스레드 동기화가 없는 OpenMP 디렉티브 프로그램을 위한 효율적인 경합검증 도구

(An Efficient Tool for Verifying Races in OpenMP Directive Programs without Interthread Synchronization)

하 옥 균 † 강 문 혜 †
(OkKyoon Ha) (MoonHye Kang)

김 영 주 †† 전 용 기 †††
(YoungJoo Kim) (YongKi Jun)

요 약 OpenMP 디렉티브 프로그램에서 경합은 의도하지 않은 비결정적인 수행결과를 초래하므로 디버깅을 위해서 반드시 탐지되어야 한다. 하지만 이러한 경합을 탐지하는 기존의 도구인 Intel Thread Checker는 경합의 존재를 검증하지 못하며 경합을 탐지하는 비용이 크므로 비실용적이다. 이러한 문제를 해결하기 위해서 본 연구팀은 프로그램의 특성 및 사용자 요구사항의 분석결과를 이용하여 경합을 검증하는 도구를 개발하였으나 스레드 동기화가 없는 모델에서는 최적화되지 못하였다. 본 논문에서는 이러한 선행연구의 결과를 확장하기 위해서 스레드 동기화가 없는 프로그램 모델을 위한 효율적 레이블링 기법과 경합탐지 프로토콜 기법을 적용한다. 합성프로그램을 이용하여 실험한 결과로는 스레드 동기화가 없는 프로그램 모델에서 경합검증 시간이 기존의 도구보다 평균 250배 이상 빠르고, 총 접근사건 수가 동일하면 최대병렬성이 증가하여도 경합검증 시간이 감소함을 보인다.

키워드 : OpenMP, 디버깅, 경합, 경합탐지

Abstract Races must be detected for debugging OpenMP programs with directives, because they may cause unintended nondeterministic results of programs. Intel Thread Checker, an existing tool that can detects races, can not verify the existence of races and is often time-consuming and tends to require large space. To solve these problems, we developed a tool that verifies the existence of races using user requirements and analyzed model of programs. However, the tool does not have optimal performance in programs which have no synchronization for interthread coordination. This paper presents an optimal tool that applies the optimum labeling and protocol for program models without interthread coordination. For synthetic programs without interthread synchronization, the tool verifies races over 250 times faster than the previous tool on the average, even if the maximum parallelism increases in every case of which the number of total accesses are identical.

Key words : OpenMP, Debugging, Race, Race Detection

1. 서 론

OpenMP[1] 표준 API는 공유 메모리를 기반으로 하는 병렬 프로그래밍 모델로써 표준 C/C++와 Fortran 77/90을 확장하여 병렬화하는 컴파일러 디렉티브(compiler diretives)와 수행시간 라이브러리(run-time libraries)들의 집합이다. 이러한 OpenMP의 디렉티브 프로그램은 순차 프로그램에 스레드 병렬화를 위한 디렉티브를 삽입하여 병렬로 수행되어지고, 동기화 디렉티브를 삽입함으로써 스레드의 동기화가 있는 병렬 수행이 가능하다.

OpenMP의 디렉티브 프로그램[1]에서 심각한 오류중의 하나인 경합(race)[2]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.

OpenMP 디렉티브 프로그램에서의 경합을 수행 중에 탐지(On-the-fly Detection)[3,4]하기 위한 기존 도구[5,6]는 병행하는 스레드들을 순서적으로 수행한 정보를 이용하여 프로그램 수행 중에 데이타의 의존성 여부를 검사한 후에 경합을 탐지하므로 소요되는 시간과 공간의 비용이 크다. 또한, 이 도구는 내포병렬성이 존재하는 프로그램 모델에서 내포된 스레드를 부모 스레드와 순서적인 스레드로 간주하고, 동기화가 있는 프로그램 모델에서

· 이 논문은 제34회 추계학술대회에서 '스레드 동기화가 없는 OpenMP 디렉티브 프로그램을 위한 효율적인 경합검증 도구'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 경상대학교 정보과학과
jassmin@race.gnu.kr,
turtle@race.gnu.kr
†† 학생회원 : 한국정보통신대학교 IT공학부
kimyj@icu.ac.kr
††† 종신회원 : 경상대학교 정보과학과 교수
jun@gnu.ac.kr

논문접수 : 2007년 12월 14일
심사완료 : 2008년 3월 5일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제14권 제3호(2008.5)

경합에 참여할 가능성이 있는 접근사건을 무시하는 경우가 존재하므로 경합의 존재를 검증하지 못한다.

이러한 기존 도구의 문제점을 해결하기 위해서 본 연구팀은 프로그램의 수행 중 생성되는 병렬 스레드에 고유한 식별자를 부여하는 레이블링[3,7] 기법과 접근사건들의 발생 시 마다 병행성 여부를 검사하는 효율적 경합탐지 프로토콜[3] 기법을 OpenMP 디렉티브 프로그램의 모델에 따라서 분류하여 경합을 검증하는 도구[8]를 제안하였다. 이 도구는 스레드 동기화가 있는 프로그램 모델에서는 최적화되어 있지만, 스레드 동기화가 없는 프로그램 모델에서는 스레드 레이블 정보에서 부모 스레드의 정보를 탐색하기 위한 시간적 효율성이 낮으므로 최적화되지 못하였다.

본 논문에서는 이러한 선행연구의 통합도구를 확장하기 위해서 시간과 공간적으로 우수한 레이블링[9] 기법과 공간적 효율성이 높은 프로토콜[4] 기법의 조합을 적용한다. 합성프로그램을 이용하여 실험한 결과로는 스레드 동기화가 없고 비내포병렬성을 가진 프로그램 모델에서 경합검증 시간이 기존의 도구보다 평균 250배 이상 빠르고, 총 접근사건 수가 동일하면 최대병렬성이 증가하여도 경합검증 시간이 감소함을 보였다.

2절에서는 OpenMP 디렉티브 프로그램에서의 자료경합과 기존 경합탐지 도구의 동작원리에 대해서 설명한다. 3절에서는 본 연구에서 제안하는 효율적 경합검증 도구와 구현에 대해서 설명한다. 4절에서는 제안된 도구의 효율성 실험결과에 대해서 살펴보고, 마지막 5절에서는 결론을 제시한다.

2. 연구배경

본 절에서는 본 논문에서 대상으로 하는 OpenMP 디렉티브 프로그램과 이 프로그램에서 발생할 수 있는 오류인 자료경합에 대해서 설명하고, 기존 경합탐지 도구의 동작원리와 문제점에 대해서 설명한다.

2.1 OpenMP 프로그램의 자료경합

OpenMP 프로그램[1]은 표준 C/C++과 Fortran 77/90을 기반으로 작성된 공유메모리 병렬 프로그램을 위한 산업표준 프로그램 모델이다. 이 프로그램 모델은 순차프로그램을 병렬 프로그램으로 변환하는데 용이하게 하기 위해서 컴파일러 디렉티브를 제공한다. OpenMP에서 제공하는 컴파일러 디렉티브에는 병렬화 디렉티브, 작업공유 디렉티브, 데이타 환경 디렉티브, 동기화 디렉티브 등이 있다.

본 논문에서 대상으로 하고 있는 스레드 동기화가 없는 OpenMP 디렉티브 프로그램은 스레드 동기화를 위한 디렉티브를 사용하지 않고 스레드 병렬화 디렉티브만을 사용하여 병렬 수행하는 프로그램 모델이다. 일반적으로

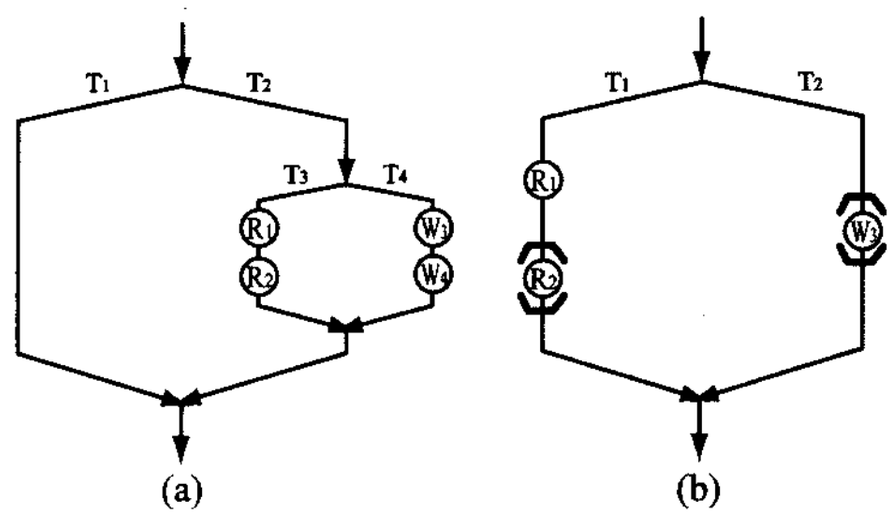

그림 1 내포와 동기화가 있는 프로그램 모델의 POEG

OpenMP 디렉티브를 사용하는 순차프로그램의 병렬화 개발 초기단계에는 스레드 동기화가 없는 프로그램을 생성한다. 이러한 병렬프로그램은 경합과 같은 의도되지 않은 비결정적 오류를 내포할 수 있다.

병렬 프로그램의 수행은 방향성이 있는 비순환 그래프인 POEG (Partial Order Execution Graph)[3]으로 나타낼 수 있다. POEG에서의 정점(vertex)은 병렬 스레드(T)의 생성과 합류를 나타내며, 정점들 사이의 간선(arc)은 정점에서 생성된 스레드를 나타낸다. 그리고 각 스레드에 접근하는 읽기와 쓰기 사건을 원으로 표시하고, r과 w로 나타내어 접근사건의 유형을 구분한다. 여기서 사용되는 숫자는 특정 수행 시에 접근사건들의 발생 순서를 의미한다. 그림 1은 내포병렬성과 스레드 동기화가 있는 프로그램 모델을 POEG으로 나타낸 것이다. POEG은 병렬프로그램의 수행 시에 스레드들 간의 부분적 순서(partial order) 관계를 나타낸다. 두 사건사이에 경로가 존재하면 두 사건이 순서화(ordered)관계에 있고 경로가 존재하지 않으면 이 두 사건은 병행(concurrent)관계에 있다. 그림 1(a)에서 r1와 r2는 그들 사이에 경로가 존재하므로 선행관계에 있으며, r1과 w3 사이에는 경로가 존재하지 않으므로 서로 병행관계에 있다.

이렇게 병행관계에 있는 두 개의 접근사건이 적어도 하나의 쓰기 사건을 포함하고 하나의 공유변수에 대해서 적절한 동기화가 없이 나타날 때 발생하는 오류를 경합(race)이라고 하며, ei-ej로 표시한다. 그림 1의 (a)에서는 경합 {r1-w3, r1-w4, r2-w3, r2-w4} 이 존재한다. 그림 1(b)는 스레드 동기화가 있는 프로그램 모델로 r2과 w3은 병행관계에 있으나, 동일한 락(lock)으로 보호되어 있으므로 경합 r1-w3 만이 존재한다.

2.2 기존의 경합탐지 도구

경합을 프로그램 수행 중에 공유변수의 접근사건에 대한 감시를 통해서 탐지하는 수행 중 탐지기법(On-the-fly Detection)[3,4]은 접근사건의 정보를 수집하는 과정에서 불필요한 정보를 삭제하므로 기억공간에 있어서 효율적이다. OpenMP 디렉티브 프로그램을 대상으로 수행 중에 경합을 탐지하는 기존의 도구는 Intel사의

Thread Checker[5,6]가 유일하다. Thread Checker에서 탐지되는 오류들은 교착상태, 경합, 논리적 오류 등이며 본 논문에서는 경합만을 분석 대상으로 한다.

Thread Checker는 OpenMP 디렉티브 프로그램의 순차적 수행 정보를 이용하여 데이타의 의존성 여부를 수행 중에 검사한 후에 경합을 탐지하는 Projection 기법[6]을 사용한다. 이그림 1(a)의 내포 병행성이 있는 프로그램 모델에서는 경합 {r1-w3, r1-w4, r2-w3, r2-w4}이 존재하지만 Thread Checker는 내포된 스레드들(T3, T4)을 부모 스레드(T2)와 동일한 스레드로 인식하고 접근사건 r1, r2, w3, w4를 순차적으로 수행시킴으로써 병행성 관계를 알지 못하므로 경합을 탐지하지 못한다. 그림 1(b)의 스레드 동기화가 있는 프로그램 모델에서는 경합 r1- w3이 존재하지만 Thread Checker는 순서관계에 있는 접근 사건들에 대해서 나중에 발생한 사건만을 유지하므로 경합에 참여하는 접근사건 r1을 제외시키고 r2만을 유지하여 경합을 탐지하지 못한다. 이와 같이 Thread Checker는 내포된 병렬 프로그램 모델과 스레드 동기화가 있는 병렬 프로그램 모델에서는 경합을 탐지하지 못하므로 경합의 존재 여부를 검증하지 못한다.

3. 최적화된 경합검증 도구

본 절에서는 기존 도구의 문제점을 해결하기 위해서 개발된 본 연구팀의 선행연구[8]에 대하여 설명하고, 이를 확장하기 위해서 본 연구에서 제안하는 스레드 동기화가 없는 OpenMP 디렉티브 프로그램 모델에 최적화된 경합검증 도구와 구현에 대해서 설명한다.

3.1 선행연구

본 연구팀은 기존 도구의 문제점을 해결하기 위해서 프로그램의 수행 중 생성되는 병렬 스레드에 고유한 식별자를 부여하는 레이블링[3,7]기법과 접근사건들의 발생 시 마다 병행성 여부를 검사하는 효율적 경합탐지 프로토콜[3] 사용한다.

이 도구는 스레드의 지역적 순차정보를 왼쪽에서 오른쪽으로 증가시키면서 내포깊이를 만족하는 공간적 효율성을 제공하는 레이블링[7] 기법과 임계구역이 존재하지 않는 스레드에서 공유변수에 대한 쓰기 접근사건이 발생한 경우에 경합여부를 검사하고 공유자료 구조내의 모든 접근사건들을 삭제하는 프로토콜[3] 기법을 사용한다.

이러한 선행연구의 통합도구에서 적용된 레이블링 기법과 프로토콜 기법의 조합은 스레드 동기화 모델에 최적화되었지만, 스레드 동기화가 없는 모델에 대해서는 스레드 레이블 정보에서 부모 스레드의 정보를 탐색하기 위한 시간적 효율성이 낮으므로 최적화되지 못했다.

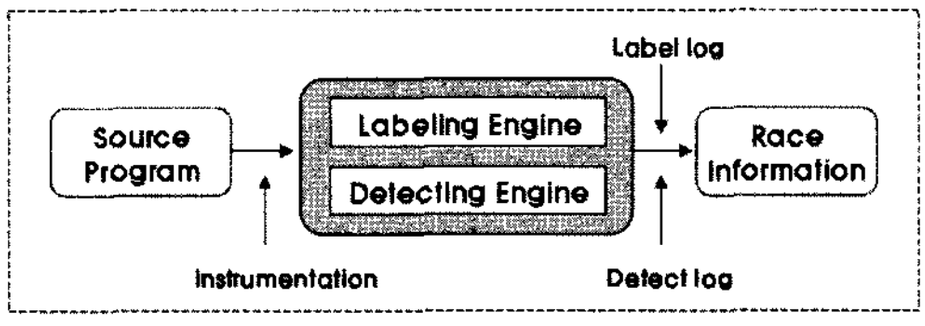

그림 2 경합검증 도구의 구조

3.2 최적화를 위한 경합검증 도구

스레드 동기화가 없는 프로그램 모델에서의 최적화되지 못한 문제를 해결하고 선행연구의 통합도구를 확장하기 위해서 제안하는 도구의 레이블링을 위해서는 지역자료 구조를 사용하여 병행성 정보를 생성하며 최대 병행 스레드 수에 비례하는 병목현상이 발생하지 않고 시간과 공간적으로 가장 우수한 성능을 보이는 Nest-Region (NR) Labeling[9] 기법을 적용한다. 그리고 탐지를 위한 프로토콜은 접근역사에 가장 최근의 쓰기 접근사건과 읽기 접근사건 중에서 가장 오른쪽, 왼쪽에서 발생한 읽기 접근 사건만을 유지함으로써 공간적 효율성이 높은 Mellor-Crummey의 프로토콜[4] 기법을 적용한다.

스레드 동기화가 없는 모델을 위한 NR Labeling과 Mellor-Crummey의 프로토콜 조합은 레이블링을 위한 시간과 공간적인 측면에서 우수한 성능을 보이고, 경합의 탐지 시 접근사건의 감시를 위한 공간의 효율성이 높으므로 최적화가 가능하다.

그림 2는 본 논문에서 구현하는 경합검증 도구에 대한 구조도이다. OpenMP 디렉티브 프로그램으로 작성된 원시프로그램(source programs)은 감시 라이브러리들이 삽입되어 변형된 프로그램(Instrumented Program) 형태로 레이블링 엔진과 프로토콜 엔진의 입력이 된다. 레이블링 엔진은 수행 중에 발생하는 스레드의 병행성 정보를 생성하는 label log를 바탕으로 스레드의 생성/합류 정보를 생성한다. 레이블링 엔진에 의해 생성된 정보를 입력받은 탐지 엔진은 수행 중 공유변수에 접근하는 접근 사건들의 감시 정보를 생성하는 detect log를 바탕으로 탐지된 경합의 정보를 생성한다.

3.3 최적화 경합검증 도구의 구현

레이블링 엔진은 라이브러리 형태의 Forker, Joiner 등으로 구성된다. Forker는 "#pragma parallel for"와 같은 디렉티브에 의해서 생성되는 스레드에 레이블 정보를 생성하고 유지하는 일을 담당하며, Joiner는 합류를 위한 디렉티브나 묵시적인 합류 명령에 의해서 병렬 스레드의 합류에 따른 레이블 정보의 생성과 삭제 그리고 유지를 담당한다. 탐지 엔진은 Mellor-Crummey의 프로토콜 기법을 사용하여 공유 변수의 접근 역사를 생성하고, 접근 사건들에 대한 감시와 레이블 정보의 분석을 통해서 경합을 탐지하고 보고한다.

```
01: main() {
02:    ...
03:    InitLabel(...); //레이블 자료구조 초기화
04:    InitDetection(...); //접근역사 초기화
05:    ...
06: #pragma omp parallel for shared(x) private(y,z,i)
07:    // 스레드 생성
08:      for(i=0;i<100;i++) {
09:      Forker(...);
10:          ...
11:          y = x * i;
12:      ReadChecker(...);
13:          ...
14:          x = z + i;
15:      WriteChecker(...);
16:          ...
17:      } //end for
18:      Joiner(...);
19:          ...
20: } //end main
```

그림 3 레이블링 및 탐지엔진의 적용 예

그림 3은 레이블링 엔진과 경합탐지 엔진이 대상 프로그램에 적용되는 것을 보인 것이다. 3번째 줄의 InitLabel은 이후에 생성될 모든 레이블들의 최상위 부모가 되는 레이블 정보를 생성하고 레이블을 위한 내포 영역의 범위와 기억공간 확보 등의 레이블 자료구조를 초기화한다. 4번째 줄의 InitDetection은 스레드들 간의 경합을 탐지하기 위해서 공유 자료구조인 접근 역사를 프로그램의 모델에 따라 기억 공간을 할당하고, 다른 모듈과 라이브러리 들이 활용할 수 있도록 초기화 한다. 9번째 줄의 Forker는 부모 스레드의 레이블 정보를 참조하여 현재 스레드의 레이블 정보를 생성한다. 그리고 12번째 줄의 ReadChecker는 탐지엔진의 구성원으로 현재의 읽기접근사건과 접근역사에 저장되어 있는 쓰기사건과 경합발생 여부를 판단하여 경합이면 보고한 후에 현재 접근사건 정보의 갱신여부를 결정하여 반영한다. 15번째 줄의 WriteChecker 역시 탐지엔진의 구성원으로 현재의 쓰기접근사건과 접근역사에 저장되어 있는 이전의 모든 읽기/쓰기사건과 경합발생 여부를 판단하여 경합이면 보고한다. 경합이 보고되면 접근역사에 저장되어 있는 모든 사건들의 레이블 정보를 삭제하고, 현재의 쓰기사건 정보를 접근역사에 추가한다. 18번째 줄의 Joiner는 부모 스레드의 레이블 정보와 현재 스레드의 레이블 정보를 참조하여 합류되는 스레드의 레이블 정보를 생성한다.

4. 실험

본 절에서는 경합검증과 최적화된 성능의 입증을 위해서 작성된 합성프로그램으로 경합검증과 경합을 탐지하는데 소요되는 시간을 측정하는 효율성 실험을 통해 기존의 도구와 제안된 도구를 비교한다.

1		r1 w3 r2 w4	r2-w3 r2-w4	-
2	r1	r2 w4 r3	r1-w4 r3-w4	r1-w4
3	r1 w2	r3 w5 r4	w2-r4, r1-w5, r3-w5, r4-w5, w2-w5	w2-r4, r1-w5, w2-w5

그림 4 경합검증 실험결과

4.1 합성 프로그램

본 도구와 기존 도구의 경합검증 여부와 최적화된 성능을 비교 실험하기 위해서 합성 프로그램(synthetic programs)을 사용한다.

경합의 검증을 비교실험하기 위해서 스레드의 동기화 여부와 내포병렬성의 여부를 달리하면서 작성하고, 효율성을 측정하기 위해서는 스레드 동기화가 없는 비내포병렬성 프로그램 모델에서 최대병렬성, 그리고 총 접근사건수 등을 달리하면서 작성한다. 최대병렬성은 2의 지수만큼 증가시키며, 스레드 당 접근사건은 100개, 200개, 300개씩 생성하고 접근사건의 종류는 읽기와 쓰기사건으로 번갈아 생성시킨다.

4.2 경합검증 및 효율성 실험

합성 프로그램으로 경합의 검증여부와 경합탐지 시에 소요되는 시간을 측정한다. 실험을 위해서 사용되는 시스템은 리눅스 운영체제하의 64bit Intel Xeon 듀얼 CPU 컴퓨터에 대상 프로그램 모델인 OpenMP 디렉티브 프로그램을 위한 Intel C/C++ 컴파일러를 설치하였으며 기존도구의 실험을 위해서 Thread Checker ver. 3.0을 설치하고 제안된 도구의 실험을 위해서 경합검증을 위한 라이브러리들을 설치한다. 이 라이브러리들은 C 언어로 구현되었다. 그리고 두 도구는 동일한 시스템 환경 하에서 실험한다.

그림 4는 스레드 동기화와 내포병렬성이 존재하는 합성 프로그램에서 제안된 도구와 기존 도구를 이용하여 경합을 검증한 결과이다. 세 개의 프로그램 중에 첫 번째 프로그램에서는 경합이 존재함에도 불구하고 기존도구에서는 탐지하지 못한 반면 본 도구에서는 경합을 탐지하여 검증이 가능함을 보인다. 도구의 효율성은 작성된 합성 프로그램에서 경합의 탐지를 위해 소요된 시간을 비교하여 측정한다.

그림 5는 스레드 동기화가 없고 비내포병렬성이 존재하는 Activity Management 프로그램 모델을 대상으로 최대병렬성을 2의 지수승으로 증가시키고 스레드 당 접

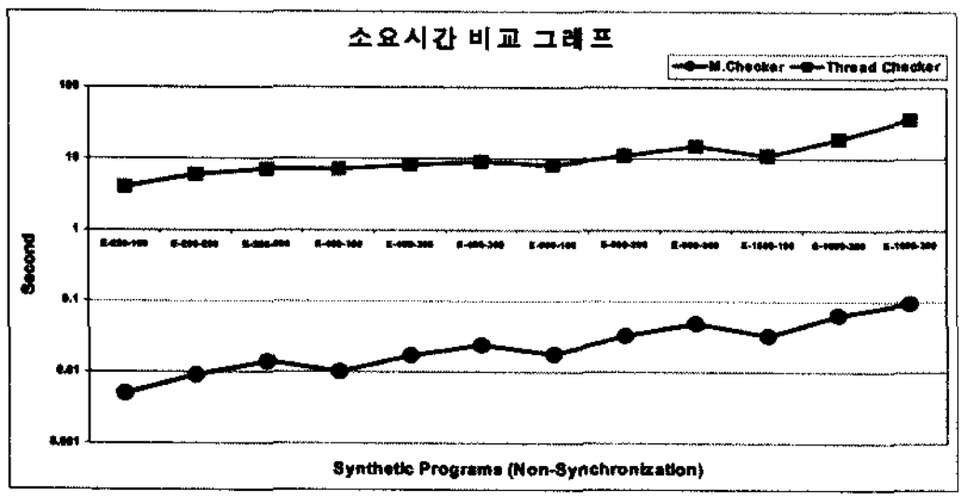

그림 5 총 접근사건 수에 따른 경합탐지 소요시간

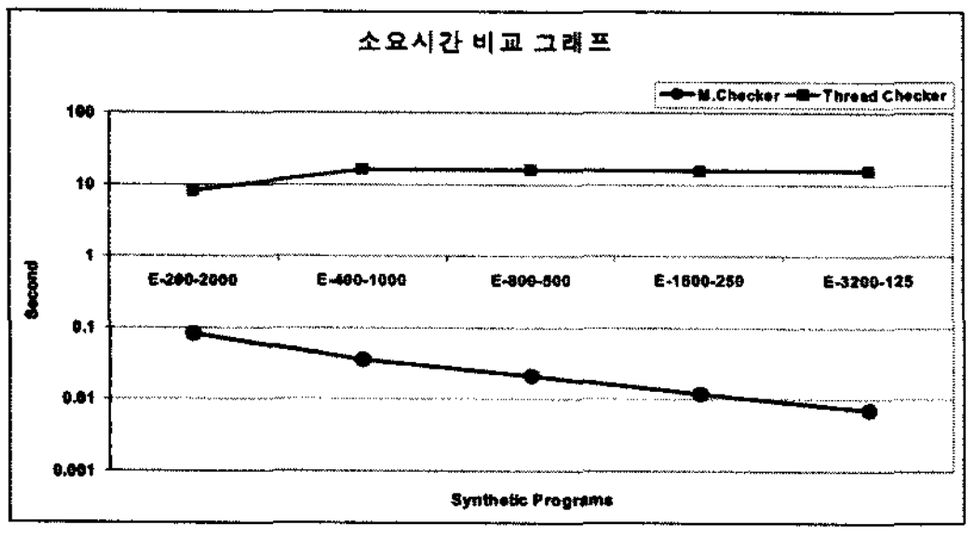

그림 6 최대병렬성에 따른 경합탐지 소요시간

근사건 수를 증가시키며 작성된 합성 프로그램에서 경합탐지의 소요시간을 측정한 결과이다. 그림 5에서 위의 선은 Thread Checker의 소요시간이며, 아래의 선은 본 도구의 소요시간 결과이다. 두 결과 모두 총접근사건수(최대병렬성 * 스레드 당 접근사건수)에 따라 증가하고 증가되는 패턴이 유사하며, 제안된 도구는 기존도구에 비해서 평균 250배 이상 빠름을 알 수 있다.

그림 6은 총접근사건 수를 4000으로 고정시키고 최대병렬성을 2의 지수승만큼 증가시키며 작성된 비동기적 비내포병렬성의 합성프로그램으로 경합탐지에 소요되는 시간을 측정한 것이다. 위의 선은 Thread Checker의 시간이며, 아래의 선은 본 도구의 측정결과이다. Thread Checker는 최대병렬성이 증가하여도 경합검증을 위한 소요 시간의 변화에 미치는 영향이 거의 없는 반면 제안된 도구는 최대병렬성이 증가함에 따라서 오히려 탐지시간이 감소함을 보인다. 그림 5와 그림 6의 효율성 비교 실험에 의해 제안된 도구는 스레드 동기화가 없는 프로그램 모델에 최적화되었음을 알 수 있다.

5. 결 론

OpenMP 디렉티브 프로그램에서 수행 중에 경합을 탐지하는 기존의 도구는 내포병렬성이 존재하는 프로그램 모델에서 내포된 스레드를 부모 스레드와 순서적인 스레드로 간주하고, 스레드 동기화가 있는 프로그램 모델에서 경합에 참여할 가능성이 있는 접근사건들을 무시하는 경우가 존재하므로 경합의 존재를 검증하지 못한다. 본 논문에서는 선행 연구에서 개발된 통합 도구를 확장하기 위하여 스레드 동기화가 없는 OpenMP 디렉티브 프로그램 모델에 최적화된 도구를 제안하였다.

제안된 경합검증 도구는 내포병렬성이 존재하는 프로그램 모델과 스레드 동기화가 없는 프로그램 모델에서 경합의 검증이 가능함을 보였다. 또한 일반적인 합성 프로그램을 이용하여 실험한 결과에서 스레드 동기화가 없는 모델의 경합검증 시간이 기존의 도구에 비해서 평균 250배 이상 빠르고, 총접근사건 수가 동일하면 최대 병렬성이 증가하여도 경합검증 시간이 감소하여, 최적화된 도구임을 보였다. 향후과제로는 제안된 도구에서 지원하는 OpenMP 디렉티브를 확장하여 다양한 모델에서 경합검증이 가능한 실용적 도구를 개발하는 것이다.

참 고 문 헌

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

[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] Dinning, A., and E. Schonberg, "Detecting Access Anomalies in Programs with Critical Sections," 2nd Wrokshop on Parallel and Distributed Debugging, pp. 85-96, ACM, May 1991.

[4] 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.

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

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

[7] I. Nudler, L. Rudolph, "Tools for the Efficient Development of Efficient Parallel Programs," First Israeli Conference on Computer Systems Engineering, May 1986.

[8] Kim, Y., M. Park, S. Park, and Y. Jun, "A Practical Tool for Detecting Races in OpenMP Programs," Proc. of 8th Int'l Conf. on Parallel Computing Technologies (PaCT), Krasnoyarsk, Russia, Lecture Notes in Computer Science, 3606: 321-330, Springer-Verlag, Sept. 2005.

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