OpenMP 프로그램을 위한 디버깅 도구

김영주* · 김정시**

1. 서 론

컴퓨터가 개발된 이래로 프로세서(CPU)는 눈부신 발전을 거듭하였지만, 처리해야 되는 작업의 양(Task)은 지속적으로 대형화됨에 따라 단일 프로세서로 처리할 수 있는 능력이 한계에 도달하게 되었다. 이러한 문제를 극복하기 위한 방법이 병렬 처리이다. 병렬 처리란 하나의 작업을 처리하기 위해서 동시에 여러 개의 프로세서를 사용하여 작업의 처리 속도를 높이는 것을 말한다. 하나의 프로세서를 사용하여 작업을 수행하였을 때 소요된 시간을 t라고 할 때 p개의 프로세서를 사용하였을 때 이상적인 소요시간은 t/p일 것이다. 하지만 실제 병렬 처리에서는 프로세서 개수에 비례적으로 성능이 증가하지 않는다는 "**Amdahl's Law**"에 의해 프로그램에서 병렬로 수행되지 않는 부분으로 인하여 성능이 제한되며 성능향상의 상한선이 있다. 이러한 문제를 해결하기 위해서 주어진 프로그램에서 데이터를 병렬 처리하느냐 또는 작업을 분할하여 병렬 처리하느냐에 따라 데이터 병렬성(data parallelism)[9]과 타스크 병렬성(task parallelism)[9]으로 구분된다.

C나 Fortran 같은 언어를 이용하여 병렬 처리하는 프로그램을 구현하기 위해서는 프로세서간의 통신 등을 지원하는 병렬 프로그래밍 지원 라이브러리의 도움이 필요하다. 병렬 컴퓨터 개발 초기에서는 하드웨어 벤더들마다 서로 다른 병렬 라이브러리를 개발하여 지원을 하였지만 병렬 컴퓨터가 확산되면서 성능문제나 포팅문제의 해결로 인해 많은 병렬 지원 라이브러리가 사라지고 현재는 공유 메모리형 병렬 지원 라이브러리인 OpenMP[14]와 분산 메모리형 병렬 지원 라이브러리인 MPI[19] 만이 광범위하게 사용되고 있다.

이중에서 1997년에 산업표준으로 채택된 OpenMP는 공유메모리와 분산된 공유메모리를 기반으로 하는 다중프로세서를 위한 병렬 프로그래밍 모델이다. 이 프로그램은 실리콘 그래픽사(SGI)에 의해 개척되기 시작하여 다른 여러 컴퓨터 벤더들인 컴팩(Compaq), 인텔(Intel), 아비엠(IBM), 그리고 썬 마이크로 시스템사(SUN) 등이 참여하여 개발하였다. OpenMP는 새로운 프로그래밍 언어가 아니며, 표준 C/C++와 Fortran 77/90을 확장하는 병렬화 디렉티브(directive)와 라이브러리들의 집합 및 SPMD (Single Program Multiple Data) 수행 환경을 명시하고 있다. 여기서 디렉티브는 OpenMP를 지원하는 컴파일러에서 인식할 수 있는 API이다. 이전에 사용되었던 공유메모리 병렬 프로그래밍 모델인 POSIX 스레드와는 달리 OpenMP 프로그램 개발자가 기존

* 경상대학교 자연과학대학 컴퓨터과학과 박사과정
** 한국전자통신연구원 임베디드 소프트웨어 기술 센터 선임 연구원

원시 프로그램의 수정 없이 OpenMP에서 명시되어 있는 디렉티브를 삽입함으로써 손쉽게 병렬 프로그램화 할 수 있다. 또한 OpenMP는 coarse-grain parallel의 구현을 간단히 하는 orphan 디렉티브 개념을 제공한다. orphan 디렉티브는 어휘적으로 병렬 영역(parallel region) 외부에 선언된 디렉티브를 말하는 것으로 병렬 프로그래밍의 확장성 (scalability)을 높여준다. OpenMP에서 제공하는 디렉티브로는 크게 병렬화를 위한 디렉티브, 작업 공유를 위한 디렉티브, 데이터 환경을 위한 디렉티브 그리고 동기화를 위한 디렉티브 등이 있다. 그림 1은 순차 프로그램의 병렬화를 위해서 디렉티브를 삽입한 간단한 OpenMP 병렬 프로그램이다.

```
program compute_pi
  integer n, i
  double precision w, x, sum, pi, f, a

! Function to integrate
  f(a) = 4.d0 / (1.d0 + a*a)
  n = 1000
! Calculate the interval size
  w = 1.0d0/n
  sum = 0.0d0

!$OMP PARALLEL DO &
!$OMP& DEFAULT(PRIVATE), &
!$OMP& SHARED(w,n) &
!$OMP& REDUCTION(+: sum)
  do i = 1, n
     x = w * (i - 0.5d0)
     sum = sum + f(x)
  enddo
  pi = w * sum
  print *, 'computed pi = ', pi
  stop
end program compute_pi
```

그림 1. 간단한 OpenMP 프로그램

이러한 OpenMP 프로그램은 순차적 프로그램에서 발생하지 않는 오류로 인해서 성능저하나 잘못된 결과를 유발할 수 있다. 오류가 발생하는 근본적인 원인들은 병렬 프로그램의 수행 시에 발생하는 공유변수에 동시 접근으로 인한 메모리의 충돌과 프로그램 수행 시에 작업 스케줄링의 변화로 인한 프로그램의 비결정적 수행 때문이다. 프로그램에서 발생하는 오류를 탐지하여 올바른 수행이 되도록 그것을 적절하게 수정하거나 제거하는 것이 디버깅이다. 순차 프로그램에서 발생하는 오류는 "Single-Step"과 "Break-Point"와 같은 디버깅 방법으로 쉽게 디버깅할 수 있지만, 병렬 프로그램에서 발생하는 오류는 순차 프로그램에서 사용하는 디버깅 방법으로는 오류의 탐지조차도 어려울 수 있다. 왜냐하면, 프로그램 라인에 break-point를 설정하게 되면 여러 개의 스레드가 동일한 코드를 수행하게 되어서 임의의 스레드가 break-point를 만나게 되면 실행중인 모든 프로세스의 스레드가 수행을 중단되기 때문이다. 따라서 병렬 프로그램을 효과적으로 디버깅하지 못한다. 그림 2에서 case1과 case2

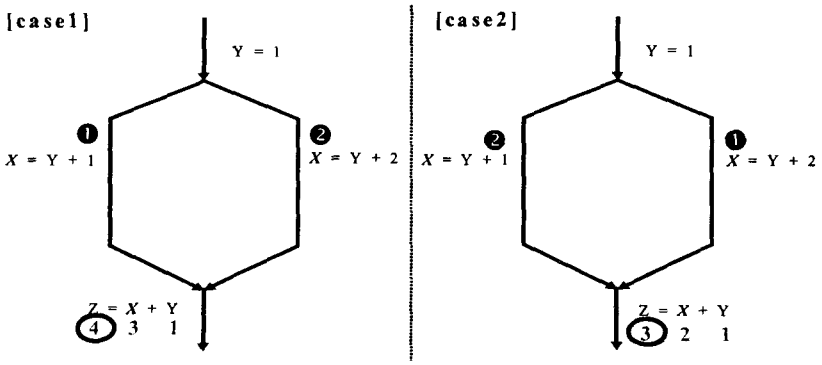


그림 2. 경합이 발생되는 양상

는 작업스케줄링에 의해서 수행 순서의 변화에 경합이 발생하는 것을 보인 것이다.

본 기사에서는 병렬 프로그램에서 오류를 탐지하는 디버깅 기법들을 소개하고, OpenMP 프로그램에서 발생하는 오류를 탐지하는 디버깅 도구들과 각 도구들의 특징, 오류 탐지방법, 탐지된 오류 종류, 그리고 탐지된 오류의 시각화에 대해서 소개하고자 한다. 마지막으로 향후 개발될 디버깅 도구에 필요한 기능들을 제시한다.

2. 병렬프로그램의 오류탐지 기법

병렬 프로그램에서 발생하는 오류[5,9,11,13]에는 Deadlocks, Races, Stalls, Live locks, Logic Errors, API violations, 그리고 Failing library routines 등이 있다. 이 중에서 본 기사에서는 Races(경합)에 대한 특징과 경합을 탐지하는 기법을 설명한다. 병렬 프로그램의 수행에서 병행 스레드들이 동일한 공유변수를 사용하고, 적절한 동기화 없이 적어도 하나의 쓰기 접근을 포함하면 경합이라는 접근 오류가 발생한다. 경합은 프로그래머의 의도와는 달리 프로그램을 비결정적(nondeterministic)으로 수행하게 되므로 병렬프로그램에서 발생할 수 있는 오류들을 효과적으로 디버깅하기 위해서는 반드시 탐지되어야 한다.

이러한 경합을 탐지하기 위한 기법들은 정적분석(static analysis)[2] 기법과 동적분석(dynamic analysis) 기법인 사후추적분석 기법(post-mortem detection)[4]과 수행중 탐지 기법(on-the-fly detection)[4,10]이 있다. 정적분석 기법과 동적분석 기법의 가장 특징적인 차이점은 오류수정을 대상으로 하는 프로그램의 수행 여부이다. 정적분석 기법은 프로그램의 원시코드만을 분석하여 대상이 되는 프로그램에서 발생할 가능성이 있는 모든 잠재적 경합들의 집합을 탐지해낸다. 반면, 동적분석 기법은 프로그램의 특정 수행을 분석하여 그 수행에서 나타날 수 있는 실제 경합들만을 탐지한다. 그리고 동적분석 기법인 사후추적분석 기법과 수행중 탐지 기법의 차이점을 보면, 사후추적분석 기법은 프로그램의 특정 수행 시에 생성한 추적화일(trace file)을 수행 후에 분석하는 기법이고, 수행중 탐지 기법은 프로그램 수행과 분석을 동시에 진행하여 경합을 탐지한다. 그리고 수행중 탐지 기법은 감시 중에 수집된 많은 정보가 수행 과정 중에 삭제될 수 있기 때문에 사후추적분석 기법보다 기억공간에 있어서 효율성을 가진다. 반면, 사후추적분석 기법보다 많은 경합을 탐지하지 못한다. 그렇지만 탐지 대상인 프로그램이 경합을 포함한다면, 각 공유변수에 관련된 적어도 하나의 경합은 반드시 탐지한다. 그러나 병렬 수행 중에 모든 공유변수에 대한 접근들을 감시하기 위한 시간과 공간에 대한 부담은 여전히 연구과제로 남아 있다.

3. OpenMP 프로그램의 디버깅 도구

본 장에서는 여러 종류의 병렬 프로그래밍 언어 중에서 OpenMP 프로그램을 대상으로 하여 오류를 디버깅하는 도구인 TotalView[1,3,6], Intel Thread Checker[7-9,11,15], ProDev WorkShop [18]에 대해 살펴본다.

3.1 TotalView의 특징

Etnus 사의 TotalView는 OpenMP 프로그램을 디버깅할 수 있도록 프로그래머에게 많은 정보를 제공하는 강력한 도구이며 프로그램, 프로세스, 스레드 레벨에서 프로그램을 제어할 수 있다. 최근의 프로그램은 giga바이트의 데이터를 포함하기도 하고 멀티 스레드로 동작하기도 한다. 이런

프로그램을 한번에 전체적인 데이터를 분석하기는 힘들고, 또한 전통적인 break-point 방법으로는 디버깅하기 힘들다. 따라서 TotalView는 다양한 플랫폼에서 유연성 있는 프로그램 제어를 위해서 다양한 디버깅 방법들을 제공하고 있다. 그림 3은 TotalView가 지원하고 있는 플랫폼을 보이고 있다. 그리고 오류를 탐지하기 위한 기법으로는 다양한 방법의 break-point, dive, 그리고 data analysis 방법 등이 있다.

Platform	*Operating System*
SGI	MIPS Irix
Compaq Alpha	Tru64 UNIX
HP	HP-UX
IBM RS/6000 and SP Power	Power AIX
Sun Sparc SunOS 5	Solaris
Compaq Alpha	Linux
Intel x86	X86 Linux

그림 3. TotalView의 지원 플랫폼

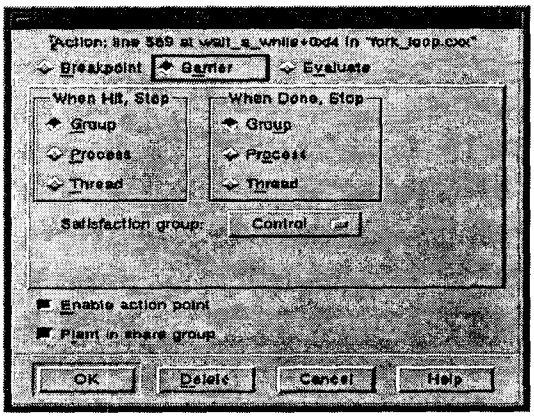


그림 4. break-point 설정 윈도우

3.1.1 TotalView 디버깅의 종류

프로그래머는 병렬 프로그램이나 복잡한 순차 프로그램을 쉽게 디버깅할 수 있도록 가능한 많은 방법을 요구한다. TotalView는 이런 요구사항을 만족한다.

① break-point

▶ *Simple break-point*: 모든 디버거에 있는 일반적인 break-point

▶ *Conditional break-point*: 프로그램의 종료 여부를 결정하기 전에 프로그램의 문법 표현을 평가하기 위한 break-point

▶ *Evaluation point*: 프로세스나 프로세스 그룹을 정지할 수 있는 Code Fragments를 프로그램의 특정부분에 삽입하여 평가하는 break-point (*Conditional break-point*를 사용하기 위해서 사용된다.)

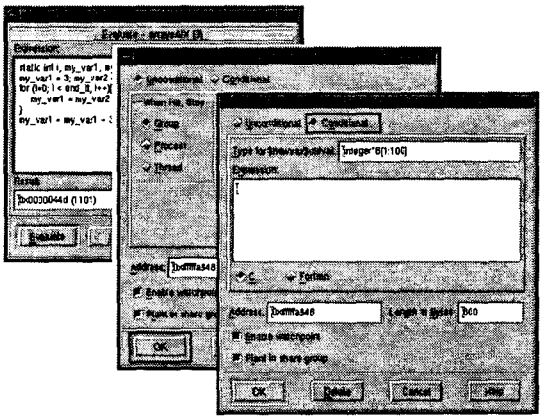


그림 5. Evaluation Point와 Watchpoint

▶*Barrier point*: 대상 프로그램의 특정 위치에 barrier 동기화를 사용하여 첫 번째 프로세스가 barrier에 도착할 때, 모든 프로세스들은 중지하고 그 지점에서 프로그램의 상태를 조사하거나 진행여부를 결정하는 break-point

▶*Data watchpoint*: 메모리의 특정부분이나 지역변수 등의 값이 변경되는 것을 감시할 수 있으며, 특히 "stray" 메모리 부분에 접근하는 문장을 찾을 시에 사용되는 break-point (조건 watchpoint와 무조건 watchpoint이 있다.)

② Dive

TotalView에서 가장 좋은 디버깅 방법이다. Dive는 임의의 객체에 관해서 보다 많은 정보를 얻기 위한 강력한 방법이다. 예를 들면, 간단하게 정보를 얻기 위한 객체에 마우스의 왼쪽 버튼을 더블클릭하면 특정변수의 값을 볼 수 있다. 만약에 함수에 대해서 Dive를 한다면 TotalView는 그 함수에 대한 원시코드를 보여준다. 그림 6, 그림 7, 그리고 그림 8은 Dive 방법으로 디버깅하는 것이다.

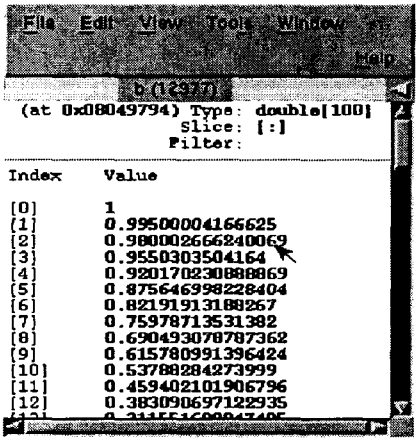


그림 6. 100개의 배열

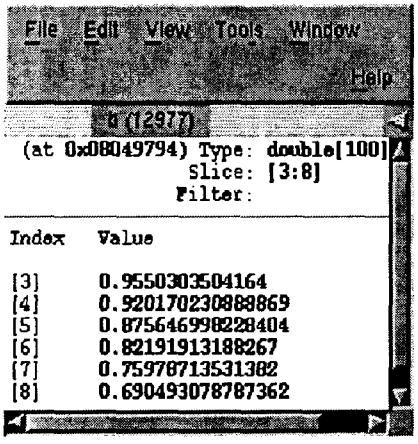


그림 7. 100개의 배열, sliced

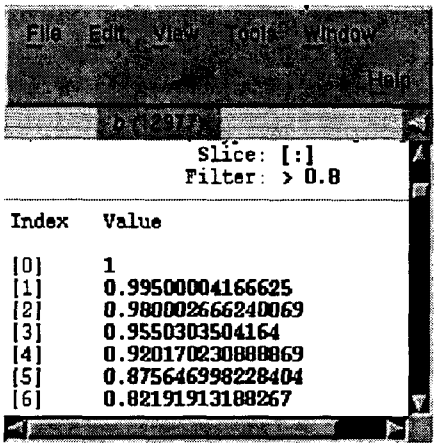


그림 8. 100개의 배열, filtered

③ Data Analysis

원시코드에 포함된 데이터를 분석하기 위한 특징들을 가지고 있다. 이들 많은 특징들은 다른 디버거에 존재하지 않는다.

▶*Dive*: 임의의 객체에 대한 정보를 출력

▶*Slicing*: 임의의 객체의 특정부분이나 특정 범위에 대해서 정보를 출력

▶*Filtering*: 임의의 조건을 만족하는 elements 들만을 출력

▶*Statistics*: 임의의 객체에 대해서 최소값, 최대값, 그리고 Checknum 등의 정보를 출력

▶*Visualization*: 디버깅한 프로그램의 전체적인 data를 그래픽적으로 보이며 잘못된 부분을 쉽게 찾을 수 있는 기능을 제공(Array Visualization 또는 Call Tree)

▶*Data watchpoint*: 메모리나 변수 등을 감시하기 위해서 사용되는 break-point

▶*Sorting*: 데이터를 오름차순이나 내림차순으로 정렬할 수 있으나 값을 변경할 수 없다.

3.1.2 OpenMP 프로그램에서의 디버깅

OpenMP Compiler는 프로그램을 병렬로 수행

시키고, OpenMP Runtime은 실행을 위한 스레드에 외부 루틴을 할당한다. 프로그래머는 루틴들이 프로그램에서 정확하게 어떻게 수행되는가에 대해서는 이해하지 못해도 상관없다. 그러나, 디버거는 프로그래머에 오류를 탐지하도록 도움을 주기 위해서 그 루틴이 어떻게 수행되는가를 알아야 한다. TotalView는 "parallel regions" 내부에 있는 코드를 디버깅한다. "parallel region" 이 수행될 시점에 프로그래머는 "parallel region" 내에 break-point를 설정해야 하고, 그 지점까지 수행한다. break-point를 설정하는 것은 프로그래머가 기대하고 있는 지점까지 수행하는 것을 보장하기 위한 안전한 방법이다. "parallel region" 이 수행될 때, 프로그래머는 단일 스레드 혹은 스레드 그룹 레벨에서 프로그램을 제어할 수 있다. 예를 들면, 그 지점까지 감시대상인 스레들이 어떻게 수행되었는가를 알 수 있다. TotalView의 stack parent token은 현재 실행중인 자식 스레드들을 생성한 부모 루틴을 표시한 프로세스 윈도우(Process Window)로써 그림 10의 stack 프레임 내부에 있는 화살표 부분이다. 그림 9와 그림 10은 TotalView에서 OpenMP 프로그램으로 수

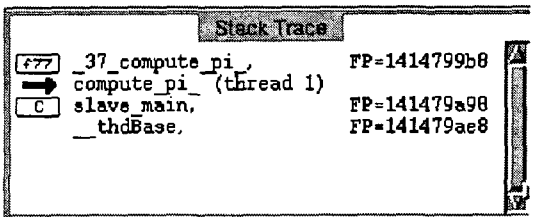


그림 10. ❶번의 Stack Trace를 확대

행한 결과를 보이고 있다.

병렬구조의 OpenMP 프로그램에서 지역변수들의 값을 보여주는 기능도 포함되어 있다. 디버깅하고자 하는 임의의 데이터 객체를 Dive 하면, 이 도구는 지역변수 또는 객체의 변화되는 값을 보여준다. 그림 6, 그림 7, 그리고 그림 8은 임의의 데이터 객체에 대해서 Dive한 결과를 보여준다. "thread-private" 데이터에 접근하는 것은 복잡하고 운영체제나 컴파일에 의존적이더라도 이 도구는 지역변수나 공유변수와 유사한 방법으로 값들을 보여준다.

3.2 Intel Thread Checker의 특징

Intel 사의 Thread Checker는 KAI에 의해 개발되었던 Assure를 재설계하였다 이 도구는 동

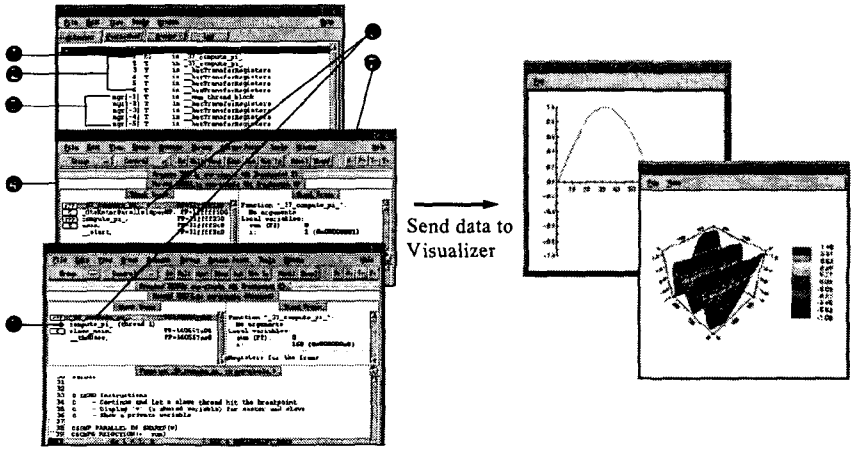


그림 9. OpenMP 프로그램 실행결과와 Array Visualization 윈도우

적분석 기법을 기반으로 오류들을 탐지한다. Intel Thread Checker는 Assure의 중요한 부분을 재설계하였다.

- ▶*Explicit Threading*: Assure for Thread와 Assure for OpenMP의 기능은 단일 제품으로 현재 통합되어서 명시적으로 스레드 기반의 프로그램의 분석 시에 이용된다. 그리고 이 도구는 프로그램의 모델에 따라 simulation technology 또는 projection technology로 변환된다.
- ▶*Binary Instrumentation*: 요즘의 프로그램은 라이브러리를 포함하고 있거나 운영체제에서 지원하는 함수들을 사용한다. 이 프로그램의 오류를 검사하기 위해서는 source instrumentation으로 탐지하기 힘들거나 비실용적이기 때문에 라이브러리 내에 있는 오류 탐지하기 위해서 binary instrumentation 기능을 지원한다.
- ▶*Source Instrumentation*: 원시 프로그램 수준에서 오류를 탐지하기 위해서 OpenMP 디렉티브를 분석하여 감시하는 기능을 지원한다.
- ▶*User Interface*: 디버깅, 성능분석 기능 등이 VTune Performance Environment에 플러그인 되어서 화면에 정보를 프로세스, 오류 종류 등으로 분류하여 보여준다.

Intel 사에서 제공되는 Intel Thread Checker 단독으로는 OpenMP 프로그램에서 발생하는 오류를 탐지할 수 없기 때문에 오류탐지를 위한 몇 가지 도구들을 지원한다.

- ▶*Intel C++ and Fortran compiler*: OpenMP 프로그램의 컴파일과 OpenMP 명세에 해당하는 디렉티브에 대한 Implementation 기능을 제공한다.
- ▶*Thread profiler*: VTune Performance Environment에 플러그인 되어 있으며, Thread Profiler는 OpenMP 프로그램의 성능분석을 위해서 설계되었다.
- ▶*Intel Thread Checker*: KAI Assure의 두 번째 통합 제품으로써 디버깅 기능을 제공한다.

3.2.1 디버깅 기법들

OpenMP 프로그램의 구조는 정적 또는 동적으로 분석이 가능하다. 정적분석 기법은 일반적으로 중요한 요소들에 의해서 제한적으로 사용되어진다. 이들 제한 요소들은 전체 프로그램의 분석, 프로그램 내에 외부 루틴들의 수, 그리고 입력 데이터에 따른 프로그램 흐름 (program flow)등이다. 동적분석 기법은 프로그램 분석에 이용되는 데이터 집합에 한계가 있다. 그러나 프로그래머는 순차 프로그램의 분석 시에 올바르게 수행 되는지에 의문을 가지면서 입력되는 값을 확장하면서 프로그램의 정확성을 입증한다. OpenMP 프로그램의 정확성을 얻는 것도 순차 프로그램에서 하는 방법과 동일한 방법으로 한다. 그러나 동적분석 기법이 임의의 프로그램에서 정확성을 입증할 수는 없지만, 프로그램에서 결점들의 사례(instance)들의 위치 파악에 매우 강력하다.

Intel Thread Checker는 OpenMP 프로그램의 정확성을 분석하기 위해서 두 가지 동적분석 기법을 이용한다. 첫 번째 기법은 일반적인 스레드 기반의 OpenMP 프로그램에서 적용된다. 스레드들간의 관계는 임의의 공유변수에 두 개 이상의 스레드들이 접근하는 것을 탐지하기 위해서 Lamport[12], Eraser[17], 그리고 RecPlay[16]에서 사용하는 탐지원리를 이용한다. 이 기법은 Assure for Java, Assure for thread에 사용되고, 명시적인 스레드 기반의 프로그램을 위해서 Intel Thread Checker에서 부분적으로 사용하고 있다.

이 기법을 "simulation technology"이라고 한다. 두 번째 기법은 순차 프로그램에 OpenMP 병렬화 디렉티브를 삽입한 프로그램에 적용된다. 이 기법은 순차 프로그램을 OpenMP 프로그램에서 명시한 수행으로 취급함으로써 이들 두 프로그램 간의 차이점들이 OpenMP 프로그램에서 오류들로 인식되어진다. 그러나 이 기법은 일반적인 OpenMP 프로그램에서는 한계가 있다. 그 이유는 스레드 번호에 의존적이고 특별한 스레드 번호에 작업을 할당하는 프로그램에 대해서는 적용되지 않기 때문이다. 따라서 OpenMP 프로그램은 "relaxed sequential programs"이라고 하는 속성을 가져야 하고, 이러한 기법을 "projection technology"라고 한다.

3.2.2 디버깅

Intel Thread Checker는 지능적인 디버거이다. 전통적인 디버거들인 dbx, Microsoft Visual Studio debugger는 break-point의 사용으로 프로그램을 제어할 수 있고, 프로그램 실행의 임의 지점에서 프로그램의 실행상태를 보여준다. 또한 프로그램 내에 감시대상이 있는 모든 명령어들이 실행될 때마다 일관성 분석하여 오류를 탐지한다. OpenMP 명세에 맞지 않는 부분은 즉시 탐지되거나 진단상의 오류를 탐지하기 위한 일반적인 "thread-safety" 규칙들이 생성되어진다. 이들 규칙들은 경합, 교착상태, 스톨, 그리고 API 위배 등을 포함하고 있다.

그림 11은 순차 프로그램에 "parallel for"를 삽입하여 병렬로 수행하는 OpenMP 병렬 프로그램이다. 이 예제는 Intel Thread Checker에 의해 프로그램의 분석을 준비하기 위해서 "icl/Qopenmp/Qtcheck sample1.c"라는 명령어를 이용하여 Intel 7.0 C++ Compiler 로 컴파일 했다. 이 프로그램이 Intel Thread Checker GUI 환경에서 실행되면

```
void
histogram( double D[], int N,
      double (*F)(double), int S ) {
  /* D[] accumulates the distribution   */
  /* F() is the function being measured */
  /* S  is the number of samples to take */
#pragma omp parallel for private(i) shared(D,F,N,S)
  for (int i=0; i<S; ++i) {
    int k=(F(i/(double)S)*N);
    D[ k ] += 1;
  }
}
```

그림 11. Intel Threa Checker의 실험프로그램

그림 12과 같은 결과를 생성한다. 이 그림은 세 가지 중요한 사용자 인터페이스를 보여준다. 첫 번째, 진단 리스트 윈도우는 변형된 프로그램(Instrumented program)의 실행동안 수집되어진 진단상의 오류를 보여준다. 프로그래머는 출력된 정보를 그룹화 할 수도 있다. 두 번째, 그래프 요약 윈도우는 진단 리스트에서 그룹화 된 것을 보여준다. 각 바(bar)에 있는 색깔은 각 그룹에서의 진단 수준별 오류의 종류이다. 바의 길이는 진단상의 오류의 개수를 의미한다. 마지막으로 소스뷰 윈도우는 다섯 개의 탭으로 구성되어져 있고, 첫 번째 탭은 소스코드의 줄을 의미하고 나머지 네 가지는 아래와 같은 것을 의미한다.

- ▶*Context*: 진단 오류를 포함하고 있는 스레드
- ▶*Definition*: 진단 오류가 발생한 객체, 메모리
- ▶*1st Access*: 객체나 메모리에 하나 이상의 스레드가 접근한 첫 번째 스레드
- ▶*2nd Access*: 객체나 메모리에 하나 이상의 스레드가 접근한 두 번째 스레드

그림 11의 OpenMP 프로그램에서 "D[k] += 1" 부분은 생성된 스레드 개수만큼 병렬로 수행하여 같은 메모리 접근하기 때문에 메모리 충돌현상이 발생하여 그림 12에 처럼 오류를 탐지한 결과를

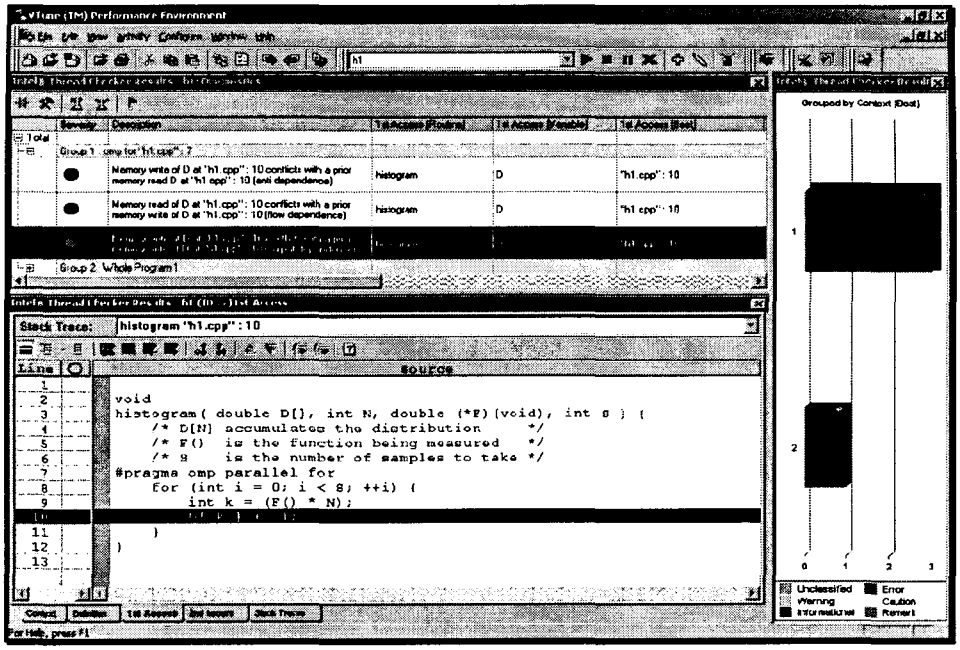


그림 12. OpenMP 프로그램을 Intel Thread Checker에서 실행된 화면

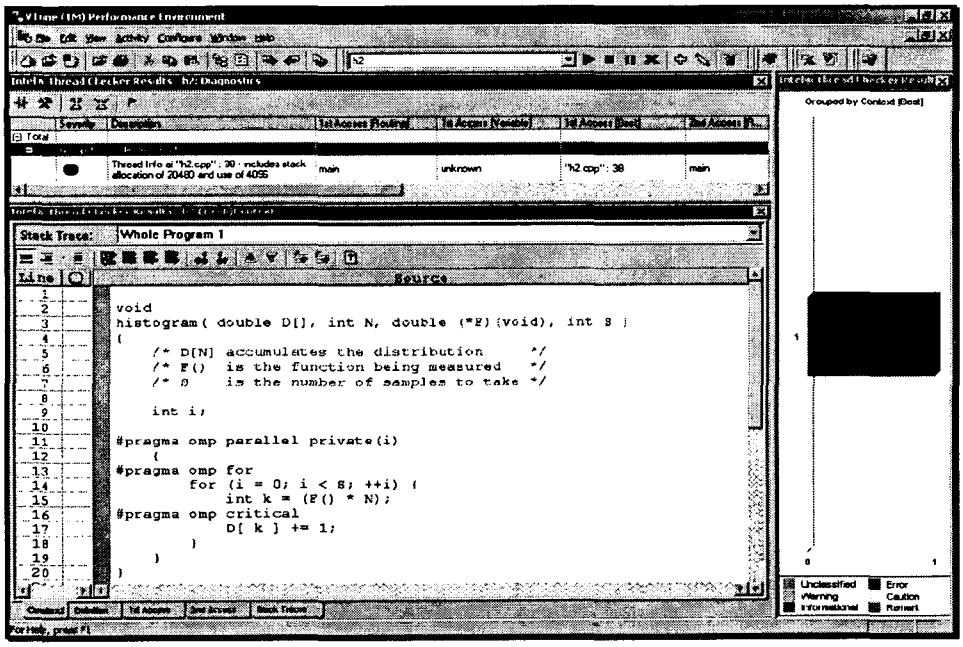


그림 13. Critical Section으로 오류 제거

보인 것이다. 이것을 해결하기 위해서는 생성된 모든 스레드가 순서적으로 메모리에 접근하게 해야 한다. 따라서 그림 13처럼 소스코드에 "Critical Section" 동기화를 사용하여 오류를 수정한다.

3.3 ProDev WorkShop

실리콘 그래픽사의 ProDev WorkShop은 디버거, 코드 분석기, 성능 분석기, 빌더 매니저, 시험범위 도구 등이 포함되어 있으며, 포함되어 있는 도구들의 이름을 살펴보면 디버거는 Visual Debugger, 코드 분석기는 Graphical Code Analyzr, 성능 분석기는 SGI SpeedShop Powerful Performance Analyzer, 그리고 빌더 매니저는 Integrated Build Managr이다. 이 중에서 디버깅을 제공하는 도구인 Visual Debugger에 대해 살펴본다.

3.3.1 Visual Debugger의 특징

Visual Debgguer는 프로그램의 실행동안 동적으로 멀티 그래픽 뷰로 소스 수준의 디버깅을 지원한다. 소스 수준의 디버거인 Visual Debugger는 프로그래머의 복잡한 필요한 것을 다루고 멀티 스레드나 멀티프로세스 기반의 프로그램을 디버깅하기 위한 특별한 지원을 제공한다. 예를 들면, 다양한 동적-업데이트 데이터 뷰는 데이터 구조나 메모리의 표현을 그래픽적으로 제공한다. 또한 멀티 스레드와 멀티 프로세서 하드웨어를 위한 프로그램과 더불어 동작하게 되고, 프로그램 코드가 정확하게 표현 되었는지 평가를 하는 특징도 있다. 종종 소스 수준이 아닌 상위 수준의 기능을 제공하는 다른 디버거들은 프로그램의 분량이 50,000 이상 라인을 처리할 수가 없을 뿐만 아니라 100MB 이상의 실행가능 한 프로그램도 처리할 수가 없다. 이러한 것을 Visual Debugger에서는 대량의 데이터를 이해가 될 수 있는 코드를 보여주는 다른 뷰를 제공함으로써 디버깅을 용이하도록 한다.

3.3.2 디버깅

Visual Debugger는 모든 플랫폼에서 지원되고 않고 단지 UNIX 시스템에서 한정으로 사용된다. 디버깅을 위해서 지원되는 기능들을 살펴보면 다

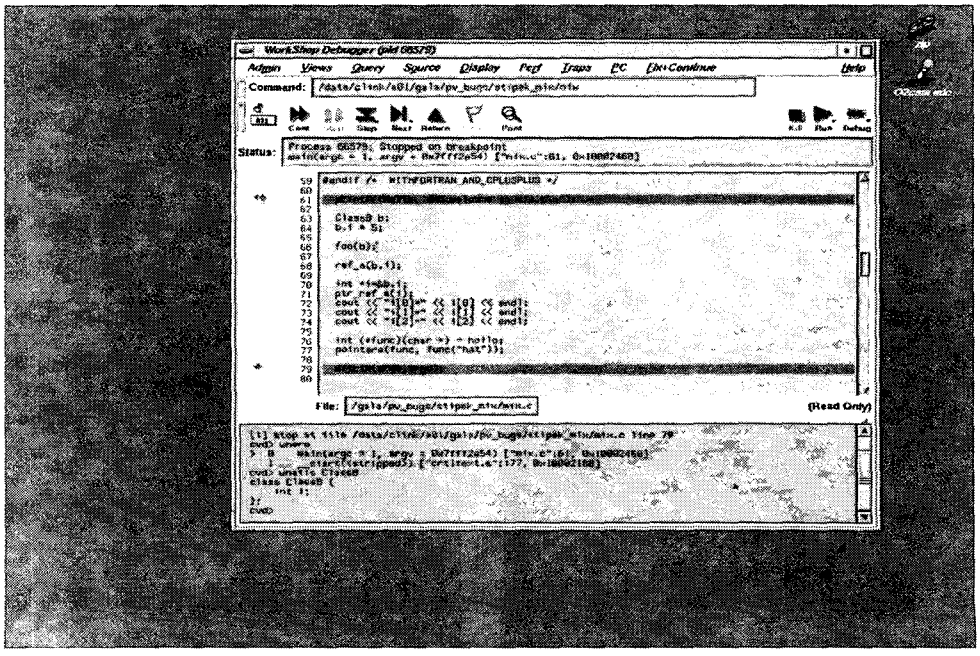


그림 14. Visual Debugger - I

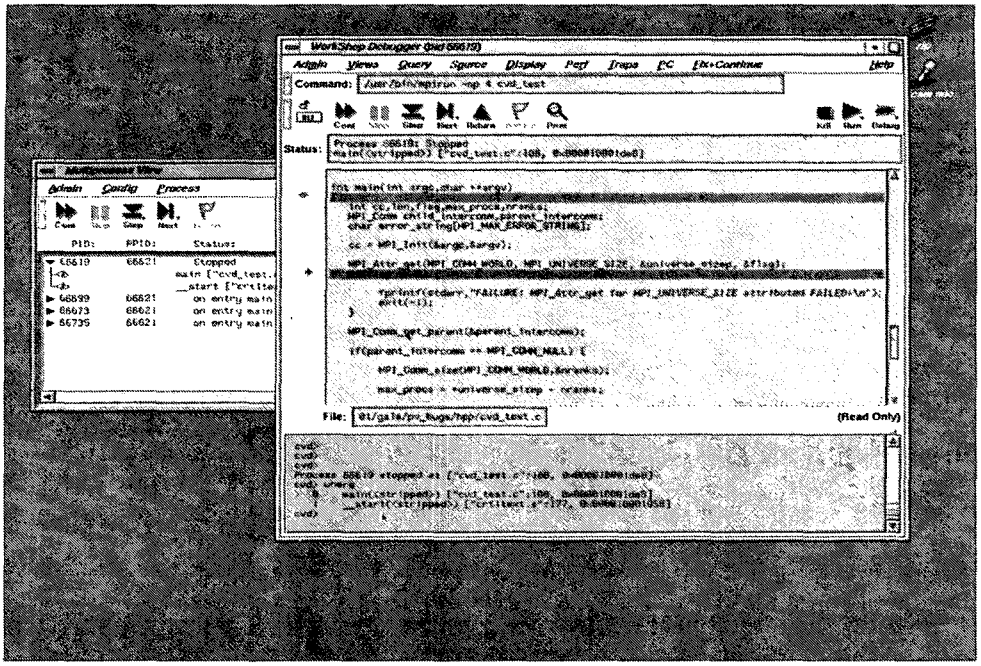


그림 15. Visual Dedugger - II

음과 같다.

▶수정과 진행

이 기능은 프로그래머에게 전체 프로그램의 재컴파일과 링크 없이 디버깅하여 변경할 수 있도록 한다. 즉 함수를 편집할 수 있고, 프로그램의 변화를 분석하거나 실행을 계속 진행할 수 있게 한다.

▶시각화

3-D 배열 Visualizer와 구조 브라우저와 같은 도구는 사용자가 표현이나 데이터의 시각적인 표시를 조사함으로써 그들의 코드에서 문제를 확인하도록 한다. Debugger은 사용자가 프로그램을 분석할 때 업데이트 된 프로그램으로 15개의 다른 "뷰"를 제공한다.

▶데이터 Watchpoints

watchpoint를 지원하는 대부분의 디버거들은 타켓 프로세스를 "single-stepping"으로 처리하기 때문에 프로그램의 속도저하를 유발하지만, WorkShop 디버거는 IRIX 운영체제에서 지원하므로 전통적인 방법보다 성능면에서 우수하다.

▶소스 수준의 표현 평가

WorkShop 디버거는 프로그램의 네이티브 구문에서 프로그래머가 포트란, C, 그리고 C++의 표현을 평가하도록 한다. 이 디버거는 정적이고 non-static 데이터 멤버와 같은 C++ 표현, 실질적인 기능, 다중 상속, 그리고 실질적인 기본 클래스 위해 폭넓은 지원하고 있다.

▶진보된 디버깅

세 가지 뷰는 강력한 기계수준의 디버깅 능력을 제공한다 - 등록 뷰, 메모리 뷰, 그리고 분해 뷰.. 각 뷰는 그것이 보여주는 값의 변경을 허락한다. 분해 뷰는 주소나 기능이나 파일에 의하여 보여지는 코드의 분해를 허락한다. 명령 줄 인터페이스를 선호하는 사람들의 대부분은 dbx 명령의 능률적인 액세스를 제공한다.

4. 결 론

지금까지 OpenMP 프로그램에서 발생하는 오류를 디버깅하는 도구에 대해서 살펴보았다. 세 가지의 디버깅 도구는 프로그램 발생에 발생한 오류를 디버깅할 수 있는 많은 방법들을 제공하고, 디버깅을 위해 필요한 정보들을 텍스트 위주나 그래픽 요소를 가미하여 사용자에게 제공한다. 그러나 이들 도구들은 프로그램의 수행이 잘못 되었더라도 어떤 부분에서 오류가 발생했는지 탐지를 못할 수도 있고, 오류를 탐지하더라도 잘못된 부분을 탐지할 수도 있다. 또한, 탐지된 오류 정보들이 너무 복잡하고 정보의 양이 많으므로 사용자가 이해하기 너무 어렵다는 것이다. 향후에 개발될 병렬 프로그램의 디버깅 도구의 연구 과제는 여러 가지들이 있겠지만 그 중에서 가장 중요하고 생각되는 것은 탐지된 오류 정보를 어떻게 하면 사용자가 오류를 효과적으로 디버깅할 수 있을까에 초점을 맞추어야 될 것이다.

참 고 문 헌

[1] Bunde, M. K., Debugging Complex Code Couldn't be Simpler, Etnus, Spring, 2002.

[2] Callahan, D., K. Kennedy, and J. Subhlok, Analysis of Event Synchronization in a Parallel Programming Tool, 2nd Symp.on Principles and Practice of Parallel Programming, pp. 21-30, ACM, March 1990.

[3] Cownie, J., and S. Moore, Portable OpenMP debugging with TotalView, Etnus, WOMPAT2000, 2000.

[4] Dagum, L., and R. Menon, OpenMP: An Industry-Standard API for Shared-Memory Programming, Computational Science & Engineering, Vol. 5, No. 1, IEEE January/March 1998.

[5] 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.

[6] Etnus Co., TotalView: Release Notes, Version 5..0, June 2001.

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

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

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

[10] Kim, J., and Y. Jun, Scalable On-the-fly Detection of the First Races in Parallel Programs, 12nd Int'l. Conf. on Supercomputing, pp. 345-352, ACM, July 1998.

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

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

[13] 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.

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

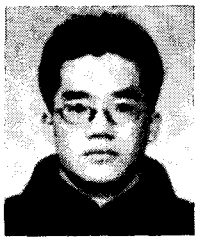
[15] Petersen, P., and S. Shah, OpenMP Support in the Intel Thread Checker, WOMPAT 2003, pp. 1-12, June 2003.

[16] Ronsse, M., and K. De Bosschere, RecPlay: A Fully Integrated Practical Record/Replay System, Tr. on Computer Systems, 17(2): 133-152, ACM, May. 1999.

[17] Savage, S., M. Burrows, G. Nelson, P. Sobalvarro, and T. Anderson, Eraser: A Dynamic Race Detector for MultiThreaded Programs, Tr. on Computer System, 15(4): 391-411, ACM, Nov. 1997.

[18] SiliconGraphics Co., Software Development Environment: Visual Tools for the IRIX Platform,

[19] Snir, M., S., Otto, S. Huss-Lederman, D.Walker, J. Dongarra, MPI: The Complete Reference, MIT Press, 1996.



김 영 주

- 1999년 경상대학교 컴퓨터과학과 학사 졸업
- 2001년 경상대학교 컴퓨터과학과 석사 졸업
- 2001년~현재 경상대학교 컴퓨터과학과 박사과정
- 관심분야 : 운영체제, 리눅스 클러스터링, 병렬프로그램 디버깅

김 정 시

- 1999년 경상대학교 컴퓨터과학과 졸업(박사)
- 2000년 한국정보통신대학원 Post Doc.
- 2000년 12월~현재 한국전자통신연구원 임베디드소프트웨어기술센터 선임연구원
- 관심분야 : 임베디드 시스템 개발 도구, 병렬프로그램 디버깅