Journal of The Korea Society of Computer and Information
Vol. 22 No. 11, pp. 1-7, November 2017
www.ksci.re.kr
https://doi.org/10.9708/jksci.2017.22.11.001

A Study on Filtering Techniques for Dynamic Analysis of Data Races in Multi-threaded Programs

Ok-Kyoon Ha*, Hongseok Yoo**

Abstract

In this paper, we introduce three monitoring filtering techniques which reduce the overheads of dynamic data race detection. It is well known that detecting data races dynamically in multi-threaded programs is quite hard and troublesome task, because the dynamic detection techniques need to monitor all execution of a multi-threaded program and to analyse every conflicting memory and thread operations in the program. Thus, the main drawback of the dynamic analysis for detecting data races is the heavy additional time and space overheads for running the program. For the practicality, we also empirically compare the efficiency of three monitoring filtering techniques. The results using OpenMP benchmarks show that the filtering techniques are practical for dynamic data race detection, since they reduce the average runtime overhead to under 10% of that of the pure detection.

▸Keyword: Data races, Data race detection, Dynamic analysis, Monitoring filtering techniques, multi-threaded programs, Efficiency

I. Introduction

멀티코어 및 멀티프로세서 시스템의 사용이 보편화되면서 다중 스레드 기반의 병렬 프로그램의 필요성이 강조되고 있다. 그러나 다중 스레드 기반 병렬 프로그램의 신뢰성은 동시에 수행되는 스레드 간의 미묘한 상호작용과 잠재적으로 초래할 수 있는 상호배제와 자료경합과 같은 동시성 오류 때문에 학문적으로도 여전히 어려운 일로 잘 알려져 있다[1-3]. 대표적인 동시성 오류인 자료경합을 탐지하기 위한 동적 분석기술은 추적된 정보를 분석 및 재 수행시켜 탐지하거나, 프로그램 수행 중에 자료경합을 탐지한다. 그러나 프로그램 수행의 감시와 충돌하는 모든 메모리 동작의 분석을 위해 발생하는 10~100배의 추가적인 오버헤드의 발생은 동적 자료경합 탐지기술의 가장 큰 단점이다. 이러한 동적 분석의 추가적인 오버헤드를 줄이는 방법으로 감시 필터링 기술이 등장하고 있으며, 본 논문에서는 다중 스레드 기반 프로그램의 동적 분석을 위한 감시 필터링 기술의 특징과 핵심 아이디어를 소개한다. 또한 소개된 필터링 기술을 동적 자료경합 탐지도구에 적용하여 각 기법들의 효율성을 실험적으로 비교 분석한다.

II. Background

1. Overview

방사선 치료기인 Therac-25의 방사능 과다누출 사고는 키보드 인터럽터 핸들러에 의한 사소한 자료경합으로부터 발생하였고[4], 2003년 8월 북미와 캐나다에서 50만 가구에 피해를 입힌 블랙아웃(blackout) 사고는 GE XA/21 오류로 보고된 제어 소프트웨어의 자료경합이 원인이다[5]. 자동차, 의료장비, 항공기, 철도 등 다양한 산업 분야에서 소프트웨어의 활용이 늘

• First Author: Ok-Kyoon Ha, Corresponding Author: Hongseok Yoo
*Ok-Kyoon Ha (okha@ikw.ac.kr), Dept. of Aeronautics & Software Engineering, Kyungwoon University
**Hongseok Yoo (hsyoo@ikw.ac.kr), Dept. of Aeronautics & Software Engineering, Kyungwoon University
• Received: 2017. 10. 13, Revised: 2017. 10. 20, Accepted: 2017. 11. 14.
• This work was supported by Kyungwoon University.

어나고 소프트웨어의 결함이 사람의 생명과 직결되면서 소프트웨어의 신뢰성과 품질에 대한 관심이 증가하고 있다.

특히 멀티코어 프로세서가 서버 컴퓨터에서 임베디드 시스템에 이르기까지 폭넓게 적용됨으로써 다중 스레드 소프트웨어가 정보 시스템을 넘어 다양한 산업으로 확산되고 소프트웨어 오류로 인해 사회적, 경제적 손실을 초래하는 사건들이 발생하고 있다. 이에 따라, 소프트웨어 개발 공정은 물론 개발이 끝난 제품의 오류를 사전에 진단하고 정상 작동여부를 검사하여 오류를 수정하는 소프트웨어 테스팅과 디버깅의 중요성에 대한 인식이 증가하고 있다.

자료경합을 탐지하기 위한 기술은 프로그램의 수행 없이 소스코드를 대상으로 가능한 모든 스레드 인터리빙을 고려하여 자료경합을 분석하는 정적 분석(static Analysis) 기법과 프로그램의 수행 동안 스레드의 발생과 동기화 정보 및 메모리 접근 등의 감시를 기반으로 자료경합을 분석하는 동적 분석(dynamic analysis) 기법으로 나눌 수 있다. 정적 분석은 실제 프로그램의 수행에서 발생할 수 없는 인터리빙까지 고려하여 다수의 잘못된 알람(false alarms)을 동반하기 때문에 결과의 신뢰성이 낮다[6-8]. 동적 분석기법은 프로그램의 추적, 재 수행(replay), 감시를 통해 수행 중에 발생한 실제 경합을 보고하며, 사후 추적(postmortem)법과 수행 중(on-the-fly) 탐지로 구분된다. 동적 분석을 위한 수행 중 탐지 기술은 다시 Lockset 분석, Happens-before 분석, 그리고 하이브리드(hybrid) 분석으로 구분할 수 있다[9-13].

다중 스레드 프로그래밍을 위한 사용자 수준(user-level)의 스레딩 모델은 OpenMP와 같이 병렬화 지시어 등을 컴파일러에 의해 자동적으로 스레드 병렬화가 OS 환경에 적절히 대응하도록 생성되는 암시적인 스레딩(implicit threading) 모델과 Pthread와 같이 프로그래머의 직접적인 스레드 생성과 소멸이 요구되는 명시적인 스레딩(explicit threading) 모델로 구분할 수 있다. 다수의 동적 자료경합 탐지를 위한 기술은 스레딩 모델의 구분 없이 사용자 수준 및 커널 수준의 스레드를 고려한다.

2. Research Trends

자료경합을 탐지하기 위한 동적 분석 기술은 대부분 프로그램의 수행 중에 자료경합을 탐지하는 수행 중 분석(on-the-fly) 탐지 기법을 사용하며 각 기술은 다음과 같다.

• **Happens-before 분석** : 스레드와 접근사건들 간의 순서관계 분석으로 자료경합을 탐지하며, 순서관계는 한 스레드 t_i가 다른 스레드 t_j보다 먼저 발생하였으면 두 스레드는 순서관계($t_i \rightarrow t_j$)로 분석하며, 두 스레드 t_i와 t_j가 $t_i \rightarrow t_j$ 또는 $t_j \rightarrow t_i$ 어느 하나도 만족하지 못하면 서로 병행관계($t_i \| t_j$)로 분석한다.

• **Lockset 분석** : 동일한 메모리 영역에 접근하는 서로 다른 두 스레드는 하나의 공통 lock으로 보호되어야 한다는 록킹 규칙(locking discipline)의 위배 여부의 분석을 통해 자료경합을 판별하는 기술이다.

• **Hybrid 분석** : Lockset 기술과 Happens-before 기술들이 보이는 정확성과 효율성의 trade-off 문제를 효과적으로 개선하기 위해 Lockset 분석을 기반으로 fork/join 연산에 의해 발생하는 접근 사건들의 Happens-before 관계를 분석한다.

이상의 기술을 사용하는 동적 자료경합 탐지기술의 주요 단점은 프로그램 수행의 감시와 충돌하는 모든 메모리 동작의 분석을 위해 발생하는 추가적인 오버헤드이다. Happens- before 분석의 경우 자료경합 탐지를 위한 수행시간이 원 프로그램 수행 시간의 10배에서 100배에 달하는 오버헤드가 발생한다. Lockset 분석은 Happens-before보다 수행시간이 빠르지만, false positives 문제가 있어 정확성이 떨어진다는 단점이 있다. 마지막으로 Hybrid 분석은 앞의 두 가지 기술의 장점을 혼합하여 Happens-before 분석의 수행시간 문제와 Lockset 분석의 정확성 문제를 개선한다. 그러나 Hybrid 분석 역시 정확성 문제를 해결하지 못하거나, 오리지널 프로그램의 수행에 비해 분석을 위한 수행시간 오버헤드가 여전히 크다는 문제를 가지고 있다.

동적 자료경합 탐지의 수행시간 오버헤드를 줄이기 위한 방법으로 샘플링(sampling) 기술이 개발되어 폭 넓게 적용하고 있다. 샘플링 기술은 프로그램에서 수행되는 병렬 수행 구간에 대해 감시 대상의 범위를 제한함으로써 불필요한 감시를 배제하고자 하는 기술이다. 그러나 이 기술은 적절한 샘플링 비율의 설정이 어려워 중복된 테스트를 유발하거나, 높은 샘플링 비율을 사용할 경우 기존의 기술과 큰 차이를 보이지 않게 된다. 결국 샘플링 기술은 감시영역 설정을 위한 샘플링 비율이 낮은 경우 (10% 미만) 효율적인 성능을 보이지만 잘못된 탐지 결과인 false alarm으로 인해 신뢰할 수 없는 결과를 보인다. 한편, 샘플링을 사용하는 자료경합 탐지 기술은 적절하고 타당한 샘플링 비율을 결정하기 위한 연구가 활발히 진행되고 있다.

III. Filtering Techniques

자료경합 탐지를 위한 동적 분석기법은 프로그램의 수행 동안 자료경합의 발생 유무를 판별하기 때문에 공유변수, 메모리 접근 연산, 스레드 연산 등에 대한 철저한 감시가 요구되기 때문에 수십 ~ 수백 배에 이르는 추가적인 시간/공간적 오버헤드가 발생한다. 본 절에서는 동적 자료경합 탐지의 추가적인 오버헤드를 줄이기 위한 필터링 기술을 소개한다.

1. Hierarchical Filtering Technique

동적 자료경합 탐지의 감시 필터링을 위한 계층적 필터링 기술(HIF: HIerarchical Filtering technique)은 프로그램의 수행을 위한 내부 메모리 연산을 세 단계 걸쳐 불필요한 연산을 걸러내고

이를 감시에서 배제하는 기술이다[14]. 이를 위해 HIF는 이진코드가 실행될 때 메모리에 적재되는 계층을 구분하여 이미지 수준(IML: Image Level), 섹션 수준(SEL: Section Level), 명령어 수준(INL: Instruction Level)에서 감시가 불필요한 정적 수행 영역 및 임시 메모리에 대해 각각 필터링을 수행한다.

IML의 필터링은 사용자가 개발한 프로그램만을 감시하기 위한 필터링으로 메모리에 적재된 일련의 오브젝트 이미지들은 사용자가 개발한 프로그램에서부터 시스템에서 제공되는 표준 라이브러리 및 제 삼자로부터 제공되는 라이브러리 등이 존재한다. 그러나 사용자는 자료경합을 탐지하기 위해 항상 모든 이미지 오브젝트에 대해 관심이 있는 것은 아니며 사용자가 개발한 프로그램과 라이브러리로 대상이 제한되는 것이 일반적이기 때문에 프로그램의 main 함수가 시작하기 전에 모든 오브젝트 이미지의 경로와 라이브러리 이름을 참조하여 선택적으로 감시대상에서 제외시킴으로서 메모리 접근사건들을 필터링하는 방법이다.

SEL의 필터링은 프로그램 실행 중에 사용되는 전역변수, 정적변수, 힙 영역 등과 같은 읽기와 쓰기 접근이 가능하고 동적으로 할당되는 데이터 섹션만을 감시하고 읽기전용 데이터와 상수 데이터가 고정적으로 할당되어 있는 데이터 섹션에 접근하는 명령어에는 감시코드를 삽입하지 않음으로서 감시에 대한 배제를 수행한다.

한편, 직접적인 감시코드 삽입은 항상 코드섹션에서 이루어지기 때문에 자료경합 탐지를 위한 메모리 주소영역에 대한 접근은 코드섹션 내의 각 명령어 별로 감시해야 한다. 명령어에 대해 공유 메모리 접근유무를 명령어의 연산자(opcode)를 분석하여 1차적으로 판단하고, 피연산자(operand)의 주소값을 이용하여 정확하게 판단함으로써 INL의 필터링을 수행한다.

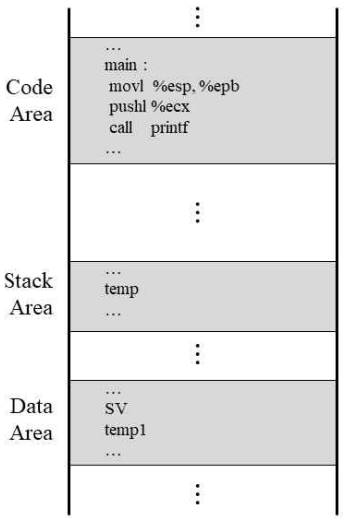

Fig. 1. An example of Memory mapping for a program

예를 들어 Fig. 1에서 코드 영역의 피연산자(operand) 분석과 참조 주소의 비교를 통해 메모리의 스택 영역을 지시하는 포인터 ebp와 esp와 같이 자료경합을 유발하는 접근 충돌에 관여하지 않는 메모리 연산들을 감시 대상에서 배제할 수 있다. Fig. 1의 경우 HIF를 적용한 동적 자료경합의 경우 데이터 영역의 SV만을 공유 메모리 영역으로 판별하게 되어 감시를 위한 감시코드를 삽입한다.

HIF를 이용한 동적 자료경합 탐지는 기존의 동적 탐지에 비해 평균 수행시간을 50% 이하로 단축한다.

2. Redundant Event Filtering Technique

중복 이벤트 필터링(REF: Redundant Event Filtering)은 중복된 메모리 및 스레드 연산에 관련된 이벤트를 감시대상에서 제외하는 방법을 통해 수행 중 분석과 같은 동적 분석 기법에서 발생하는 추가적인 시간적 및 공간적 오버헤드를 줄인다[15]. 이를 위해 REF는 공유 메모리에 대해 반복적으로 접근하는 읽기 또는 쓰기 접근사근에 대한 감시를 배제하고 수행 중 감시를 위한 감시코드의 삽입을 제한한다.

각 스레드 세그먼트(segment)별로 첫 번째 읽기 또는 쓰기 접근에 대한 정보와 감시를 유지하면서 이후 발생하는 동일한 형태의 접근사근은 배제함으로써 동적 분석의 오버헤드를 획기적으로 줄인다. REF와 같이 단일 스레드 세그먼트 상의 첫 번째 접근 이벤트만을 고려하는 것은 발생하는 자료경합과 이후 새롭게 발견될 수 있는 자료경합에 영향을 미치기 때문에 다중 스레드 프로그램의 디버깅을 위해서는 중요한 기법으로 잘 알려져 있다.

명시적인 스레딩 기법을 위한 동적 자료경합 기법[15]에서 보인 바와 같이 REF를 적용한 자료경합의 성능은 우수한 정확성을 보이면서도 시간적 및 공간적인 오버헤드 문제를 개선한다. 동적 자료경합 탐지를 위한 REF는 한 접근 이벤트 e_i가 발생하였을 때 동일한 스레드 세그먼트 상의 동일한 접근 이벤트 e_j가 이후에 발생한다면 식 (1)의 조건 따라 이후에 발생한 e_j는 감시 대상에서 제외되어 진다.

$$IsFiltered(e_i, e_j)\begin{cases} T(e_i) = T(e_j) \wedge\ t_i = t_j & \wedge \\ t_i.lockset \subseteq\ t_j.lockset & \vee \\ t_i.lockset = t_j.lockset = \varnothing \end{cases} \quad (1)$$

식(1)에서 $T(e_i)$는 접근 이벤트 e_i의 형태로 공유 메모리에 대한 읽기 또는 쓰기 중 하나이고, $t_i.lockset$은 e_i가 발생한 스레드 세그먼트 t_i가 동작하는 동안 유지되는 동기화를 위한 lock들의 집합을 의미한다.

Fig. 2는 멀티 스레드 프로그램의 한 수행의 예로 T_1스레드의 수행에서 읽기 접근 이벤트 인 R_1과 R_2가 발생하였다. T_2스레드의 수행에서 읽기 이벤트인 R_3가 발생하였고, 이후 쓰기 이벤트인 W_4가 발생한 경우이다. 두 스레드의 동기화를 위해 각 스레드별 이벤트들은 동일한 lock인 L_1으로 보호되고 있으며, R_2는 다른 스레드 또는 이벤트와의 동기화를 위해 L_1

에 내포된 lock인 L_2에 의해 추가적으로 보호되고 있다. 이 경우 식(1)의 조건에 따라 R_2는 자료경합 탐지를 위한 감시대상에서 제외되어 진다. 즉, R_1과 R_2는 동일한 스레드 세그먼트 상에 발생한 동일한 이벤트로 R_1의 lock 집합 $R_1.lockset = \{L_1\}$은 $R_2.lockset = \{L_1, L_2\}$의 부분 집합이다. 이러한 조건으로 인해 두 스레드의 동기화 L_1의 수행 순서에 상관없이 R_2는 W_4와 항상 순서화 관계(Happens-before 관계를 의미하며, Fig. 2의 경우 $R_2 \rightarrow W_4$ 또는 $W_4 \rightarrow R_2$로만 수행됨)를 유지하기 때문에 자료경합이 절대로 발생할 수 없음을 알 수 있다.

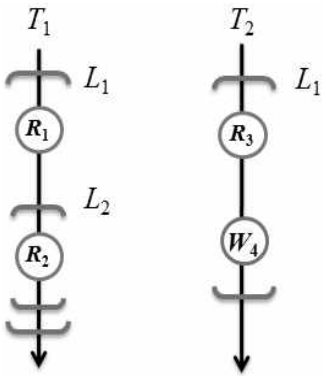

Fig. 2. An execution of a multi-threaded program

3. Loop Region Filtering Technique

동적 자료경합 탐지의 수행시간 오버헤드는 상용 프로그램의 테스트 기간에 실제로 적용하기에는 비효율적이며, 단순 반복 영역에서 공유변수에 대한 접근이 존재하는 경우가 발생하면 그 반복 횟수만큼 비례하여 수행시간 오버헤드가 증가하는 문제가 있다.

단순 반복영역은 특별한 조건문의 사용을 제외하고는 동일한 코드 수행을 반복하는 형태이다. 일반적으로 사용자 프로그램에는 거의 필수적인 동작이지만, 자료경합 탐지를 위해 스레드 동작과 메모리에 대한 접근을 판단하는 경우에는 불필요한 반복의 진행일 가능성이 매우 높다. 동일한 동작을 하는 반복코드는 동일한 수행 양상을 반복하기 때문에 감시 결과는 동일하다. 따라서 반복 영역 필터링(LRF: Loop Region Filtering)은 이러한 동일한 동작을 위해 반복되는 코드를 대상으로 실제로 수행되는 두 개의 스레드만 감시함으로써 동적 자료경합 탐지의 오버헤드를 획기적으로 줄인다[16].

LRF는 필터링 대상이 될 반복 영역을 정적 분석을 통해 식별하고 이를 바탕으로 동적 자료경합 탐지 시에 감시 필터링을 적용한다. Fig. 3은 LRF의 수행구조를 보이고 있다. 정적 분석을 통해 반복 영역을 식별한 후, 동적 자료경합 탐지 시에 루프 영역에 대한 필터링이 수행되어진다. 자료경합 탐지 대상 프로그램이 수행될 때 일반적인 동적 분석 기반 자료경합 탐지기술은 모든 명령어에 대해 자료경합 탐지를 위한 감시 코드를 삽입한다. 이에 해당하는 부분이 Fig. 3의 점선 사각형 부분을 제외한 모든 부분이며, 루프 영역 필터링에 의해 점선 사각형에 해당하는 부분이 중간에 삽입되어 수행된다.

LRF는 정적 분석을 통해 식별된 반복 영역과 현재 읽혀진 명령어를 비교해 반복 영역인지 아닌지를 판단하는 과정을 거치며, 각 반복 영역의 식별자는 해당 반복 영역의 수행 횟수를 확인하기 위한 Count변수를 두고 반복 횟수를 감시한다. 현재 명령어가 반복문 내에 존재하는 경우는 이 Count 변수를 확인하여 현재 반복 영역의 수행 횟수가 2회 미만인 경우는 반복 영역의 수행을 알리는 Count 값을 갱신하고 모니터링을 수행한다. 반복 영역의 수행 횟수가 2회 이상인 경우는 모니터링 단계로 가지 않고 곧바로 다음 명령어를 읽게 되어 자연적으로 필터링이 된다. 이러한 과정을 통해 LRF는 반복 영역에 대한 감시는 2회씩만 하게 되고, 나머지 수행에 대해서는 모니터링 조차도 하지 않게 되어 감시 오버헤드를 획기적으로 줄인다.

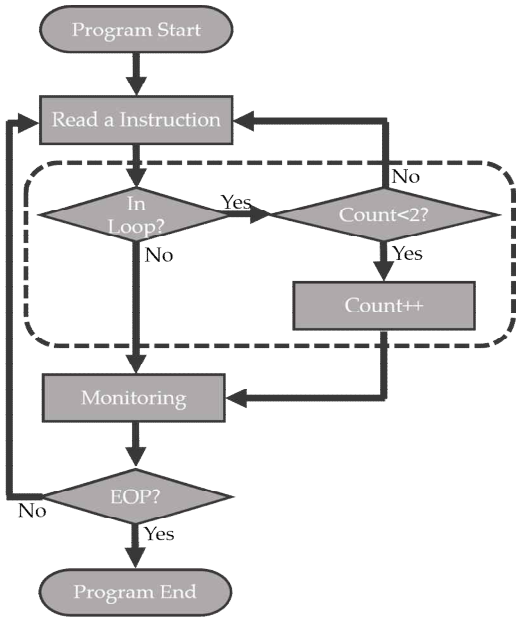

Fig 3. Process of Loop Region Filtering

IV. Empirical Comparison

본 절에서는 감시 오버헤드를 줄이는 필터링 기술들을 자료경합탐지 도구에 적용하여 그 효과를 실험적으로 비교하였다. 이진 코드에 대한 계층적 기법인 HIF를 기본적으로 적용하여, REF를 적용한 경우, LRF를 적용한 경우, 그리고 REF와 LRF를 모두 적용한 경우로 나누어 실험한 결과 REF와 LRF를 모두 적용한 경우 동적 자료경합 탐지를 위해 실용적임을 보였다

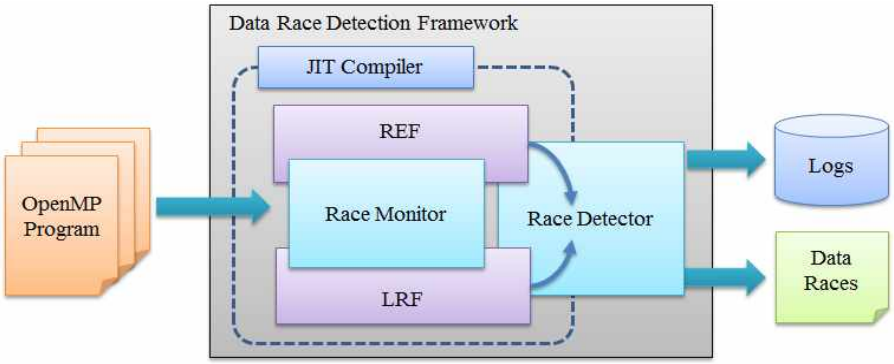

Fig. 4. Overall architecture of our data race detector

1. Design of Experiments

실험을 위해 PIN 이진 소프트웨어 프레임워크[17-18]를 기반으로 단일 동적 자료경합 탐지 도구를 구현하였다. 구현된 PIN-tool에는 이진 분석 시 HIF가 기본적으로 적용되도록 설계하고, REF와 LRF를 선택적으로 적용될 수 있도록 설계하였다. Fig. 4는 구현한 PIN기반의 동적 자료경함 탐지 도구의 구조를 보인다.

구현과 실험을 위한 환경은 Intel Xeon 2.4GHz CPU와 48GB 메인 메모리 시스템 상에서 커널 2.6 기반의 CentOS를 설치하고, gcc-4.1.2 컴파일러를 설치하였다. 또한 필터링 기법의 효율성을 비교하기 위하여 멀티 스레드 프로그램인 OpenMP 벤치마크를 사용하였으며, Only-HIF Case, With-REF Case, With-LRF Case, With-All Case의 4가지 경우를 고려하여 각 경우에 대한 수행시간 오버헤드를 측정하여 비교하였다.

실험을 위한 OpenMP 벤치마크는 멀티스레드 프로그램의 실험에 널리 사용되는 OmpSCR(the OpenMP Source Code Repository)[19]의 응용 프로그램 중에서 공유 변수의 개수, 병렬 루프의 크기, 수행 오버헤드 등의 프로그램 특성을 고려하여 FFT6, MD, Mandelbrot, PI의 네가지 응용 프로그램을 선정하여 실험하였다. Table 1은 선정된 네 가지 응용 프로그램의 특징을 정리한 것이다.

Table 1. The features of OpenMP benchmark applications

App.	Lines	Events		Locks	Loop Count
		Read	Write		
FFT6	542	2285K	15399K	1	38K
MD	266	23584K	9451K	0	5632
Mandel.	144	114537K	8	0	1024
PI	83	20000K	8	0	50000K

2. Results and Analysis

자료경합 탐지를 위해 시간 및 공간적 오버헤드가 우수한 RaceChaser 알고리즘[11]을 적용하였으며, 구현된 도구로 각 필터링 기법을 적용한 탐지 결과는 모두 동일하였다. Table 2는 탐지 도구를 이용하여 탐지된 자료경합의 개수를 보인다. 정확성을 검증하기 위한 결과에서 REF와 LRF 모두 동일한 결과를 보일 뿐만 아니라 두 필터링 기법을 모두 적용한 결과에서도 자료경합을 이상 없이 탐지하여 정확성을 유지하였다.

Table 2. The results of race detection

App.	Repored Races				Verified Results
	OH	WR	WL	WA	
FFT6	2	2	2	2	2
MD	2	2	2	2	2
Mandel.	0	0	0	0	0
PI	0	0	0	0	0
OH: Only-HIF Case, WR: With-REF Case WL: With-LRF Case, WA: With-All Case					

응용 프로그램을 이용한 동적 자료경합 탐지의 네 가지 경우에 대한 수행시간을 측정하여 효율성을 비교하였다. Table 3과 Fig. 5는 측정된 수행 시간 오버헤드의 결과를 나타낸다.

Table 3. The Measured results for applications

App.	Runtime Overheads			
	OH	WR	WL	WA
FFT6	49.5	40.5	9.6	2.4
MD	9.2	8.2	2.7	2.5
Mandel.	97.2	93.0	4.2	3.5
PI	60.9	58.2	14.2	10.5

실험의 결과는 각 경우 별로 20번의 실험을 통해 측정된 결과 중 최저 시간과 최고 시간을 제외한 평균을 적용하였으며, 자료경합 탐지를 적용하지 않은 순수 응용 프로그램의 수행 시간(Original run time)에서 추가적으로 발생한 시간적 오버헤드를 배율로 나타내었다.

Table 3과 Fig. 5의 결과에서 보인 바와 같이 Only-HIF의 경우 최소 9.2 배에서 최대 97.2 배의 시간적인 오버헤드가 발생하였으며, With-REF의 경우 최소 8.2 배와 최대 93 배의 시간적 오버헤드를 발생시킨 반면, With-LRF의 경우 최소 2.7 배와 14.2 배로 획기적으로 수행 시간 오버헤드를 줄인다. 특히 Mandelbrot의 경우 Only-HIF와 With-REF가 90배 이상의 수행 시간을 오버헤드를 유발하는 반면 With-LRF의 경우 4.2 배의 수행 시간 오버헤드만을 발생시키는 등 모든 어플리케이

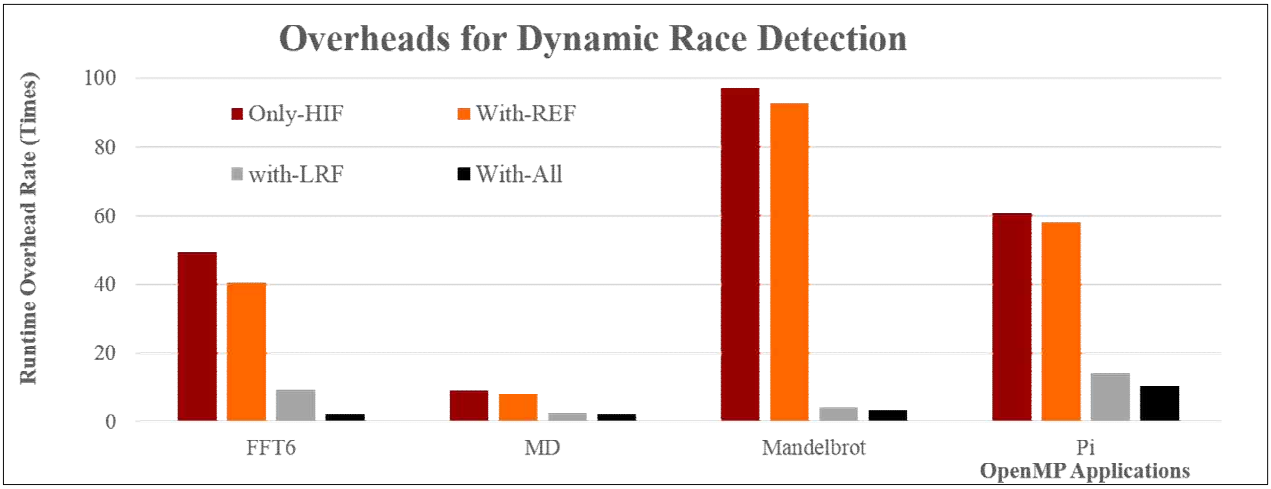

Fig. 5. Measured Overheads for Dynamic Race Detection

션에 대해 15배 이내의 오버헤드만을 발생시켰다.

최종적으로 HIF만을 적용한 동적 자료경합 탐지는 평균 54.2배의 수행 시간 오버헤드를 야기하였고, REF를 추가적으로 적용한 경우 평균 49.9배의 수행 시간 오버헤드를 발생시켰다. 반면, HIF와 LRF를 적용한 자료경합 탐지의 경우 평균 7.7배의 수행 시간 오버헤드만을 발생시켜 실용적이다. 특히 HIF와 REF 및 LRF를 모두 적용한 경우 평균 4.7배의 수행 시간 오버헤드만으로 자료경합을 정확하게 탐지할 수 있어 동적 자료경합 탐지를 위해 매우 실용적인 방법임을 실험적으로 보였다.

실험결과에서 나타난 바와 같이 HIF와 REF의 적용은 프로그램의 특성 중 감시 대상 이벤트의 개수와 루프의 반복 횟수가 많은 경우 효과가 적은 반면, LRF을 적용한 탐지에서는 MD와 Mandelbrot과 같이 루프의 반복 횟수가 프로그램에 미치는 영향이 큰 응용 프로그램의 경우 감시 필터링 효과가 크게 나타남을 분석할 수 있다. 또한, FFT6와 같이 감시 대상 이벤트의 개수와 루프의 반복이 큰 경우 REF와 LRF를 모두 적용한 경우 감시 필터링 효과가 매우 크게 나타남을 분석할 수 있다. 따라서 With-All Case와 같이 모든 감시 필터링 기법을 적용함으로써 이벤트 필터링과 루프 필터링을 동시에 실시함으로써 동적 경합탐지의 효율성을 향상 시킬 수 있음을 실험적 비교를 통해 분석하였다.

V. Conclusions

수행 중 분석과 같이 멀티 스레드 기반 프로그램에서 발생하는 자료경합을 동적으로 탐지하기 위해서는 추가적으로 발생하는 오버헤드를 감소시키는 것이 매우 중요하다. 본 논문에서는 이러한 추가적인 오버헤드를 감소시키기 위한 기술로 감시 필터링 기법인 계층적 감시 필터링(HIF)기법, 중복 이벤트 필터링(REF), 반복 영역 필터링(LRF)에 대해 소개하고, 각 기법을 적용한 동적 자료경합 탐지의 효율성을 실험적으로 비교하였다. 자료경합 탐지 도구를 구현하여, Only-HIF, With-REF, With-LRF, With-All Case로 구분하고 OpenMP 벤치마크 프로그램으로 실험한 결과에서 With-LRF Case가 Only-HIF Case가 야기한 수행 시간 오버헤드의 10%만을 발생 시켰으며, With-All Case의 경우 Only-HIF Case의 수행 시간 오버헤드를 8.7%로 획기적으로 줄이면서도 자료경합을 정확하게 탐지하여 동적 자료경합 탐지를 위해 실용적임을 보였다.

REFERENCES

[1] Netzer, Robert H. B. and Miller, Barton P, “What are Race Conditions?: Some Issues and Formalizations,” in Proceeding of Letters on Programming Languages and Systems, LOPLAS 1992, pp. 74-88, ACM, 1992.

[2] E. Farchi, Y. Nir, and S. Ur, “Concurrent Bug Patterns and How to Test Them,” in Proceeding of the 17th International Symposium on Parallel and Distributed Processing, IPDPS 2003. pp. 7. IEEE, 2003.

[3] U. Banerjee, B. Bliss, Z. Ma, and P. Petersen, “A Theory of Data Race Detection,” in Proceedings of the 2006 workshop on Parallel and distributed systems: testing and debugging, PADTAD 2006, pp. 69-78, ACM, 2006.

[4] N. G. Leveson, C. S. Turner, “An Investigation of the Therac-25 Accidents,” IEEE Computer, vol. 26(7), pp. 18-41, July, 1993.

[5] U. S. A and Canada, “Final Report on the August 14, 2003 Blackout in the United States and Canada: Causes and Recommendations,” Technical report, U.S.-Canada Power System Outage Task Force, August, 2003.

[6] D. Callahan and J. Sublok, “Static Analysis of Low-level

Synchronization," SIGPLAN Not., vol. 24(1), pp. 100-111, ACM, November, 1988.
[7] C. E. McDowell, "A Practical Algorithm for Static Analysis of Parallel Programs," Journal of Parallel and Distributed Computing, vol. 6(3), pp. 515-536, Springer-verlag, June, 1989.
[8] D. Engler and Ken Ashcraft, "RacerX: Effective, Static Detection of Race Conditions and Deadlocks," in Proceedings of the nineteenth ACM symposium on Operating systems principles, SOSP 2003, pp. 237-252, ACM, 2003.
[9] Ha, O.-K., "Case Study of Dynamic Detectors for Data Races," in Proceeding of International Conference on Electronic Engineering and Computer Science (EECS 2013), IERI Procedia, Beijing, China, vol. 4, (2013), pp. 174-180.
[10] C. Flanagan and S. N. Freund, "FastTrack: Efficient and Precise Dynamic Race Detection," in Proceedings of the 2009 ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2009, pp. 121-133, ACM, 2009.
[11] Ha, O.-K., Jun, Y.-K., "An Efficient Algorithm for On-the-fly Data Race Detection Using an Epoch-Based Technique," Journal of Scientific Programming, Article No. 13, January, 2015.
[12] T. Elmas, S. Qadeer, and S. Tasiran, "Goldilocks: A Race and Transaction-aware Java Runtime," in Proceedings of the 2007 ACM SIGPLAN Conf. on Programming Language Design and Implementation, PLDI 2007, pp. 245-255, ACM, 2007.
[13] A. Jannesari, B. Kaibin, V. Pankratius, and W. F. Tichy, "Helgrind+ : An Efficient Dynamic Race Detector," in Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing, IPDPS 2009, pp. 1-13, IEEE, 2009.
[14] Ha, O.-K., Jun, Y.-K., "Effective Monitoring Memory Operations for Dynamic Race Detection through Hierarchical Filtering Method," International Journal of Multimedia and Ubiquitous Engineering, Vol. 9(4), pp. 199-208, April, 2014.
[15] Ha, O.-K., Kuh, I.-B., G. M. Tchamgoue, Jun, Y.-K., "On-the-fly Detection of Data Races in OpenMP Programs," in Proceeding of the 2012 Workshop on Parallel and Distributed Systems: Testing, Analysis, and Debugging (PADTAD'2012), pp. 1-10, ACM, Minneapolis, USA, July 2012.
[16] Park, S.-W., Ha, O.-K., and Jun, Y.-K., "A Loop Filtering Technique for Reducing Time Overhead of Dynamic Data Race Detection," in Proceeding of the 8th International Conference on Database Theory and Application (DTA 2015), pp. 29-32, IEEE, Jeju, Korea, 2015.
[17] H. Patil, C. Pereira, M. Stallcup, G. Lueck, and Cownie, "Pinplay: A Framework for Deterministic Replay and Reproducible Analysis of Parallel Programs," in Proceeding of the 8th annual IEEE/ACM international symposium on Code generation and optimization, CGO 2010, pp. 2-11, ACM, 2010.
[18] A. R. Bernat and B. P. Miller, "Anywhere, Any-time Binary Instrumentation," in Proceedings of the 10th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools, PASTE 2011, pp. 9-16, ACM, 2011.
[19] A. J. Dorta, C. Rodriguez, F. d. Sande, A. Gonzalez-Escribano, "The OpenMP Source Code Repository," in Proceedings of the 13th Euromicro Conference on Parallel, Distributed and Network-Based Processing, pp. 244-250, 2005.

Authors

Ok-Kyoon Ha received the BS degree in Computer Science under the Bachelor's Degree Examination Law for Self-Education from National Institute for Lifelong Education, and the MS and Ph.D. degree in Informatics from Gyeongsang National University in 2007 and 2012, respectively. Dr. Ha is currently a Professor in Department of Aeronautics & Software Engineering at Kyungwoon University, South Korea. His research interests including parallel and distributed programming, software testing and debugging, embedded system programs, dependable software, and software development activities for avionics.

Hongseok Yoo received the BS and MS degrees in computer engineering from Kyungpook National University, Daegu, Korea, in 2005 and 2007, respectively. He also obtained the PhD degree in computer science and engineering from Kyungpook National University in 2014. Between 2007 and 2008, he worked as a visiting researcher at Daegu Gyeongbuk Institute of Science and Technology (DGIST), Korea. He is currently an assistant professor with the Department of Aeronautics & Software Engineering at Kyungwoon University, Gumi, Korea. His research interests include protocol design and implementation in vehicular networks, Avionics Data Networks, Device-to-device communications, Internet of Things.