OpenMP 디렉티브 프로그램의 최초경합 탐지를 위한 도구

(A Detection Tool of First Races in OpenMP Programs with Directives)

강 문 혜 † 하 옥 균 †† 전 용 기 †††
(Mun-Hye Kang) (Ok-Kyoon Ha) (Yong-Kee Jun)

요 약 OpenMP 디렉티브 프로그램의 디버깅을 위해서 비결정적인 수행결과를 초래하는 경합을 탐지하는 것은 중요하다. 특히, 프로그램 수행에서 가장 먼저 발생하는 최초경합은 이후에 발생하는 경합에 영향을 줄 수 있으므로 효과적인 디버깅을 위해서 반드시 탐지되어야 한다. 그러나 기존의 경합탐지 도구들은 최초경합의 탐지를 보장하지 못한다. 본 논문에서는 내포병렬성을 포함한 프로그램을 두 번의 수행으로만 프로그램의 수행 중에 최초경합을 탐지하는 도구를 제시한다. 본 도구의 정당성을 합성프로그램을 이용하여 보이고, 기존 경합탐지 도구와 기능성을 비교한다.

키워드 : OpenMP 프로그램, 경합, 최초경합, 디버깅 툴

Abstract Detecting data races is important for debugging programs with OpenMP directives, because races result in unintended non-deterministic executions of the program. It is especially important to detect the first data races to occur for effective debugging, because the removal of such races may make other affected races disappear or appear. The previous tools for race detecting can not guarantee that detected races are the first races to occur. This paper suggests a tool what detects the first races to occur on the program with nested parallelism using the two-pass on-the-fly technique. To show functionality of this tool, we empirically compare with the previous tools using a set of the synthetic programs with OpenMP directives.

Key words : openMP program with directives, races, first races to occur, debugging tool

1. 서 론

공유메모리를 기반으로 하는 OpenMP[1,2] 프로그램에서 발생할 수 있는 심각한 오류중의 하나인 경합(race)[3-5]은 병행적으로 수행되는 스레드들이 하나의 공유변수에 대해 적절한 동기화 없이 적어도 하나 이상의 쓰기 사건으로 접근할 때 나타난다. 이러한 경합은 프로그램 상에 의도하지 않은 비결정적(non-deterministic)인 수행결과를 초래하므로 디버깅을 위해서 반드시 탐지되어야 한다. 특히, 경합 중에서 가장 먼저 발생하고, 다른 경합으로부터 영향을 받지 않은 최초경합(first race to occur)[6-9]을 탐지하여 제거하는 것은 최초경합으로부터 영향받은 다른 경합들이 사라질 수 있거나 새로이 발생할 수 있기 때문에 효과적인 디버깅을 위해서 중요하다.

OpenMP 디렉티브 프로그램에서의 경합을 탐지하기 위한 기존의 도구에는 Intel Thread Checker[10, 11]와 Sun Thread Analyzer[11-13]가 있다. 이들 도구는 최초경합의 탐지를 보장하지 못한다. Thread Checker는 병행하는 스레드들을 순서적으로 수행하여 데이터의 의

· 본 연구는 지식경제부 및 정보통신연구진흥원의 대학 IT연구센터 지원사업의 연구결과로 수행되었음
· 본 연구는 한국 학술진흥재단의 국제공동연구-협력기관 지정 사업으로 수행되었음

† 학생회원 : 경상대학교 컴퓨터과학과
kturtle@hanmail.net
†† 학생회원 : 경상대학교 정보과학과
hannuri2000@hanmail.net
††† 종신회원 : 경상대학교 정보과학과 교수
jun@gnu.ac.kr

논문접수 : 2009년 4월 24일
심사완료 : 2009년 9월 8일

정보과학회논문지: 시스템 및 이론 제37권 제1호(2010.2)

존성 여부를 검사한 후에 경합을 탐지하므로 내포병렬성이 존재하는 프로그램 모델에서 내포된 스레드를 부모 스레드와 순서적인 스레드로 간주한다. 따라서 Thread Checker는 최초경합을 포함한 경합의 존재를 검증하지 못한다. Thread Analyzer[11-13]는 소스코드에 경합탐지를 위한 추가코드를 삽입하여 프로그램 수행 중에 공유변수 등의 정보를 수집하여 경합을 탐지한다. 그러나 Thread Analyzer는 OpenMP 프로그램에서의 경합탐지를 위한 기능 측면에서 분석된 적이 없으므로, 간단한 합성프로그램을 작성하여 실험해 본 결과 최초경합을 검증하지 못함을 알 수 있었다.

본 논문에서는 내포병렬성이 존재하는 OpenMP 프로그램에서 효과적으로 경합을 탐지하는 도구를 개발하기 위해서 수행 중 생성되는 병렬 스레드에 고유한 식별자를 부여하는 레이블링[14]기법과 효율적으로 최초경합만을 탐지하는 프로토콜[6]기법을 적용한다. NR labeling [14]은 프로그램 수행 중에 각 스레드의 논리적 병행정보를 생성하며, 지역 자료구조를 사용하여 병목현상이 발생하지 않는다. 효율적 최초경합 탐지기법[6]은 두 번의 프로그램 수행 중에 최초경합의 가능성이 있는 후보 사건들만을 감시하여 최초경합만을 탐지한다.

2절에서는 OpenMP 디렉티브 프로그램에서의 최초경합을 설명하고, 경합탐지 측면에서 기존 경합탐지 도구의 기능성과 문제점을 살펴본다. 3절에서는 본 연구에서 제안하는 효과적 경합탐지 도구의 설계와 구현에 대해서 설명한다. 그리고 4절에서는 합성프로그램을 이용하여 도구의 정확성과 효율성을 실험한 결과를 살펴보고, 마지막 5절에서 결론을 제시한다.

2. 연구배경

최초경합을 탐지하여 제거하는 것은 이 경합에 영향받은 다른 경합들이 사라질 수 있거나 새로이 발생할 수 있기 때문에 효과적인 디버깅을 위해서 매우 중요하다. 그러나 기존의 경합탐지 도구인 Thread Checker와 Thread Analyzer는 최초경합의 탐지를 보장하지 못한다. 이 절에서는 OpenMP 프로그램에서 발생하는 최초경합의 개념에 대해서 설명하고, 기존도구들의 기능과 문제점에 대해서 기술한다.

2.1 OpenMP 프로그램에서의 경합

OpenMP 프로그램[1,2]은 표준 C/C++과 Fortran 77/90을 기반으로 작성된 공유메모리 병렬 프로그램을 위한 산업표준 프로그램 모델이다. 이 프로그램 모델은 순차 프로그램을 병렬 프로그램으로 변환하는데 용이하게 하기 위해서 컴파일러 디렉티브를 제공한다. OpenMP에서 제공하는 컴파일러 디렉티브에는 병렬화 디렉티브, 작업공유 디렉티브, 데이터 환경 디렉티브, 동기화 디렉티브 등이 있다. 병렬화 디렉티브는 'PARALLEL'이 있고, 작업공유 디렉티브는 'FOR'와 'SECTION'이 있고, 데이터 환경을 위한 디렉티브는 데이터의 접근범위를 지정하는 'PRIVATE'과 'SHARED'가 있으며, 동기화를 위한 디렉티브는 'CRITICAL'과 'BARRIER' 등이 있다. 'PARALLEL FOR'는 병렬 스레드들의 생성(fork)을 의미하고, 'END PARALLEL FOR'는 병렬 스레드들의 합류(join)를 의미한다.

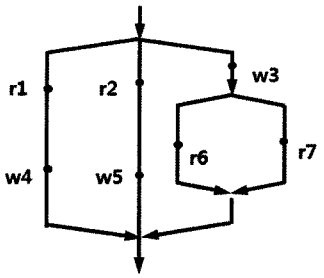

그림 1 Races in OpenMP program

병렬프로그램의 수행양상은 그림 1과 같이 방향성이 있는 비순환 그래프인 POEG(Partial Order Execution Graph)[15]으로 나타낼 수 있다. POEG에서의 정점(vertex)은 병렬 스레드의 생성과 합류를 나타내며, 정점들 사이의 간선(arc)은 정점에서 생성된 스레드를 나타낸다. 그리고 각 스레드에 접근하는 읽기와 쓰기 사건을 점으로 표시하고, r과 w로 나타내어 그 유형을 구분한다. 여기서 사용된 숫자는 특정수행 시에 접근사건들의 발생 순서를 의미한다. POEG은 병렬프로그램의 수행 시에 스레드들 간의 부분적 순서(partial order)관계를 나타낸다. 두 사건사이에 경로가 존재하면 선행(happened-before) 관계가 있고, 두 사건이 순서화(ordered)되었다고 한다. 그러나 경로가 존재하지 않으면 이 두 사건은 병행(concurrent)하다. 그리고 임의의 사건 e_i가 e_j보다 선행하고 $\{e_i, e_j\}$의 각 내포수준이 $\{m, n\}$일 때, $m < n$을 만족하면 e_j가 e_i에 내포(nested)된다고 한다. 그림 2.1에서 w3과 r7은 그들 사이에 경로가 존재하므로 w3은 r7에 선행하고, r7의 내포수준이 2로서 r3의 내포수준 1보다 크기 때문에 r7은 w3에 내포된다. 그러나 r1과 w3 사이에는 경로가 존재하지 않으므로 서로 병행하다.

이렇게 병행관계에 있는 두 개의 접근사건이 적어도 하나의 쓰기 사건을 포함하고 하나의 공유변수에 대해 적절한 동기화가 없이 나타날 때 발생하는 오류를 경합(race)[3-5]이라고 하며, e_i-e_j로 표시한다. 여기에서 임의의 사건 e_h가 e_i에 선행하고 e_h가 경합 R_h에 포함되면, e_i는 e_h나 R_h에 의해 영향받은(affected) 사건이라고 한다. 임의의 경합을 구성하는 두 사건 $\{e_i, e_j\}$중에서 어느

사건도 다른 사건들에 의해 영향받은 사건이 아니면, 이러한 경합을 영향받지 않은(unaffected) 경합이라고 하고, 두 사건들 중에서 하나만 다른 사건에 의해 영향받으면 부분적으로 영향받은(partially affected) 경합이라 한다. 이 때, 서로 부분적으로 영향받는 경합들이 두 개 이상인 경합집합을 얽힘(tangle)이라 하고, 얽힘에 속한 임의의 경합을 얽힌(tangled) 경합이라 한다. 또한 모든 접근사건들이 기껏해야 하나의 얽힌 경합에 영향받은 경합들의 집합인 얽힘을 최초얽힘(first tangle)이라 한다. 이러한 영향받지 않은 경합이거나 그렇지 않으면 최초얽힘을 최초경합(first race to occur)[6-9]이라 한다.

예를 들어, 그림 1에서의 경합은 {r1-w3, r2-w3, w3-w4, w3-w5, w4-w5, r2-w4, r1-w5, w4-r6, w4-r7, w5-r6, w5-r7}이다. 11개의 경합 중에서 단 두 개의 경합{r1-w3, r2-w3}만이 최초경합이며, 나머지 경합은 모두 영향받은 경합이다. 이러한 최초경합을 탐지하여 제거하는 것은 이 경합에 영향받은 다른 경합들이 사라질 수 있거나 새로이 발생할 수 있기 때문에 효과적인 디버깅을 위해서 매우 중요하다.

2.2 기존의 경합탐지 도구

OpenMP 디렉티브 프로그램에서의 경합을 탐지하기 위한 기존의 도구에는 Intel Thread Checker[10,11]와 Sun Thread Analyzer[11-13]가 있다. Intel Thread Checker는 Intel 사에서 병행스레드 프로그램에서 발생하는 경합과 데드락 같은 오류를 탐지하기 위해서 개발한 도구로 윈도우와 리눅스 버전으로 현재 3.1버전까지 개발되어 있다. 이 도구는 감시하고자 하는 소스코드에 감시코드를 삽입하여 병행하는 스레드들을 순서적으로 수행하고, 프로그램 수행 중에 데이터의 의존성 여부를 검사하여 경합을 탐지한다. 그러나 이 도구는 최초경합을 포함한 경합의 존재를 검증하지 못한다. 그림 1의 예제에서 Thread Checker를 사용하여 탐지된 경합은 {w4-r2, w4-w5, r1-w5, w5-w3, w5-r6, w5-r7} 등이다. 이는 경합{r1-w3, r2-w3}에 영향받은 경합이므로 최초경합이 아니다.

Sun Thread Analyzer[11-13]는 소스코드에 추가적인 감시코드를 삽입하여 프로그램 수행중에 성능데이터와 공유변수 등을 기록한다. 그러나 이 도구는 OpenMP 프로그램에서의 경합탐지를 위한 기능면에서 분석된 적이 없으며, Sun Microsystems사에서도 Thread Analyzer의 내부 동작원리를 공개한 적이 없다. 따라서 본 연구에서 OpenMP 프로그램으로 작성된 간단한 합성프로그램으로 경합탐지를 위한 도구의 기능을 실험해 본 결과, Thread Analyzer는 최초경합 탐지를 보장하지 못함을 알 수 있었다. 그림 1의 예제에서 탐지된 경합은 {w4-w3, w3-w5, w4-w5, w5-r7, w4-r7, w5-r6, w4-r6}이다. 이 또한 경합{r1-w3, r2-w3}에 영향받은 경합이므로 최초경합이 아니다.

3. 효과적 경합탐지 도구

본 도구는 두 번의 프로그램 수행으로 프로그램 수행 중에 최초경합에 참여할 가능성이 있는 후보사건을 감시하여 최초경합만을 탐지한다. 그림 2는 본 도구의 전체구조를 나타낸 것으로 클라이언트 서버 모델로 구성된다. 클라이언트는 selector, scanner, instrumentor, sender, 그리고 reporter 모듈로 구성되며, 서버에는 경합탐지를 위한 run-time 라이브러리들이 구현되어 있으며, 컴파일러가 포함되어 있다. 먼저, 클라이언트에서 디버깅하고자하는 OpenMP 병렬프로그램이 입력되면, selector 모듈은 사용자가 감시를 위한 공유변수와 경합탐지 프로토콜을 선택하도록 도와주며, 선택된 정보를 저장한다. 다음으로 scanner 모듈은 소스코드를 분석하고 분석된 정보를 이용하여 instrumentor 모듈이 선택된 경합탐지 엔진을 실제로 호출하기 위한 감시코드를 소스코드에 삽입한다. 감시코드가 삽입된 프로그램은 sender 모듈에 의해서 서버로 전송되며, 서버에서 프로그램을 실행시켜 탐지된 경합결과를 클라이언트에 넘긴다. 마지막으로 클라이언트의 reporter 모듈에 의해서 탐지된 경합정보를 사용자에게 출력해 준다.

최초경합 탐지를 위해서 삽입된 기법은 두 번의 프로그램 수행으로 프로그램 수행 중에 최초경합을 탐지한다. 내포병렬성을 지원하는 효율적 기법[6]은 첫 번째

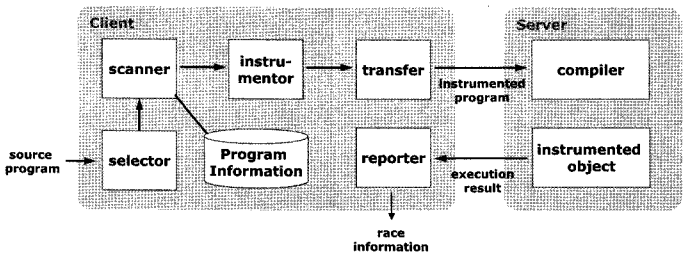

그림 2 도구모듈 설계

수행에서 접근역사 내의 접근사건들을 상수개로 유지하고, 접근역사 내의 사건들과 처음으로 경합을 구성하는 사건들을 최초경합을 발생시키는 후보사건으로 수집한다. 두 번째 수행에서는 각 접근 사건들을 감시하면서 첫 번째 수행에서 수집된 후보사건들과의 선행 관계(happened-before relation)[16]와 좌우관계(left-of relation)[17]를 검사하여 최초경합을 구성하는 후보사건들의 집합을 완성하여 최초경합으로 보고한다. 이 기법은 수행 중에 각 공유변수에 대한 접근역사를 상수적 크기로 유지하므로 각 접근사건의 수행 시에 상수적 복잡도의 사건비교 횟수와 기억공간만을 요구하지만 최악의 경우 하나의 읽기-쓰기 후보사건이 제외된다.

그림 3은 본 도구의 사용자 인터페이스를 보인다. 도구는 메뉴바, 도구바, 작업창, 그리고 로그 창으로 구성된다. 작업창은 디버깅하고자 하는 소스프로그램의 파일 목록을 보여주는 창①과, 소스프로그램을 보여주는 창②, 그리고 탐지된 경합의 결과를 보여주는 창③으로 구성된다. 도구바는 도구설계에 맞춰 경합을 탐지하기 위한 순서대로 나열하였으며, Mode Selector, Program Analyzer, Program Instrumentor, Race Detector 등의 항목으로 구성된다. Mode Selector는 그림 4와 같이 경합탐지 엔진과 감시하고자 하는 공유변수를 선택할 수 있는 창을 호출한다. Program Analyzer는 소스프로그램을 분석하며, 그 결과들은 로그창에서 확인할 수 있다. Program Instrumentor는 감시하고자 하는 공유변수를 선택하여 Mode Selector에서 선택된 엔진을 소스프로그램에 Instrument한다. Instrument된 소스프로그램은 그림 3의 작업창 ②에서 보여진다. Race Detector는 감시코드가 삽입된 소스프로그램을 서버로 보내어 엔진 라이브러리를 포함하여 컴파일한 실행파일을 생성한다. 마지막으로 Race Reporter에 의해서 탐지된 경합정보가 그림 3의 작업창 ③에 보여진다. 탐지된 경합정보는 소스상의 라인정보를 포함하여 디버깅하기 쉽도록 하였으며, Report Highlight에서는 탐지된 경합정보를 더블클릭하면 소스 상의 해당라인에 하이라이트가 되도록 하였다.

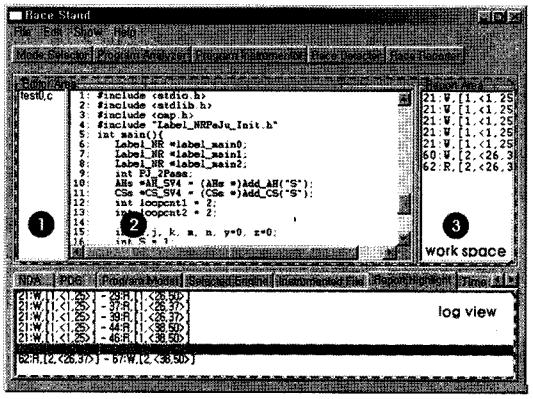

그림 3 도구의 인터페이스

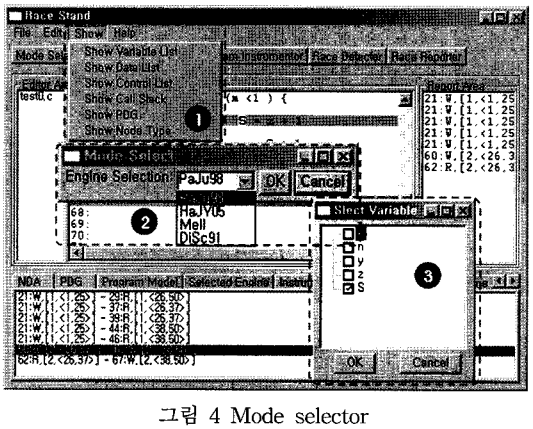

그림 4 Mode selector

4. 실 험

본 절에서는 도구의 구현환경을 설명하고, 합성프로그램의 집합을 이용하여 경합탐지의 정확성과 효율성을 분석한다. 실험 결과 제시된 도구는 최초경합만을 정확하게 탐지하였으며, 최초경합 탐지시간 또한 현실적임을 보인다.

클라이언트 프로그램은 Java Development Kit 1.6버전의 Java 언어로 구현되었으며 Eclipse환경에서 개발하였다. 서버는 리눅스 운영체제의 Intel Xeon 듀얼 CPU 컴퓨터에 OpenMP 병렬프로그램의 컴파일을 위해서 Intel C/C++ 10버전을 설치하였으며, 엔진 라이브러리는 C언어로 작성하였다. 이러한 모델은 엔진 라이브러리가 아닌 프로그램 실행 시에 필요한 라이브러리 등도 모두 서버에 설치되어 있어야 한다는 문제점이 남아있다. 클라이언트와 서버간의 파일을 전송하기 위해서 소켓프로그래밍이 구현되었으며, 디버깅하고자 하는 입력파일은 OpenMP C/C++ 프로그램만을 지원한다.

동일한 시스템 환경에서 기존 도구들의 최초경합 탐지여부를 실험하기 위해서 Intel Thread Checker는 리눅스 버전 3.1을 설치하였으며, Sun Thread Analyzer를 위해서 Sun Studio12를 설치하였다. Thread Checker는 Openmp 프로그램 소스 삽입을 위해서 컴파일 시에 -tcheck, -openmp 옵션을 사용하고, tcheck_cl 명령어를 이용해서 경합탐지 결과를 확인할 수 있다. Thread Analyzer는 컴파일 시에 OpenMP 프로그램을 지원하기 위해서 -xopenmp=noopt 옵션을 사용하고, 경합을 탐지하기 위해서 -xinstrument =datarace 옵션을 사용한다.

그리고 collect 명령어를 사용함으로써 collect tool과 함께 프로그램을 실행하며 경합을 탐지하고, 이때 생성된 tha.num.er파일을 er_print명령어를 통해서 경합탐지 결과를 확인할 수 있다.

본 도구의 정확성과 효율성을 실험하기 위해서 합성프로그램(synthetic programs)을 이용한다. 정확성을 위해서는 스레드 수, 내포병렬성 여부, 그리고 각 사건들의 배치를 다양하게 변형하여 합성프로그램을 작성하였으며, 효율성 실험을 위해서는 프로그램 최대병렬성과 각 스레드 당 이벤트 수를 달리하여 작성하였다. 이때, 최대병렬성은 5, 10, 20, 50, 100, 500으로 증가시키고 스레드 당 이벤트 수는 각각에 대해서 5개, 10개, 20개로 하였다. 프로그램명은 '최대병렬성_스레드 당 이벤트 수[_l 또는 s].c'로 하였다. 예를 들어, 10_50.c 프로그램은 최대병렬성이 10이고, 각 스레드 당 이벤트가 50개인 합성프로그램이다. 이때 각 스레드 당 발생하는 이벤트 타입은 모든 스레드의 처음 이벤트를 쓰기 사건으로 두고 나머지는 모두 읽기 사건을 발생시켰다. 파일명에 'l' 또는 's'가 붙은 프로그램은 하나의 스레드에서 쓰기사건이 단 하나 발생한 프로그램으로 발생시점에 따라서 l(last)과 s(start)로 나타내었다. 이는 적용된 최초경합 기법이 쓰기사건의 발생시점에 따라서 다른 성능을 보임을 실험으로 확인하기 위한 것이다.

표 1은 합성프로그램을 이용하여 최초경합탐지여부를 실험한 결과의 일부이다. 1번은 병렬스레드가 2개인 경우이고, 2-4번은 병렬스레드가 3개이며 내포병렬성이 존재하지 않는 합성프로그램이다. 5-6번은 병렬스레드가 3개이면서 내포병렬성이 2인 경우의 합성프로그램이다.

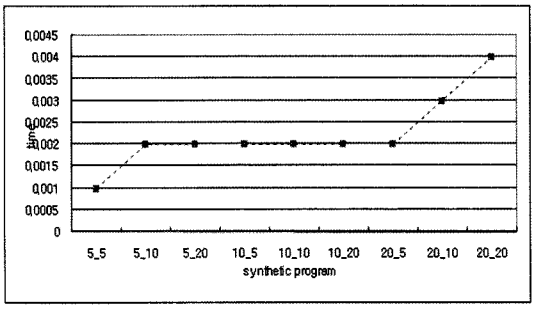

그림 5 최대병렬성/ 접근사건의 수 증가에 따른 경합탐지 소요시간

경합탐지 결과 제안된 도구는 모든 프로그램에서 최초경합만을 탐지하였으며, TC는 2번 프로그램에서 최초경합을 탐지하지 못하였다. 마지막으로 TA의 경우 1번, 4번, 6번 프로그램에서 탐지된 경합이 최초경합이 아님을 알 수 있다.

작성된 합성프로그램으로 최초경합을 탐지하기 위해 소요된 시간을 측정하여 본 논문에서 제안하는 도구의 효율성을 실험한다. 탐지시간은 1/1000 초 단위로 측정하였다. 그림 5에서 최대병렬성과 접근사건 수의 증가에 따른 경합탐지 소요시간을 보인다. 원 프로그램의 수행시간이 0.001초임에 비해서 프로그램을 두 번 수행함에도 불구하고 탐지시간이 매우 현실적임을 알 수 있다.

5. 결 론

기존의 경합탐지 도구들은 최초경합의 탐지를 보장하지 못하므로 본 논문에서는 두 번의 프로그램 수행으로

표 1 정확성 실험결과

	synthetic program	detected races		
		Our Tool	Thread Checker	Thread Analyzer
1	r1 w3 / r2 w4	r1-w4 r2-w3	r1-w4, r2-w3 w3-w4	w3-w4
2	r1 r4 / r2 r5 / w3 r6	r1-w3 r2-w3	r4-w3	r2-w3 r1-w3
3	r1 w4 / r2 r5 / r3 r6	r2-w4 r3-w4	w4-r2, w4-r5 w4-r3, w4-r6	w4-r3
4	r1 w4 / r2 w5 / w3 r6	r1-w3 r2-w3	w4-r2, w4-w5 r1-w5, w5-w3 r1-w3	w4-w5 w5-w3 w4-w3
5	r1 w4 / r2 r5 / w3 (r6 | r7)	r1-w3 r2-w3	w4-r2, w4-r5 w4-w3, r1-w3	w4-r2, r2-w3 w4-w3, w4-r7 w4-r6
6	r1 w4 / r2 w5 / w3 (r6 | r7)	r1-w3 r2-w3	w4-r2, w4-w5 r1-w5, w4-w3 r1-w3	w5-w3, w5-w4 w4-r7, w5-r7 w4-w3, w5-r6 w4-r6

프로그램 수행 중에 최초경합을 탐지하는 기능을 적용하여 효과적으로 경합을 탐지하는 도구를 제시하였다. 합성프로그램을 이용하여 기존 경합탐지 도구들과 비교 실험한 결과 기존도구들은 탐지된 경합이 최초경합임을 보장하지 못했으며, 제안된 도구는 최초경합을 정확하게 탐지함을 확인할 수 있었다. 또한 본 도구의 경합탐지 소요시간을 측정한 결과 두 번의 프로그램 수행으로 경합을 탐지함에도 불구하고 매우 현실적임을 알 수 있었다. 특히 쓰기사건이 먼저 발생하는 모델에서 효율적임을 알 수 있었다.

현재 사용자 컴퓨터에서 경합탐지 및 실행까지 모두 수행할 수 있도록 도구를 클라이언트 프로그램이 아닌 standalone으로 만드는 작업을 수행중이다. 또한 동기화 지원을 위한 레이블링 기법의 개발, 시각화 기법의 개발 등 다양한 기능을 도구에 적용함으로써 통합디버깅 환경을 구축하는 것이 향후과제로 남아있다.

참 고 문 헌

[1] Banerjee, U., B. Bliss, Z. Ma, and P. Petersen, "A Theory of Data Race Detection," Workshop on Parallel and Distributed Systems: Testing and Debugging (PADTAD), pp. 69-78, *Int'l Symp. On Software Testing and Analysis (ISSTA), ACM*, Portland, Maine, July 2006.

[2] 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.

[3] Bücker, H. M. , A. Rasch, and A. Wolf, "A Class of OpenMP Applications Involving Nested Parallelism," *Pro. of the 19th ACM Symposium on Applied Computing (SAC)*, 1: 220-224, Nicosia, Cyprus, New York, ACM, March 14-17, 2004.

[4] Dagum, L., and R. Menon, "OpenMP: An Industry-Standard API for Shared-Memory Programming," *IEEE Computational Science & Engineering*, 5(1): 46-55, IEEE, January/March 1998.

[5] OpenMP Architecture Review Board, "OpenMP Application Program Interface," www.openmp.org, version3.0, May 2008.

[6] Ha, K., Y. Jun, and K. Yoo, "Efficient On-the-fly Detection of First Races in Nested Parallel Programs," *Proc. of Workshop on State-of-the-Art in Scientific Computing (PARA)*, pp.75-84, Copenhagen, Denmark, June 2004.

[7] Jun, Y., and K. Koh, "On-the-fly Detection of Access Anomalies in Nested Parallel Loops," *Proc. of the 3rd ACM/ONR Workshop on Parallel and Distributed Debugging*, pp.107-117, ACM, San Diego, California, May 1993. Also in SIGPLAN Notices, 28(12): 107-117, ACM, Nov. 1993.

[8] Park, H., and Y. Jun, "Two-Pass On-the-fly Detection of the First Races in Shared-Memory Parallel Programs," *Proc. of the 2nd Symp. on Parallel and Distributed Tools (SPDT)*, ACM, Welches, Oregon, August 1998.

[9] Kim, J., and Y. Jun, "Scalable On-the-fly Detection of the First Races in Parallel Programs," *Proc. of the 12nd Int'l Conf. on Supercomputing (ICS)*, pp. 345-352, ACM, Melbourne, Australia, July 1998.

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

[11] C. Terboven, "Comparing Intel Thread Checker and Sun Thread Analyzer," *Minisymposium on Scalability and Usability of HPC Programming Tools, PARCO2007*, September 2007.

[12] Y. Lin, C. Terboven, D. an Mey and N. Copty, "Automatic Scoping of Variables lin Parallel Regions of an OpenMP Programs," *Workshop on OpenMP Applications and Tools (WOMPAT)*, 2004.

[13] SUN Microsystems, Inc. Sun Studio 12: Thread Analyzer User's Guide, 2007.

[14] Jun, Y., and K. Koh, "On-the-fly Detection of Access Anomalies in Nested Parallel Loops," *Proc. of the 3rd ACM/ONR Workshop on Parallel and Distributed Debugging*, pp.107-117, ACM, San Diego, California, May 1993. Also in SIGPLAN Notices, 28(12): 107-117, ACM, Nov. 1993.

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

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

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

강 문 혜

2001년 2월 경상대학교 컴퓨터과학과(이학사). 2003년 2월 경상대학교 대학원 컴퓨터과학과(공학석사). 2005년 2월 경상대학교 대학원 컴퓨터과학과(공학박사수료). 관심분야는: 운영체제, 분산병렬처리, 시스템/임베디드 소프트웨어

하 옥 균

2003년 2월 독학학위제 컴퓨터과학(학사). 2007년 2월 경상대학교 정보과학대학원(공학석사). 2007년~현재 경상대학교 대학원 정보과학과 박사과정. 관심분야는 운영체제, 병렬 처리 및 디버깅, 임베디드 소프트웨어, USN

전 용 기

1980년 2월 경북대학교 컴퓨터공학과(공학사). 1982년 2월 서울대학교 컴퓨터공학과(공학석사). 1993년 2월 서울대학교 컴퓨터공학과(공학박사). 1985년~현재 경상대학교 정보과학과 교수. 관심분야는 운영체제, 병렬 및분산처리, 시스템 프로그래밍