DOI: 10.3745/KIPSTA.2010.17A.1.9

내부적 비결정성을 가진 공유 메모리 병렬 프로그램에서 잠재적 경합탐지를 위한 전처리기

김 영 주[†] · 정 민 섭[††] · 전 용 기[†††]

요 약

OpenMP와 같은 공유 메모리 기반의 병렬 프로그램에서 발생하는 경합은 프로그래머가 의도하지 않은 비결정적인 수행 결과를 초래하므로 반드시 탐지되어야한다. 이러한 경합의 존재를 수행 중에 검증하는 기존의 기법들은 내부적 비결정성이 존재하지 않는 프로그램에 대해서만 가능하다. 하지만 이 조건을 만족하지 못한다면 경합검증을 위해서 각 임계구역마다 적어도 N! 만큼의 프로그램 수행이 필요하다. 여기서 N은 탐지 대상 프로그램이 가진 최대 병렬성을 의미한다. 본 논문에서는 프로그램 슬라이싱을 이용하여 내부적 비결정성을 가진 프로그램에 존재하는 비결정적 접근사건을 정적으로 분석하고, 이 정보를 이용하여 한번의 수행으로 실제 경합뿐만 아니라 잠재적 경합까지 탐지할 수 있는 경합 전처리기를 제안한다. 제안된 도구는 OpenMP 병렬 프로그램에서 발생할 수 있는 비결정적 접근사건들에 대해서 항상 결정적으로 감시할 수 있으므로 임계구역 가진 프로그램 모델에 적용할 수 있는 어떠한 경합탐지 프로토콜을 사용하더라도 경합 검증이 가능하다. 본 도구의 실험적 증명을 위해서 비결정성이 포함된 합성 프로그램, 공인된 벤치마크 프로그램인 OpenMP Microbenchmark, NAS Parallel Benchmark, 그리고 OpenMP 응용 프로그램을 이용하여 제안된 도구의 정확성을 보인다.

키워드 : 병렬 프로그램, 경합, 수행중 경합탐지, 내부적 비결정성, 실제 경합, 잠재적 경합, 합성 프로그램, Microbenchmark, NAS Parallel Benchmark

A Preprocessor for Detecting Potential Races in Shared Memory Parallel Programs with Internal Nondeterminism

Young-Joo Kim[†] · Min-Sub Jung[††] · Yong-Kee Jun[†††]

ABSTRACT

Races that occur in shared-memory parallel programs such as OpenMP programs must be detected for debugging because of causing unintended non-deterministic results. Previous works which verify the existence of these races on-the-fly are limited to the programs without internal non-determinism. But in the programs with internal non-determinism, such works need at least N! execution instances for each critical section to verify the existence of races, where N is the degree of maximum parallelism. This paper presents a preprocessor that statically analyzes the locations of non-deterministic accesses using program slicing and can detect apparent races as well as potential races through single execution using the analyzed information. The suggested tool can deterministically monitor non-deterministic accesses to occur in OpenMP programs so that this tool can verify the existence of races even if it is used any race detection protocol which can apply to programs with critical section. To prove empirically this tool, we have experimented using a set of benchmark programs such as synthetic programs that involve non-deterministic accesses, OpenMP Microbenchmark, NAS Parallel Benchmark, and OpenMP application programs.

Keywords : Parallel Programs, Races, On-The-Fly Race Detection, Internal Non-Determinism, Apparent Races, Potential Races, Synthetic Programs, Microbenchmark, NAS Parallel Benchmark

1. 서 론

병렬 프로그램은 병행하게 수행하는 스레드간의 동기화가 필수적이다. 하지만 잘못된 동기화의 사용으로 인해 경합이라는 치명적인 오류가 발생할 수 있다. 경합[1]은 병행하게 수행하는 스레드가 적절한 동기화 없이 적어도 하나 이상의 쓰기 사건으로 공유메모리에 접근할 때 발생하며 이러한 경

※ 본 연구는 지식경제부 및 정보통신산업진흥원의 대학 IT연구센터 지원사업의 연구결과로 수행되었음 (NIPA-2009-(C1090-0904-0001)) 그리고 정부(과학기술부)의 재원으로 한국과학재단의 지원을 받아 수행된 연구임 (R0A-2007-000-10038-0)
† 정 회 원 : 한국과학기술원 연구교수
†† 정 회 원 : ㈜글로벌스텐다드테크놀로지 SE사업본부
††† 종신회원 : 경상대학교 정보과학과 교수
논문접수 : 2009년 11월 2일
수 정 일 : 1차 2010년 1월 5일
심사완료 : 2009년 1월 7일

합은 프로그래머가 의도하지 않은 비결정적인 수행 결과를 초래하므로 반드시 탐지 되어야 하며 디버깅의 대상이 된다.

이러한 경합을 프로그램의 수행 중에 탐지하는 것이 가장 현실적인 방법으로 알려져 있다. 이러한 수행 중 경합 탐지 기법[2-7]을 이용하여 임계구역을 가진 프로그램에서 적어도 하나의 경합을 탐지하기 위해서는 SISE (Single Input, Single Execution) 속성[2, 8]을 만족하여야 한다. 하지만 이 속성을 만족하지 못한다면 경합검증을 위해서 각 임계구역마다 $N!$ 만큼의 프로그램 수행양상이 필요하다. 여기서 SISE는 주어진 입력에 대하여 한번의 수행 양상에 대해서만 경합을 탐지하는 것을 의미하고, N은 프로그램 수행 중에 발생할 수 있는 스레드의 최대 병렬성을 의미한다. 경합을 탐지하는 기존의 도구들[2, 4, 8-11]은 수행 중에 발생한 접근 사건정보만을 이용하여 경합을 탐지하므로 임계구역에 의한 내부적 비결정성이 존재하는 SISE를 만족하지 못하는 프로그램에 대해서는 경합 검증을 하지 못한다. 그러므로 기존의 도구들은 경합 검증 능력을 위해서는 $N!$ 만큼의 프로그램 수행이 필요하다. 따라서 본 논문에서는 내부적 비결정성을 가진 프로그램에 존재하는 비결정적 접근사건을 정적으로 분석하고, 이 정보를 이용하여 프로그램 수행 중에 경합을 탐지함으로써 내부적 비결정성이 존재하는 프로그램에서도 주어진 입력에 대하여 한 번의 수행만으로도 잠재되어 있는 경합까지 탐지할 수 있는 전처리기를 제안한다.

2절에서는 병렬 프로그램에서 발생할 수 있는 경합과 기존 도구의 문제점을 소개하고, 3절에서는 OpenMP 프로그램을 고려한 프로그램 슬라이싱을 이용하여 잠재적 경합의 원인이 되는 비결정적 접근사건을 탐지하여 프로그램의 결정적 감시가 가능하게 하는 도구에 대해서 설명한다. 그리고 4절에서는 비결정성이 포함된 합성 프로그램과 공인된 벤치마크 프로그램을 이용하여 제안된 도구의 경합 검증 능력을 보인다. 마지막으로 5절에서는 결론 및 향후과제에 기술한다.

2. 연구배경

본 절에서는 OpenMP[4, 11, 12]]에 대해서 설명하고, POEG (Partial Order Execution Graph)[2, 13]을 이용하여 프로그램에서 발생하는 경합과 임계구역을 가진 병렬 프로그램에서 경합을 탐지하는 기존 도구에 대한 문제점을 기술한다.

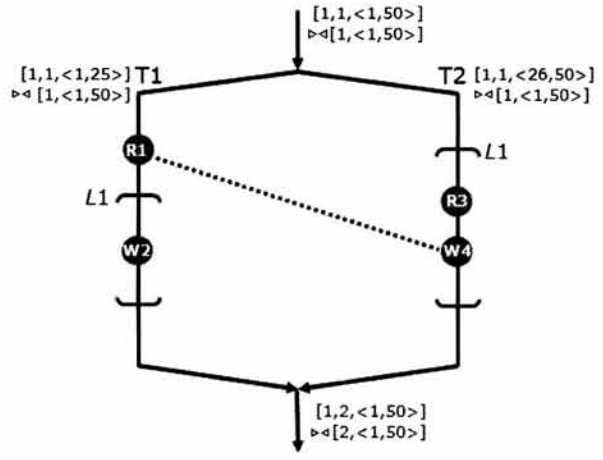
2.1 병렬프로그램에서의 경합

공유메모리 기반의 병렬 프로그램인 OpenMP는 표준 C/C++와 Fortran 77/90 언어로 공유메모리 기반의 병렬 프로그램을 작성할 수 있도록 컴파일러 디렉티브(compiler directives)와 수행시간 라이브러리(run-time libraries)를 제공한다. C/C++나 Fortran 77/90 언어로 작성된 순차 프로그램에 병렬화 디렉티브만을 삽입함으로써 쉽게 병렬 프로그램을 작성이 가능한 고수준(high-level)의 프로그래밍 환경을 지원한다. 또한 OpenMP는 'coarse-grain parallelism'의 구현을 위해 'orphan' 디렉티브 개념을 제공하고 있다. 'orphan' 디렉티브는 프로그램 코드 상에서 병렬 영역(parallel region)의 외부에 선언된 디렉티브를 말하는 것으로 병렬 프로그래밍의 확장성(scalability)을 높여줄 수 있어 효율적이다. 현재 많은 주요 하드웨어 및 소프트웨어 회사들이 일반 데스크 탑에서부터 슈퍼컴퓨터에 이르기까지 다양한 플랫폼에서 OpenMP를 이용하여 병렬 프로그램을 쉽고 유연하게 개발할 수 있도록 지원하고 있다. (그림 1)은 C로 작성 된 OpenMP 프로그램의 예이다. 5번 줄의 "#pragma omp parallel for" 디렉티브를 사용해서 스레드를 생성하여 작업을 할당하고 "shared(account, rate)"와 "private(i, temp)" 절을 사용하여 공유변수로는 *account*와 *rate*를 private 변수로는 *i*와 *temp*를 각각 지정 하였다. 그리고 9번 줄의 "#pragma omp critical (L1)" 디렉티브를 사용하여 록(lock)변수 *L1*에 의한 임계구역(critical section)을 지정하였다. 그리고 6번 줄의 for 문 내에 있는 7번 줄의 if문과 13번 줄의 else if문은 각각 서로 다른 스레드에 의해서 병행하게 수행 된다.

(그림 2)는 (그림 1)의 병렬 프로그램의 소스코드를 기반으로 작성된 POEG이다. POEG은 병렬 프로그램의 논리적 순서관계를 그래프로 나타낸 것이다. POEG에서의 정점(vertex)은 병렬 스레드의 생성이나 종료 명령에 의해 병렬 스레드의 포크(fork) 및 조인(join)되는 지점을 의미하며, 임의의 정점에서 시작하는 간선(arc)은 그 정점으로부터 수행되는 스레드의 블록을 표시한다. 이러한 POEG을 이용하여 스레드의 병행성 관계를 파악할 수 있다. POEG에서 Ⓡ 심볼과 Ⓦ 심볼은 스레드에서 발생한 공유변수 *account*에 대한 접근사건을 의미하며 ⊂ 심볼은 임계 구역의 시작과 끝을 의미한다. (그림 2)에서 T1 스레드의 임계구역이 수행된 후에 T2의 임계구역이 수행되면 *account*의 최종값은 50이고, T2 스레드의 임계구역이 수행된 후에 T1 스레드의 임

```
1.  void main() {
2.      int i, temp;
3.      int account = 100, rate = 50;
4.      ...
5.      #pragma omp parallel for shared(account,rate) private(i,temp)
6.      for(i = 0; i<2; i++) {
7.          if(i==0) {
8.              temp = account + rate;
9.              #pragma omp critical(L1)
10.             {
11.                 account = temp;
12.             }
13.         } else if (i==1) {
14.             #pragma omp critical(L1)
15.             {
16.                 temp = account - rate;
17.                 account = temp / 2;
18.             }
19.         }
20.     }
21.     ...
22. }
```

(그림 1) OpenMP 병렬프로그램



(그림 2) OpenMP 병렬프로그램의 경합

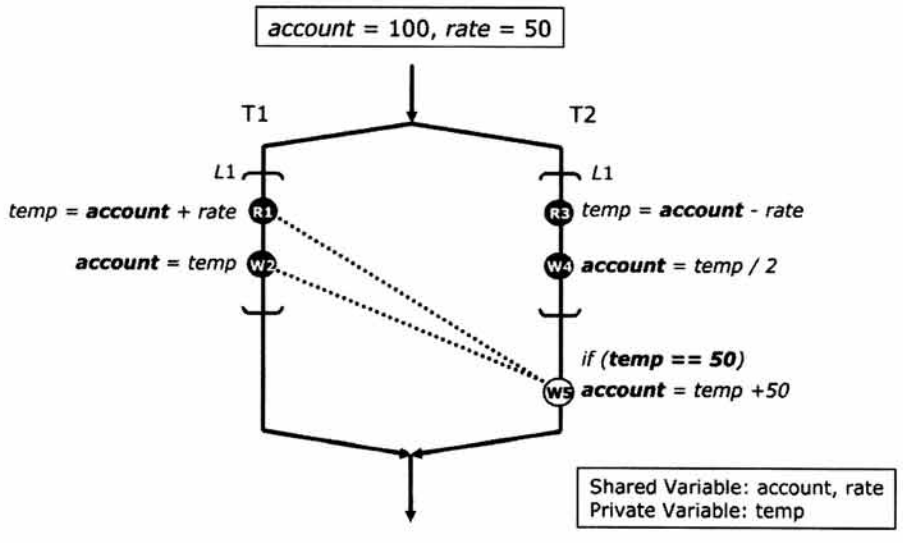
계구역이 수행되면 *account*의 최종값은 75가 된다. 하지만 프로그래머가 의도하지 않은 비결정적인 수행결과가 나올 수 있다. 예를 들어, T2 스레드의 임계구역이 먼저 록 변수 *L1*을 획득하여 *account*에 대한 읽기접근사건이 수행한 후에 다음의 쓰기접근사건이 수행하기 전에 T1 스레드의 읽기접근사건이 발생하게 되면 *account*의 최종값은 150이 된다. 이러한 발생의 원인은 T1 스레드의 읽기접근사건과 T2 스레드의 쓰기접근사건 사이에 경합이 존재하기 때문이다. 경합은 병렬로 수행하는 두 스레드가 적어도 하나 이상의 쓰기사건으로 적절한 동기화 없이 같은 메모리 영역을 접근할 때 발생한다. 하지만 T1 스레드의 쓰기접근사건(W2)와 T2 스레드의 쓰기접근사건(W4)은 동일안 임계구역으로 보호되어 있기 때문에 경합이 아니다.

2.2 경합탐지 기법

2.1절에서 설명한 병렬프로그램에서 발생하는 경합을 수행중에 탐지하기 위해서는 병렬로 수행하는 스레드간의 논리적 순서관계를 생성하는 레이블링 기법[2, 13-16]과 각 스레드에서 발생한 공유변수에 대한 접근사건 정보를 저장하는 접근역사(Access History)의 유지정책으로 경합을 보고하는 경합탐지 프로토콜[2, 3, 8, 13] 기법이 있다. 레이블링 기법은 각 스레드에 생성되는 유일한 값으로써, (그림 2)의 T1 스레드에 생성된 "[1,1,<1,25>]▷◁[1,<1,50>]"이며, NR 레이블링 기법[15, 16]에 의해서 생성된 값이다. 경합탐지 프로토콜은 각 스레드에서 생성된 레이블을 이용하여 수행중에 경합여부를 검사한다. (그림 2)에서 T1 스레드의 읽기접근사건인 R1과 T2 스레드의 쓰기접근사건인 W4는 경합탐지 프로토콜[2]에서 의해서 경합으로 탐지된다.

그러나, (그림 3)은 SISE 속성을 만족하지 않는 프로그램이다. 왜냐하면 각 스레드에 존재하는 임계구역의 수행 양상에 따라 수행되는 명령어 문장들이 달라질 수 있기 때문이다. T2 스레드의 임계구역이 수행한 뒤 T1 스레드의 임계구역이 수행하게 되면 T2 스레드의 private 변수인 *temp*의 값은 50이 된다. 따라서 if문의 조건을 만족하게 되어 공유변수 *account*에 대한 쓰기사건 W5가 발생하게 된다. 하지만 T1 스레드의 임계구역이 먼저 수행한 뒤 T2 스레드의 임계 구역이 수행하게 되면 T2 스레드의 *temp* 값은 100이 되기 때문에 if문에 내포 되어 있는 쓰기사건 W5는 발생 하지 않는다. 즉 임계구역의 수행순서에 의한 명령어 문장들의 수행양상이 달라지는 내부적 비결정성이 존재한다. 따라서 공유변수 *account*에 대한 쓰기사건 W5는 비결정적 접근사건이다.

비결정적 접근사건을 가진 프로그램에서 기존의 기법[2, 4, 8-10]을 이용하여 경합을 탐지할 경우 T1 스레드의 임계구역이 먼저 수행 되었다면 W5에 대한 접근 정보가 없기 때문에 (그림 3)에 보인 것처럼 {R1-W5}와 {W2-W5} 사이의 경합은 보고하지 못한다. 왜냐하면 (그림 3)은 내부적 비결정성이 존재하는 프로그램이기 때문이다. 따라서 내부적 비결정성을 가진 프로그램에서 기존의 기법은 경합을 탐지



(그림 3) 내부적 비결정성에 의한 잠재적 경합

하지 못한다. 그러나 이러한 프로그램에서는 비결정적 접근사건에 의한 잠재적 경합이 존재할 수 있으므로 잠재적 경합을 탐지하기 위해서는 $N!$ 만큼의 프로그램 수행이 필요하다. 예를 들어, (그림 3)에서 T1 스레드의 임계구역이 수행한 후에 T2 스레드의 임계구역이 수행했을 경우와 T2 스레드의 임계구역이 수행한 후에 T1 스레드의 임계구역이 수행했을 경우의 수행양상을 모두 고려하여야만 경합의 검증이 가능하다.

3. 잠재적 경합 탐지: 전처리기

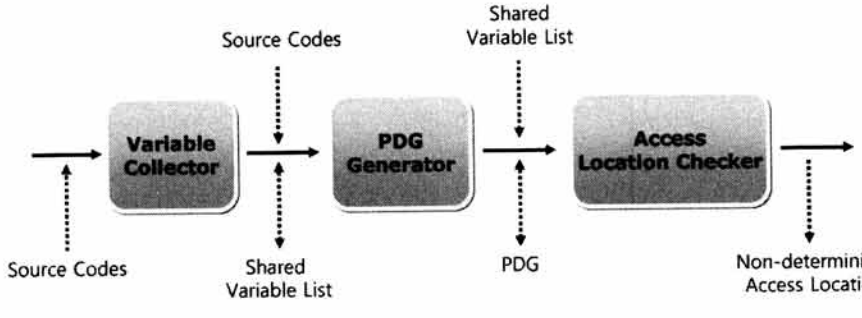
본 절에서는 OpenMP 병렬 프로그램의 특성을 고려한 PDG(Program Dependence Graph)[17]을 생성하고, 프로그램 슬라이싱 기법[2]을 사용하여 잠재적 경합의 원인이 되는 비결정적 접근사건을 탐지하는 프로그램의 결정적 감시를 가능하게 하는 도구에 대해서 설명한다.

3.1 내부적 비결정성 분석

잠재적 경합을 탐지하기 위해서 OpenMP 병렬 프로그램의 특성인 병렬화 디렉티브와 동기화 디렉티브를 고려한 OpenMP-PDG를 생성하고, 생성된 PDG를 이용하여 프로그램 슬라이싱을 한다. 이 과정에서 대상 프로그램의 수행전에 코드를 분석하여 내부적 비결정성을 유발하는 접근사건의 위치를 탐지한다. 이러한 탐지를 할 수 있는 분석도구는 (그림 4)와 같다. Variable Collector는 OpenMP 프로그램에서 공유될 가능성이 있는 전역변수나 지역변수들에 대한 정보를 수집하는 모듈이다. 전역변수의 경우에는 데이터 타입과 변수명에 대한 정보, 공유가 가능한 변수인지를 수집한다. 그리고 지역변수의 경우에는 다른 함수내에서 동일한 이름으로 선언된 지역변수가 존재할 수 있으므로 변수가 선언된 함수명과 그 함수가 존재하는 파일명에 대한 정보를 추가적으로 수집한다. Variable Collector 모듈은 OpenMP-PDG의 효율적 생성을 위해서 경합탐지의 대상이 되는 변수들만을 선택하기 위한 것이다. 그리고 이들 변수는 리스트 형태로 저장된다.

이렇게 수집된 변수 리스트와 원시코드를 이용하여 PDG Generator는 OpenMP 병렬 프로그램을 고려하여 프로그램의 명령어 문장이나 조건식을 노드로 표시하며, 노드간의 데이터 의존성과 제어 의존성을 생성하는 모듈이다. 이러한 PDG를 생성하기 위한 첫 번째 단계는 프로그램에서 각 문장별로 사용된 변수에 대한 정보를 수집하고, 그 다음 단계에서는 프로그램의 문장을 기준으로 자신의 수행에 영향을 미치는 *Predicate*에 대한 문장번호를 구하고, 자기 이후에 올 수 있는 문장들, 즉 *Successor*를 구한다. 마지막으로 변수의 사용 정보, *Predicate* 정보, 그리고 *Successor* 정보를 이용하여 PDG를 생성한다. 변수에 대한 사용 정보를 수집하기 위해서 각 문장 또는 *Predicate* 단위로 노드를 생성한다. 그리고 각 노드별로 할당자인 "="을 기준으로 좌변에 있는 것을 정의(definition) 리스트에 저장하며 우변에 있는 것을 참고(reference) 리스트에 저장한다. 또한 *Predicate*를 결정하기 위해서 각 함수와 조건문 그리고 반복문에 대하여 *Life Scope*를 결정한다. 또한, 임계구역 내에서 발생한 접근사건들은 록 변수에 대한 정보를 가지고 있다. 임의의 문장이 조건문과 반복문에 내포되어 있을 경우에 그 조건문 혹은 반복문의 문장 번호를 자신의 *Predicate*로 지정한다. 조건문 또는 반복문이 중첩되어 있는 경우에는 가장 가까운 조건문을 자신의 *Predicate*로 한다. 그렇지 않은 경우 그 문장이 속해 있는 함수의 시작을 *Predicate*로 가진다. *Successor*의 경우에는 자기 이후에 수행될 가능성이 있는 모든 문장들을 함수 단위로 수집한다. OpenMP 프로그램에서 병렬로 수행되는 스레드의 경우에 자기와 병렬로 수행되는 스레드 또한 *Successor*로 가져야 한다. 함수 단위로 *Successor*를 구하게 되면 전역변수에 대한 *Successor*가 정확하지 않다. 따라서 본 논문에서는 main함수의 시작에서부터 종료될 때까지의 프로그램 함수 호출 관계까지 고려하여 프로그램이 수행될 수 있는 수행 순서에 대한 정보를 수집한다. <표 1>은 최종적으로 생성된 PDG를 나타낸 것이다. Data Dependence를 결정하는 방법은 변수를 정의하는 노드에 대하여 자신의 *Successor* 중에서 변수를 재정의 하지 않고 사용한 노드들을 연결리스트로 연결하고, Control Dependence를 결정하는 방법은 자신을 *Predicate*로 하는 모든 노드를 수집하여 연결 리스트로 연결한다. 그리고 Global Data Dependence는 전역변수에 대한 Data Dependence 리스트를 수집한 것이다.

Access Location Checker는 수집된 변수의 정보와 생성된 OpenMP-PDG를 이용하여 Dinning이 제안한 Forward-Slice 알고리즘[10]을 이용하여 비결정적 접근사건의 위치를



(그림 4) Program Analyzer: 내부적 비결정성 분석모듈

〈표 1〉 OpenMP 특성을 고려한 PDG

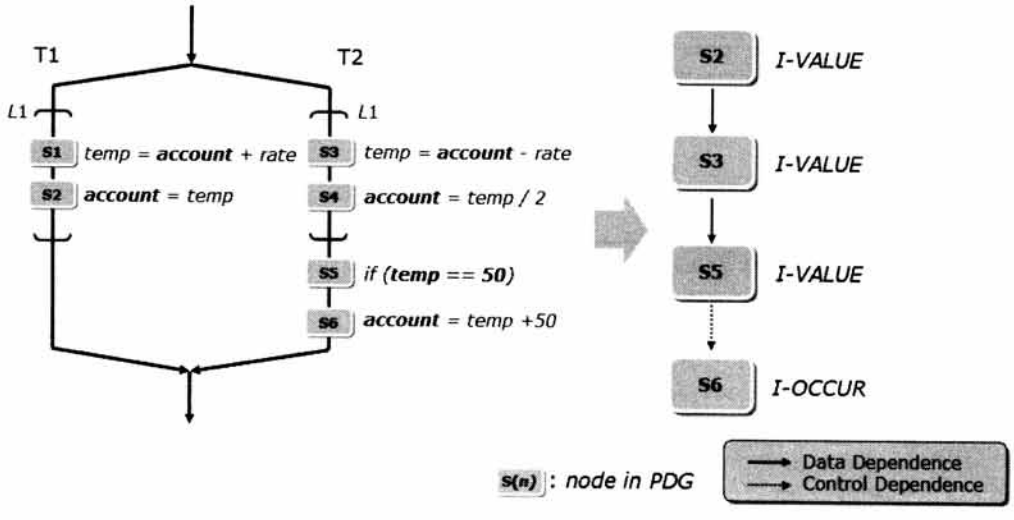
No	Index	Data Dep.	Control Dep.	Global Data Dep.
1	void main()	NULL	2,3,4,5,6,7,8...	NULL
...				
5	for(i=0; i<2; i++) {	6,12	6,12	NULL
6	if(i==0) {	NULL	7,8,9,10,11	NULL
7	#pragma omp critical(L1)	NULL	NULL	NULL
8	{	NULL	NULL	NULL
9	temp = account + rate;	9,10	NULL	NULL
10	account = temp;	15	NULL	NULL
11	}	NULL	NULL	NULL
...				

검사하는 모듈이다. (그림 5)은 PDG를 이용하여 Forward Slicing 알고리즘으로 비결정적인 접근사건의 위치를 파악하는 예를 보인 것이다. T1 스레드의 S1 노드에서 변수 *temp*의 값은 S2 노드의 변수 *account*의 값에 영향을 미친다. 이때 이 두 노드 사이에는 데이터 의존성이 존재한다. 그리고 T2 스레드의 S5 노드에서 조건문에 따라 S6노드의 수행 여부가 결정된다. 이때 이 두 노드 사이에는 제어 의존성이 존재한다. 임계구역 내에서 발생한 공유변수에 대한 쓰기접근사건은 내부적 비결정성의 발생 원인이 된다. S2 노드와 S4 노드의 공유변수 *account*는 T1 스레드와 T2 스레드에 존재하는 임계구역의 수행 순서에 따라 비결정적인 값을 가진다. 그리고 S6 노드의 경우 S5 노드에 의해 비결정적으로 수행되며 S6 노드의 공유변수 *account*에 대한 접근사건은 비결정적으로 발생한다. 따라서 생성된 PDG를 이용하여 Dinning이 제안한 Forward Slice 알고리즘을 수행하면 S1 노드부터 S5 노드까지 I-VALUE로 설정이 되고, S6 노드는 I-OCCUR로 설정된다. 여기서 S6 노드에 직접적으로 영향을 주는 I-VALUE 노드들을 (그림 5)과 같이 S2, S3, 그리고 S5이다. I-OCCUR로 설정된 S6 노드의 *account*에 대한 접근사건 정보를 PDG에 추가하여 유지한다. I-VALUE는 임계구역 내에서 공유변수나 지역변수에 대한 쓰기접근사건이 발생한 것을 의미하고, I-OCCUR는 명령어 문장이 비결정적으로 수행되거나 변수의 참조가 달라지는 것을 의미한다.

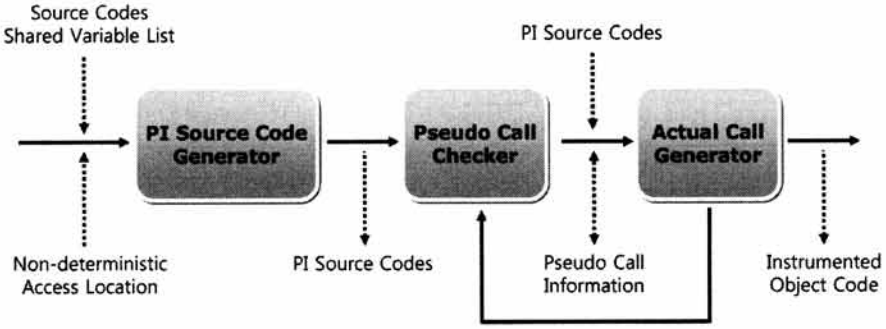
3.2 감시코드 생성

내부적 비결정성을 수행중에 탐지하기 위해서는 원시프로그램이나 바이너리 코드에 감시코드를 추가해야 된다. 본 연구에서는 원시프로그램에 감시코드를 추가하는 방법을 사용하며 경합탐지 엔진은 RaceStand[18, 19]에 탑재되어 있는 엔진을 사용한다. (그림 6)은 (그림 4)에서 수집된 변수 리스트와 비결정적 접근사건에 대한 정보를 입력으로 받아서 감시코드가 추가된 변형된 코드를 생성하는 모듈들을 보인 것이다. PI Source Code Generator는 원시 프로그램을 분석하여 중간코드를 생성한다. 중간코드에는 스레드의 생성 및 합류, 임계구역의 시작과 끝, 공유변수에 대한 읽기/쓰기 접근사건, 그리고 내부적 비결정성 접근사건에 대한 의사코드(Pseudo Code)가 포함되어 있다. Pseudo Call Checker는 소스코드를 한 라인씩 읽어와 의사코드 여부를 검사한다. 의사코드가 발견되면 Actual Call Generator가 실제 경합탐지 엔진들을 호출할 수 있는 라이브러리 형태로 변환한다. <표 1>은 OpenMP와 같은 병렬 프로그램의 경합탐지를 위한 의사코드를 정한 것이며, 그 의사코드에 해당하는 실제 경합탐지 라이브러리명을 보인 것이다. 여기서 경합탐지 엔진들은 스레드의 생성과 합류지점에는 레이블링 엔진, 공유변수에 대한 읽기/쓰기접근사건 및 비결정적 접근사건 지점에는 탐지프로토콜 엔진, 그리고 임계구역의 시작과 끝 지점에는 록커버 엔진이다.

이러한 감시코드를 삽입할 때 정적 분석기법을 통하여 분석한 비결정적 접근사건의 위치정보를 이용하여 비결정적 접근사건에 대한 감시코드를 임계구역의 수행순서와 관계없



(그림 5) PDG를 이용한 비결정적 접근사건



(그림 6) Program Instrumentor: 감시코드 생성모듈

이 항상 동일하게 수행하도록 삽입함으로써 수행 중 경합 탐지 프로토콜에 의하여 경합의 검증이 가능하도록 한다. (그림 7)은 OpenMP 병렬 프로그램에 레이블링, 탐지프로토콜, 록커버 엔진이 추가된 것을 보인 것이다. 3번 줄에 있는 NR_Fork은 스레드의 생성을 의미하는 레이블링 엔진이고, 7번 줄에 있는 NR_Add_Lock는 임계구역을 의미하는 록커버 엔진이다. 그리고 8번, 9번 줄에 있는 CSR_Checker는 읽기접근사건 탐지프로토콜 엔진이고 17번 줄에 있는 D_W_Checker는 비결정적 접근사건 탐지프로토콜 엔진이다. D_W_Checker가 17번 줄에 삽입되는 이유는 15번 줄이 수행되지 않더라도 접근사건을 감시하여 잠재적 경합을 탐지하기 위해서이다.

```
1. #progma omp parallel for shared (account,rate)
2. for(i=0; i<2; I++) {
3. label_main1 = (Label_NR *)NR_Fork(label_main0,i,1,0,2,1,7)
4. if(i==0) {
5.     #pragma omp critical(L1)
6.     {
7.         NR_Add_Lock(label_main1, "L1", 10);
8.         CSR_Checker(label_main1, SV0, 12);
9.         CSR_Checker(label_main1, SV1, 12);
10.          temp = account + rate;
11.          ...
12. } else if(i==1) {
13.        ...
14.       if(temp == 50) {
15.           account = temp + 50;
16.       }
17.       D_W_Checker(label_main1, SV0, 24);
18. }}
```

(그림 7) OpenMP 프로그램에 대한 변형된 원시 프로그램

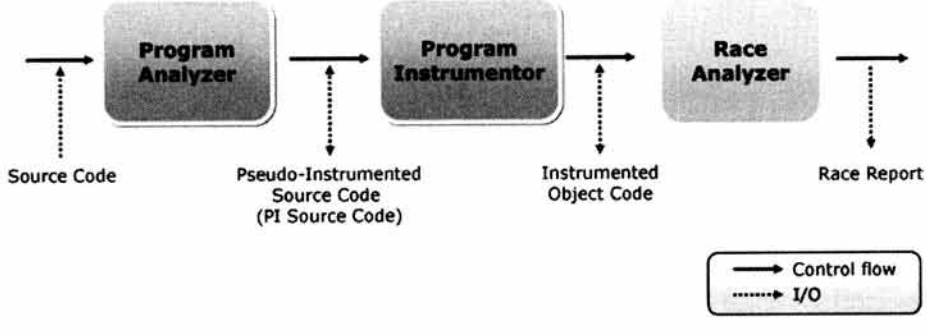
4. 도구의 구현

본 절에서는 제안된 도구의 전체구조와 구현환경에 대해서 기술하고, 비결정적 접근사건의 위치와 경합검증여부를 실험한 결과에 대해서 설명한다.

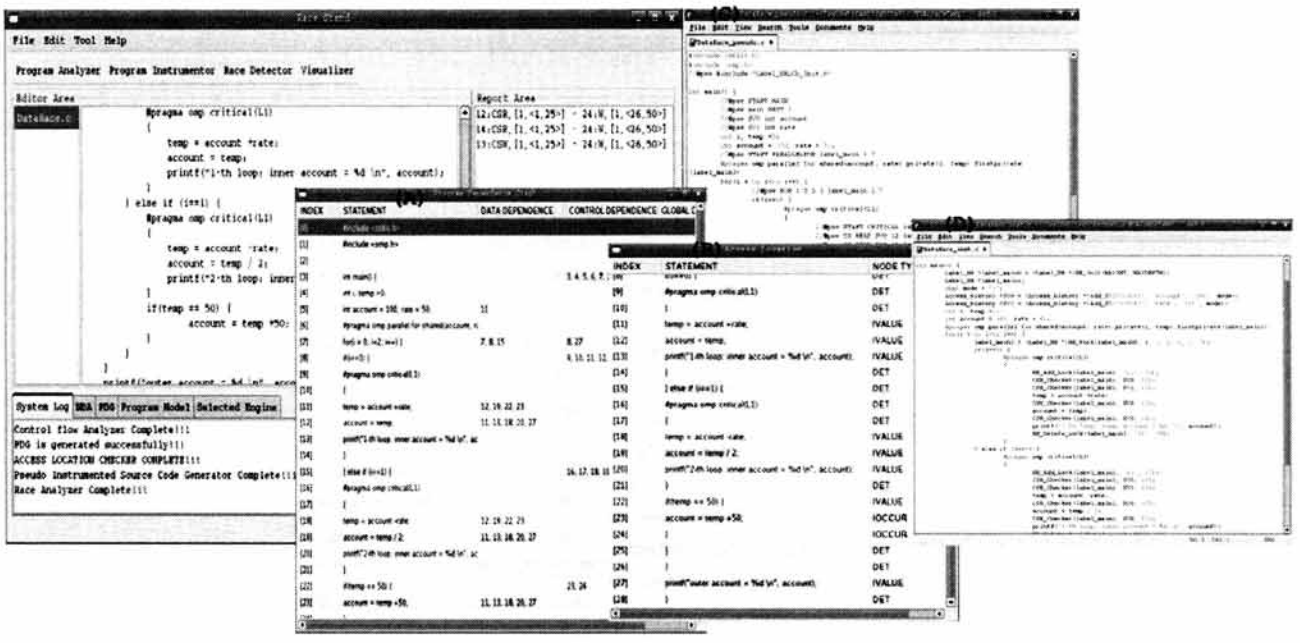
4.1 전체구조

(그림 8)은 정적분석 기법과 동적분석 기법을 혼합한 하이브리드 기법으로 프로그램내에 존재하는 잠재적 경합을 탐지할 수 있는 도구의 전체 구조를 나타낸 것이다. OpenMP 병렬 프로그램과 같은 소스코드가 주어지게 되면 Program Analyzer에 의해 비결정적 접근사건에 대한 존재여부를 파악하게 되고, 파악된 정보를 이용하여 변형된 중간코드가 생성된다. 중간 코드가 삽입된 코드는 다시 Program Instrumentor에 의해 실제 경합 탐지엔진을 호출하기 위한 감시코드로 대체되어 컴파일 된다. 컴파일 된 변형된 오브젝트 코드는 Race Analyzer에 의해 수행중에 경합을 탐지한다. Race Analyzer는 3절에서 언급한 레이블링, 탐지프로토콜, 그리고 록커버 엔진에 해당하는 부분이다.

제안된 도구는 Java Development Kit 1.5버전의 Java 언



(그림 8) 제안된 도구의 전체 구조



(그림 9) 제안된 도구의 실행화면

어로 구현하였으며, 원시코드의 변수를 효과적으로 수집하기 위해서 Java Compiler Compiler 4.0으로 구현하였으며 Eclipse환경에서 개발하였다. (그림 9)은 제안된 도구의 개발된 인터페이스 화면이다. Program Analyzer, Program Instrumemtor, 그리고 Race Detector를 실행하기 위한 Toolbar와 소스코드를 보기 위한 창이 있으며, 경합을 보고하기 위한 창과 시스템 로그를 보여주는 창 있다. 인터페이스 화면 위에 실행된 창은 (A)는 Program Dependence Graph 창으로써 Data Dependency, Control Dependency, 그리고 Global Data Dependency가 자동적으로 생성된 것을 보인 것이다. (B)는 PDG를 이용하여 [23]번에 비결정적 접근사건들을 탐지한 결과를 보인 것이다. (C)와 (D)는 수행중 경합탐지를 위해서 의사코드와 실제 감시코드가 삽입된 결과를 보인 것이다. (D)코드를 수행하면 인터페이스에 "Report Area"창에 경합을 보고하게 된다.

4.2 실험결과

본 도구의 구현된 모듈들을 검증하기 위해서 비결정적 접근사건들이 포함된 합성 프로그램, OpenMP.org 포럼에서 공식적으로 사용하고 있는 벤치마크 프로그램인 EPCC의 OpenMP Microbenchmark[20] 프로그램, 그리고 OpenMP 프로그램으로 변형한 응용 프로그램으로 실험한다. 합성 프로그램은 OpenMP Microbenchmark와 NAS Benchmark[21]를 분석한 결과를 이용하여 내포 병렬성과 임계구역 기반의 동기화를 이용하여 작성하였다. 작성된 합성 프로그램은 NNDA, NNNA, NEDA, NENA 네 가지 형태를 가지고 있다. 이 네 가지 프로그램 if, for, while을 기준으로 분리되어 작성되었다. <표 2>는 실험을 위해서 작성된 합성 프로그램의 모든 형태를 보인 것이다. 여기서 NNDA-x는 내포병렬

〈표 2〉 실험을 위해서 작성하기 위한 합성 프로그램 종류

Nest	Access	Control Statement	Code Name
Non-Nested Models	Deterministic Accesses	if	NNDA-IF
		for	NNDA-FOR
		while	NNDA-WHILE
	Non-deterministic Accesses	if	NNNA-IF
		for	NNNA-FOR
		while	NNNA-WHILE
Nested Models	Deterministic Accesses	if	NEDA-IF
		for	NEDA-FOR
		while	NEDA-WHILE
	Non-deterministic Accesses	if	NENA-IF
		for	NENA-FOR
		while	NENA-WHILE

성이 없고 비결정성 접근사건들이 없는 프로그램, NNNA-x는 내포병렬성이 없고 비결정성이 존재하는 프로그램, NEDA-x는 내포병렬성이 있고 비결정성 접근사건이 없는 프로그램, NEDA-x는 내포병렬성이 있고 비결정성 접근사건이 존재하는 프로그램이다. 그리고 OpenMP Microbenchmark 프로그램은 OpenMP 디렉티브와 런타임 라이브리를 이용하여 배열 오퍼레이션, 루프 스케줄링, 동기화에 대한 오버헤드를 측정하는 OpenMP 포럼에서 공인한 프로그램이고 NAS Benchmark 프로그램은 Omni Compiler Project에서 제공하는 프로그램으로써, NASA에서 제공하는 NPB(NAS Parallel Benchmarks)를 OpenMP C 버전으로 변경한 것이다. 또한 OpenMP 포럼에서 제공하는 Jacobi 프로그램은 Helmholtz 방

〈표 3〉 벤치마크프로그램과 합성프로그램에 대한 비결정적 접근사건 실험결과

Program		Shared Variables	Dependence Nodes			Non-deterministic Accesses	
			Global	Local	Control	I-VALUE	I-OCCUR
합성 프로그램	NNDA	2	0	8	4	5	0
	NNNA	2	0	7	4	8	2
	NEDA	2	0	8	4	5	0
	NENA	2	0	9	6	7	2
EPCC 벤치마크	Arraybench	34	6	37	14	0	0
	Schedbench	26	11	45	19	0	0
	Syncbench	52	16	67	26	0	0
NAS 벤치마크	EP	24	20	10	6	0	0
	FT	62	44	34	10	0	0
	IS	41	27	15	4	0	0
	MG	15	29	29	6	0	0
응용 프로그램	Jacobi	24	1	3	2	0	0
	MD	4	0	6	2	0	0

정식을 Jacobi 방식의 반복문을 사용하여 해결하는 OpenMP 프로그램이며 MD 프로그램은 Velocity Verlet 알고리즘을 사용하여 간단한 분자 역학 운동 구현한 OpenMP 프로그램이다.

<표 3>은 4.1절에 구현된 도구를 이용하여 합성프로그램, 벤치마크 프로그램, 응용 프로그램에 대한 비결정적 접근사건에 대한 실험 결과표이다. 표에서 Shared Variables은 각 프로그램에서 수행중에 공유변수가 될 가능성을 있는 변수의 수를 나타낸 것이며, Dependence Nodes는 Data Dependence나 Control Dependence를 가진 노드의 수를 의미하며, 다시 Data Dependence의 경우는 Global 변수에 의한 Data Dependence와 Local 변수에 의한 Data Dependence로 구별하였다. Non-Deterministic Accesses는 비결정적인 값을 가지는 노드(I-VALUE)와 비결정적으로 발생할 수 있는 노드(I-OCCUR)의 수를 의미한다.

<표 3>의 실험결과에서 보듯이 합성 프로그램 중에서 NNNA와 NENA 모델에서 잠재적 경합이 존재한다는 것을 정확하게 탐지하였다. 이것으로 인해 본 논문에서 제안된 도구의 모듈에 대해서 실험적 증명을 하였다. 그리고 제안한 도구를 이용하여 일반적인 응용에서도 잠재적 경합의 존재여부를 확인하기 위해서 공인된 벤치마크 프로그램과 응용 프로그램에서 실험을 하였다. 그 결과, 어떠한 프로그램에서도 비결정적 접근사건들이 발생하지 않았으며 경합도 발생하지 않았다.

5. 결론 및 향후 과제

논문에서 제안하는 도구는 비결정적 접근사건의 위치를 찾기 위하여 프로그램의 제어흐름과 자료흐름 정보를 가진 Program Dependence Graph와 슬라이싱 알고리즘을 사용하여 정적으로 분석한다. 그리고 분석된 비결정적 접근사건의 위치 정보를 이용하여 수행 중 경합 탐지엔진으로 임계구역의 수행 양상과 관계없이 감시 되도록 감시코드를 삽입하여 경합을 탐지함으로써 잠재적으로 존재할 수 있는 경합까지 탐지하였다. 제안된 도구는 현재 OpenMP 디렉티브 기반의 프로그램에만 적용할 수 있지만 향후에 런타임 라이브러리 기반의 프로그램에서도 잠재적 경합을 탐지할 수 있도록 할 것이다.

참 고 문 헌

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

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

[3] O'Callahan, R. and J. Choi, "Hybrid Dynamic Data Race Detection," *Proc. of ACM SIGPALN Symp. on Principle and Practice of Parallel Programming* (PPoPP), San Diego, California, ACM, June, 2003.

[4] Petersen, P., and S. Shah, "OpenMP Support in the Intel Thread Checker," *Int'l Workshop on OpenMP Applications and Tools*, 1-12, June, 2003.

[5] Praun, C., T. R. Gross, "Object Race Detection," *Proc. of the 16th ACM SIGPLAN Conference on Object Oriented Programming, Systems, Languages, and Applications*, pp. 70-82, ACM, October, 2001.

[6] Stefan S., M. Burrows, G. Nelson, P. Sobalvarro, T. Anderson, "Eraser: a dynamic data race detector for multithreaded programs," *ACM Transactions on Computer Systems* (TOCS), pp.391-411, ACM, Nov., 1997.

[7] Yu, Y., T. Rodeheffer, W. Chen, "Race Track: Efficient Detection of Data Race Conditions via Adaptive Tracking," *Proc. of the twentieth ACM Symposium on Operating Systems Principles*, pp.221-234, ACM, Oct., 2005.

[8] Cheng, G. I., M. Feng, C. E. Leiserson, K. H. Randall, A. F. Stark "Detecting Data Races in Cilk Programs that Use Locks," *Proc. of the Tenth Annual ACM Symposium on Parallel Algorithms and Architectures* (SPAA), pp.298-309, ACM, June, 1998.

[9] Kim, Y., D. Kim, and Y. Jun, "An Empirical Analysis of Intel Thread Checker for Detecting Races in OpenMP Programs," *Int'l Conf. on Computer and Information Science* (ICIS), pp.409-414, IEEE, Portland, USA, May, 2008.

[10] SUN Microsystems Inc., *Sun Studio12: Thread Analyzer User's Guide*, 2007.

[11] Terboven, C., "Comparing Intel Thread Checker and Sun Thread Analyzer," *Mini-symposium on Scalability and Usability of HPC Programming Tools, In Parallel Computing* (PARCO), September, 2007.

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

[13] Dinning, A., 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.

[14] Audenaert, K., "Clock Trees: Logical Clocks for Programs with Nested Parallelism," Tr. on Software Engineering, 23(10): 646-658, IEEE, Oct., 1997.

[15] 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., 193.

[16] Park, S., M. Park, and Y. Jun, "A Comparison of Scalable Labeling Schemes for Detecting Races in OpenMP Programs," *Proc. of the Int'l Workshop on OpenMP Applications and Tools*, Purdue Univ., West Lafayette, Indiana, Lecture Notes in Computer Science, 2104: 68-80, Springer-Verlag, July, 2001.

[17] Ferrante, J., T. J. Watson, J. Ottenstein, J. D. Warren, "The Program Dependence Graph and Its Use in Optimization," *ACM Transactions on Programming Languages and Systems* (TOPLAS), pp.319-349, ACM, July, 1987.

[18] Kim, Y., M. Kang, O. Ha, and Y. Jun, "Effcient Race Verfication for Debugging Programs with OpenMP Directives," *Int'l Conf. on Parallel Computing Technologies* (PaCT), Pereslavl-Zalessky, Russia, Sept. 2007, Lecture Notes in Computer Science, 4671: 230-239, Springer, Aug., 2007.

[19] 김영주, 전용기, "OpenMP 디렉티브 프로그램을 위한 자료경합 검증도구," 정보과학회논문지: 시스템 및 이론, 34(9):395-406, 한국정보과학회, 2007.9.

[20] Bull, J. M., "Measuring Synchronisation and Scheduling Overheads in OpenMP," *Proc. of European Workshop on OpenMP* (EWOMP), pp.99-105, Sept. 1999.

[21] Jin, H., M. Frumkin, and J. Yan, "The OpenMP Implementation of NAS Parallel Benchmarks and its Performance," NAS TR: NAS-99-011, NASA Ames Research Center, 1999.

김 영 주

e-mail : yjkim73@kaist.ac.kr

1999년 국립경상대학교 컴퓨터과학과(학사)

2001년 국립경상대학교 컴퓨터과학과(공학석사)

2007년 국립경상대학교 컴퓨터과학과(공학박사)

2007년 9월~2008년 8월 한국정보통신대학교 박사후과정

2008년 9월~2009년 2월 한국정보통신대학교 연구교수

2009년 3월~현 재 한국과학기술원 연구교수

관심분야 : 병렬/분산 컴퓨팅, 임베디드 시스템, 센서 네트워크 (시스템 소프트웨어, 경합탐지, 디버깅, 시각화)

정 민 섭

e-mail : msjung@gst-in.com

2003년 국립진주산업대학교 컴퓨터공학과(학사)

2007년 국립경상대학교 컴퓨터과학과(공학석사)

2008년 1월~2009년 12월 ㈜피더스테크 기술연구소

2009년 12월~현 재 ㈜글로벌스텐다드테크놀로지 SE사업본부

관심분야 : 병렬/분산 컴퓨팅, 시스템 소프트웨어

전 용 기

e-mail : jun@gnu.ac.kr

1980년 경북대학교 컴퓨터공학과(학사)

1982년 서울대학교 컴퓨터과학과(공학석사)

1993년 서울대학교 컴퓨터과학과(공학박사)

1995년 3월~1996년 8월 Univ. of California, Santa Cruz 연구원

1982년 2월~1985년 3월 한국전자통신연구원(ETRI) 연구원

1985년 3월~현 재 경상대학교 정보과학과 교수

관심분야 : 분산병렬처리, 내장형시스템, 시스템소프트웨어