헌]

[1] Dinning, A., and E. Schonberg, "An Empirical Comparison of Monitoring Algorithms for Access Anomaly Detection," 2nd Symp. on Principles and Practice of Parallel Programming (PPoPP), pp. 1-10, ACM, March 1990.

[2] Intel Co., Getting Started with the Intel Thread Checker and Thread Profiler, 2003

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

[4] Intel Co., Threading Methodology: Principle and Practices, Version 2.0, 2003

[5] Kuck & Associates, Inc., Assure User's Manual, Version 4.0, 2001.

[6] Lamport, L., "Time, Clocks, and the Ordering of Events in a Distributed System," Comm. of the ACM, 21(7): 558-565, ACM, July 1978.

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

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

[9] Petersen, P., and S. Shah, "OpenMP Support in the Intel Thread Checker," WOMPAT 2003, pp. 1-12, June 2003