□ 기술해설 □

병렬 프로그램 디버깅 기술

한국전자통신연구소 온기원* · 이범식* · 홍철의* · 지동해** · 임기욱**

목 차

1. 서 론

병렬 프로그램을 병렬 컴퓨터 또는 다중 프로세서 시스템 상에서 실행시킬 경우, 서로 다른 프로세서나 노드(프로세서 셋)에 있는 실행 객체(프로세스, 쓰레드)들이 동시에 수행된다. 이는 병렬 프로그램 실행의 비결정적 특성[1]을 말하는데, 프로그램 실행에 있어서 전반적인 성능 향상을 가져올 수 있다. 반면에 각각의 실행에 대하여 시스템 상태 및 객체간 통신 상황에 따라 서로 다른 실행 경로를 가지게되므로, 오류가 발생한 실행 경로에 대한 보장을 어렵게 한다. 즉 병렬 프로그램 디버깅의 관점에서 볼때 단일 실행 경로 상에서 이용되었던 순차 디버거의 한계를 의미하며, 특히 탐침 효과 및 데이타 경합 등의 문제를 동시에 해결할 새로운 디버깅 방법을 요구하게 된다. 한편 병렬 프로그램의 실행 과정에서 발생하는 오류의 대부분(deadlock, data race)은 실행 객체들간의 데이타 교환 과정에 기인하게 되므로, 병렬 디버거는 디버깅에 필요한 모든 데이타를 한곳에서 처리할 수 있는 기능을 제공할 필요가 있다. 이는 디버깅 환경에서 디버깅 데이타에 대한 가시적인 통합 뷰를 제공하는 사용자 인터페이스를 요구하게 된다[2, 3, 4]. 본 논문에서는 병렬 프로그램 개발 환경의 일부인 병렬 디버거에 대하여 기술하였다.

2장에서는 병렬 프로그램의 비결정적 특성을 유발하게 하는 원인 및 영향을 디버깅 관점에서 살펴 보았다. 3장에서는 기존의 순차 프로그램을 위한 디버깅 방법의 핵심을 이루는 정지점(breakpoint) 기법과 병렬 프로그램 실행의 재실행(execution replay) 기법에 대하여 중점적으로 다루었다. 정지점 기법은 병렬 프로그램의 디버깅에서도 기본적으로 이용될 수 있기 때문에 본 고에서는 여러 가지 형태의 정지점 방법(국부적/전역적 방법)을 분석하였다. 이어서 재실행에 필요한 사건의 개념 및 종류, 사건의 전역적 순서화를 해결하기 위한 타임 스탬프 기법에 관하여 설명하였다. 또한 재실행 기능에 대한 정형적인 표현 방법을 제시하였고, 대상 하드웨어 시스템의 구조를 크게 메시지 전달형과 공유 메모리형으로 구분하여 각각의 형태에 따라 대표적인 재실행 기법

*정 회 원
**종신회원

들을 분석하였다. 4장에서는 병렬 디버거의 사례로 현재 한국전자통신연구소에서 고속병렬컴퓨터(SPAX)에 탑재를 목표로 개발 중인 병렬 디버거 ParaDebug에 대하여 소개하였다. ParaDebug는 순환 디버깅 기법과 재실행 기법을 기반으로 하는 소스 레벨 디버거로서, 디버깅되는 프로그램의 데이타를 텍스트와 그래픽 형태로 보여주는 그래픽 디버깅 환경을 제공한다. ParaDebug의 구성 요소인 디버거 코아, 프로그램 실행 재실행 구동기, 그래픽 사용자 인터페이스에 대하여 설계 내용과 기능 등을 기술하였다.

2. 병렬 프로그램 디버깅의 문제점

2.1 탐침 효과(Probe Effect)

일반적으로 프로그램을 디버깅할 때 프로그램 실행 상태에 관한 정보를 얻기 위하여 부가적인 코드를 원래의 프로그램에 추가시키는 경우가 있다. 이때 결정적인 실행 특성을 갖는 순차적인 프로그램의 경우, 이로 인한 문제가 발생하지 않지만 병렬 프로그램에 대해서는 추가적인 내용이 원래의 실행 결과에 영향을 주는 현상이 발생할 수 있다. 이러한 현상을 탐침효과(probe effect)라 한다. 예를 들어, 임의의 한 프로세스가 동일한 비결정적 선택(nondeterministic selection) 언어 구조를 사용하여 상대 프로세스들 중의 하나와 통신하는 경우를 생각하여 보자. 이때 상대 프로세스의 일부에 디버깅을 목적으로 프로그램의 일부를 추가시킨다면 원래의 실행 속도보다 추가된 디버깅 코드로 인하여 지연되므로 통신할 수 있는 기회가 적어질 것이다. 만일 이와 같이 상대 프로세스의 실행이 항상 연기된다면 디버깅 코드를 추가시켰음에도 불구하고 원하는 실행 순서를 얻지 못하게 된다.

2.2 비결정적 프로그램 실행

병렬 프로그램은 실행되는 시스템의 형태에 따라 서로간에 메시지를 교환하거나(message passing mechanism) 공유 메모리를 이용하여 통신하는(shared memory mechanism) 여러 개의 프로세스/쓰레드들로 구성된다. 병렬 프로그램의 중요한 특징 중의 하나는 병렬 프로그램을 이루는 각 프로세스의 상대적인 실행 속도에 대한 가정이 없다는 것이다. 이는 병렬 프로그램의 실행에 있어서 소위 말하는 '비결정성(nondeterminism)'을 의미한다. 비결정성의 원인은 병렬 프로그램을 작성할 때 가능한 모든 실행 순서를 명시할 수 없음으로 인하여, 프로세스들 사이에 존재하는 경합(race)에 따라 하나의 동일한 입력에 대한 병렬 프로그램의 실행을 예측할 수 없기 때문이다[1, 5]. 프로그램 실행의 비결정성은 프로세스들의 경합 이외에도 프로그램의 비결정적 언어 구조에 의해서도 나타날 수 있다. 이러한 병렬 프로그램의 비결정성은 순차적인 프로그램에서와는 달리, 동일한 입력에 대해서도 다른 결과를 발생시키므로, 프로그램의 작성 및 판독, 그리고 디버깅을 어렵게 하는 요인이 된다.

3. 병렬 프로그램 디버깅 기술 분석

3.1 정지점 기법

본 장에서는 전통 디버거에서 사용되는 일반적인 정지점 기법을 설명한다. 다음에 병렬처리시 병렬 프로세스/쓰레드간의 인과 관계를 표현하는 인과 관계 정지점 기법에 대하여 설명하고, 이를 효율적으로 실행하는 근사 일치형 검사점에 대하여 기술한다.

3.1.1 기본 정지 기능

순환 디버깅 기법을 기반으로 하는 전통 디버거에서 정지점 설정은 가장 중요한 기능 중의 하나이다. 일반적으로 병렬 디버거에서는 순차 디버거와 같은 방법으로 정지점을 설정한다. 이러한 정지 점은 지정된 소스 문장에서의 정지 기능, 예외가 발생하였거나 사용자가 지정한 사건이 탐지하였을 때 정지하는 기능, 특정한 변수가 접근되었을 때 정지하는 기능, 어떤 조건식이 만족 되었을 때 정지하는 기능 등을 제공한다. 순차 디버거와는 달리 병렬 디버거에서는 정지점을 두 가지 방법으로 실행할 수 있다. 즉 정지점에 도달 하였을 때 모든 프로세스가 즉시 정지하거나, 정지점을 만난 프로세스만 정지 시키는 방법이다. 첫번째 방법

은 충분히 작은 시간 안에 실행하기가 힘들며, 두번째 방법은 시간 초과와 같은 시간 종속적 기능을 가진 시스템에서는 심각한 장애가 발생한다. 두번째 방법을 구현하기 위해서 어떤 시스템에서는 임의의 프로세스가 정지점에 도달하였을 때, 정지하는 기능을 가진 논리적 시간을 제공하여 표면적인 경합 조건을 풀려고 시도하였다. 일정한 프로세스만 정지시키는 시스템에서는 다른 프로세스는 시간 종속적 연산을 만날 때까지 실행된다. 이러한 경우, 시간 종속적 연산을 측정하는 데 논리적 시계를 사용한다. 반면에, 정지점을 만났을 때 모든 프로세스를 정지시키는 시스템에서는 논리적 시간이 정지하여 정지된 프로세스들이 최소한의 영향으로 다시 실행할 수 있다. 이러한 방법은 탐침 효과를 전부 없앨 수는 없으나, 시간 초과와 같은 시간 종속적인 연산이 존재할 때 전통적 방법의 디버깅 기능을 제공한다. 정지점을 지정하는 데 필요한 표현식이나 조건식의 범위는 여러 프로세스의 상태나 사건을 포함하기 때문에 순차 프로그램에 비하여 훨씬 크다. 사건은 단위 프로세스의 범위 밖에서도 측정 기능한 atomic 실행으로 정의될 수 있다. 대부분의 시스템에서는 전역 시계의 부재로 인하여, 전역 상태를 나타내는 조건 문은 문제를 발생시킬 수 있다.

예를 들어, "프로세스 A가 변수 X를 변경하고 있는 동안 프로세스 B는 변수 X를 변경할 수 없다"는 조건식은 통신 지연으로 인하여 틀리게 판정될 수 있다. 보다 중요한 것은 조건식이 만족된 것을 탐지한 후 그 상태가 변경되기 전에 프로세스를 정지시키는 것이 불가능할 지도 모른다는 사실이다. 사건과 전역 상태의 차이는 불분명하나, 대체로 다음과 같이 정의된다. 예를 들어, "프로세스 A가 메시지를 보내는 것"은 사건으로 정의 되며, "메시지 버퍼가 프로세스 A로 부터의 메시지를 갖고 있는 것"은 전역 상태로 정의 되어 진다. 이러한 것은 프로그램 카운터를 이용하여, "프로세스 A의 메시지 보냄" 문장 바로 뒤에 정지점을 설정하는 기법으로 사용될 수 있다. 사건 기반형 조건식을 사용하는 시스템에서는 사건을 나타내는 언어가 필요하다.

3.1.2 인과 관계 정지점

프로그래머가 정지점 기능을 이용하여 정지된 프로그램의 상태에 영향을 미치는 인과 관계의 사건들을 조사하고자 할 때, 순차 프로그램에서는 인과 관계를 가지는 사건을 정의하는 것이 모든 사건들이 완전 순서화를 이루고 있으므로 손쉬우나, 병렬 프로그램에서는 사건이 부분 순서화를 이루고 있으므로 전체 프로그램에 대하여 순간 snapshot을 얻기가 불가능하다. 인과 관계 정지점은 Lamport[6]의 병렬 시스템에서의 사건의 부분 순서화에 기초하여 각 프로세서가 정지점 이전에 발생한 모든 사건들을 반영하는 상태로 복귀하게 한다. 반면에, 일반적인 병렬 정지점 기법은 단지 정지점을 포함한 임의의 전역 일치 상태를 나타낸다[7]. 인과 관계 정지점을 실행하는데, 종속적 로그인(login)을 이용하여, 각각의 프로세스를 재시작 또는 재실행을 가능하게 하여 실행의 결정성을 유지한다. 종속적 로그인은 메시지 전달시 보내는 프로세서의 사건 색인을 함께 보내게 하여 보내는 프로세서 및 사건을 함께 저장하여 재실행시 결정성을 유지시킨다. 또한, 근사 일치 검사점 집합을 이용하여 적은 로그 데이타를 갖고도 빠르게 원하는 상태를 회복하게 한다. 정지점 사건은 정지점을 실행한 프로세스에서 정지점 이전에 발생한 사건중 가장 최근에 발생한 사건을 말한다. 인과 관계 정지점은 다음과 같은 시스템 상태로 정의된다.

1. 정지점 프로세스의 경우, 정지점을 설정할 때의 상태를 말한다.
2. 다른 프로세스의 경우, 정지점 사건 이전의 해당 프로세스의 모든 사건을 반영하는 최초의 프로세스 상태를 말한다.

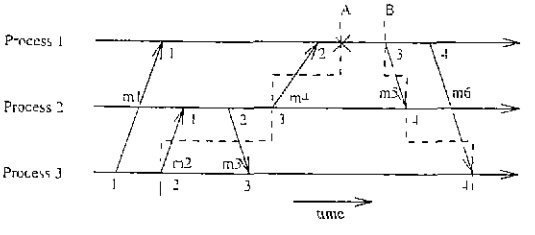


그림 1 프로세스 1의 프로세스 3에 대한 인과 관계 정지점

어떠한 인과 관계 정지점도 일치 시스템 상태이다. 그림 1은 3개의 프로세스 실행 상태를 보여준다. 메시지 m1에서 m6가 프로세스 사이에서 전달되었다. 정지점은 프로세스 1 에서 x로 표시된 부분에서 발생한다. 정지점 사건에 대한 인과 관계 정지점은 A 선에 걸쳐 있는 각 프로세스의 상태로 표시된다. 정지점을 포함한 임의의 일치 상태를 정지점 상태로 표시하는 기존의 방법과는 달리, 인과 관계 정지점 에서는 각각의 프로세스가 정지점 이전에 발생한 모든 사건을 포함하는 최초의 상태로 표시된다. 그림 2에서, A와 B선 사이의 어느 상태도 일치 시스템 상태를 나타낸다. 이 중에서 인과 관계 정지점은 일치 시스템 상태중 최초의 상태인 A선에 의하여 나타내어 진다. B선을 시스템 상태로 가정하면, 프로세스 3은 메시지 m3를 받은 후이다. 그 결과 프로세스 3의 상태가 변화되어 메시지 m2의 전송에 대한 정보가 없어졌다. 메시지 m2는 프로세스 1의 정지점 상태에 대하여 인과 효과를 가지고 있는 사건이므로 이에 대한 정보가 프로세스 상태에 나타내져야 한다.

3.1.3 근사 일치형 검사점

종속 정보만이 로그 파일에 저장되어 있을 경우, 구하여진 인과 관계 정지점 상태로 복귀하기 위해서는 처음부터 프로그램을 재시작 한다. 만약 종속 정보와 더불어 메시지의 데이타까지 로그 파일에 저장하면, 복귀 시간은 많이 단축되나, 로그 파일의 데이타 양이 엄청나게 많아 진다. 실행 중 검사점을 사용하면 원하는

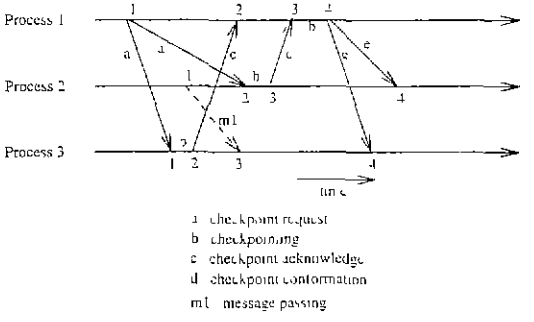


그림 2 근사 일치형 검사점 집합

상태로 복귀하기 위해서 원하는 상태에서 가장 가까운 검사점으로부터 프로그램을 재실행함으로써 복귀 시간을 단축시킬 수 있다. 근사 일치형 검사점 집합은 로그 파일의 양을 적게 유지하면서 복귀 시간을 줄이는 기법이다.

그림 2에서 검사점를 설정하는 프로세스(검사점 프로세스, 프로세스 1)는 검사점 요구(a)를 모든 다른 프로세스에게 전파한다. 검사점 요구를 받은 프로세스는 검사점를 행하고(b), 검사점 요구에 대한 응답을 검사점에 기록된 사건 색인과 함께 보낸다(c). 검사점 프로세스는 다른 모든 프로세스로 부터 응답을 받은 후 검사점을 실행하고(b), 다른 모든 프로세스에게 각 프로세스의 검사점에 기록된 사건 색인을 포함한 확인 메시지를 보낸다(e). 각 프로세스는 검사점과 확인 메시지 수신사이에 전달된 메시지 (m1)에 대하여 종속 정보 및 메시지의 데이타까지 로그 화일에 저장한다. 각 프로세스는 확인 메시지를 수신한 후, 검사점과 확인 메시지 수신 사이에 전달된 메시지의 전송 사건 색인 (1)과 확인 메시지에 의하여 전달된 같은 전송 프로세스의 검사점 사건 색인 (2)와 비교하여, 검사점 사건 색인 보다 검사점과 확인 메시지 수신 사이에 전달된 메시지의 전송 사건 색인이 작으면, 전송 사건 후에 검사점이 발생하였으므로, 종속 정보와 더불어 메시지 데이타도 함께 저장한다. 그리하여, 검사점으로 부터 재실행할 때 검사점 전에 보내진 메시지도 로그 화일을 통하여 데이타를 얻을 수 있다.

3.2 프로그램 실행의 재실행 기법

3.2.1 재실행 기본 개념

프로그램의 실행 과정에서 나타나는 오류를 찾기 위해서는 먼저 프로그램이 실행되었던 경로와 똑같은 경로의 재실행이 보장되어야 한다. 순차 프로그램에서의 경우, 하나의 동일한 입력에 대하여 항상 일정한 실행 경로를 가지게 되므로 재실행 과정에서는 이렇게 한번 수행된 경로를 그대로 따라서 디버깅을 하면 된다. 그러나 병렬 프로그램 실행은 여러 개의 실행 단위들을 토대로 이루어지는 것을 전제로 하기 때문에, 프로그램 실행 과정에 비결정성

을 내재하고 있다. 이는 결코 하나의 동일한 프로그램 실행 경로에 대한 재실행을 보장할 수 없는 것으로써, 병렬 프로그램을 효과적으로 디버깅 하기 위해서는 재실행 기능의 제공이 필수적이다.

프로그램 재실행의 대표적인 기법으로 공유 메모리 시스템 상에서 실행되는 프로그램을 대상으로 하는 Instant-Replay 기법[5], 메시지 전달형과 공유 메모리형을 각각 고려한 최적 추적 및 재실행 기법[8, 9], 타임스탬프와 인과 관계를 이용한 인과 관계 기반형 재실행 기법[10], 그리고 병행 Ada 프로그램을 대상으로 한 결정적 실행[11] 등을 들 수 있다. 이러한 기법들은 대상으로 하는 시스템의 구조적 특징이나 운영 체제, 그리고 여러 가지의 환경 변수들에 따라 다양한 형태의 재실행 방법을 보여 주고 있다. 재실행 기능이 고려하는 몇 가지의 공통적인 요소들을 정리하여 보면 다음과 같다.

- 병렬 프로그램의 실행을 사건의 연속으로 구성함
- 재실행을 위하여 사건에 순서를 부여함
- 각 처리기(프로세스 또는 쓰레드)들 간의 정보 교환 과정을 중심으로 사건을 구성함
- 재실행 상태를 가시적으로 보여 줌

3.2.2 재실행을 위한 사건의 정의 및 종류

병렬 프로그램을 디버깅하는 과정에서 프로그램의 재실행에 필요한 실행 경로를 정하기 위해서는 기준점들이 있어야 한다. 병렬 프로그램의 실행에서 경로의 구성은 사건을 기반으로 이루어진다. 사건의 정의는 디버거를 제공하는 각각의 시스템 및 프로그래밍 환경에 따라 서로 다른데, 기본적으로 각 처리기(프로세스 또는 쓰레드)들 간의 정보 교환 과정을 중심으로 이루어진다. 대체적으로 메시지 교환이나, 메모리 접근, 공유 자원 및 객체의 접근 등을 사건으로 정의한다[12,13]. 표 1은 병렬 시스템의 형태별로 각각 달리 정의된 사건을 보여준다.

분산형 메모리 구조를 가진 시스템에서 수행되는 병렬 프로그램의 경우 프로세스들간의 통신 연산자들(send, receive)을 대상으로 사건을 정의한다. 이에 비하여, 공유 메모리 구조를 가지는 시스템에서는 공유 메모리 접근에 관련한 연산자들을 대상으로 사건이 정의된다. 예를 들어, 다중 쓰레드형 병렬 프로그래밍 환경하의 공유 메모리 형태를 가지는 시스템을 대상으로 할 때, 다음과 같은 연산자들이 시스템 차원의 사건으로서 정의될 수 있다. 이러한 사건들은 쓰레드 관련형 사건, 객체 관련형 사건, 그리고 동기화 사건들로 구성된다[13].

표 1 병렬 시스템별로 본 사건의 정의 형태

사건형태	병렬시스템 특징	구현 사례
메모리 접근	전역적 공유메모리 제공	DISDEB 시스템
메시지 교환	메시지 전달형	Radar
공유객체 접근	공유메모리형, 국부적 순서	Agora
IPC	모든 사건이 기록됨, 공유메모리형	mtdbx
IPC, 공유객체 접근	공유메모리/메시지 전달 지원	Recap
프로그래머로 부터 정의됨	Ada rendezous 모델, 스케줄러 조절 기능	TSL, EDL
IPC & porcess internal activity	explicit interprocess 통신	

- 쓰레드 관련형 사건
  - fork/join 사건 :
    Thread_Fork, Thread_Join
  - 쓰레드 시작/종료 사건 :
    Thread_Start, Thread_End
- 객체 관련형 사건
  - 잠금/해제 사건 :
    Read_Lock, Read_Unlock,
    Write_Lock, Write_Unlock
  - Invocation 사건 : Object_Invoke
    (쓰레드의 국부적 사건으로 간주됨)
- 동기화 사건
  - Semaphore 사건 :
    SemP_Attempt. SemP_Success, SemV
  - Barrier 사건 :
    Barrier_Attempt, Barrier_End

3.2.3 사건 데이터의 저장

병렬 디버거는 사용자가 알고자 하는 프로그램의 동작 상태를 보여 주거나 혹은 프로그램을 재실행시키기에 필요한 정보를 얻을 수 있

어야 한다. 그러기 위해서 병렬 프로그램은 실행 시에 그 정보를 나타낼 수 있는 특정 데이타를 어떤 장소에 저장해 둬야 한다. 실행 프로그램 자신이 소프트웨어적으로 사건의 발생을 기록하도록 프로그램 안에 probe 코드를 가져야 하며, 이 코드가 데이타를 외부로 전송함으로써 데이타를 기록할 수 있다. 이때 저장되는 데이타의 양은 저장된 데이타를 나타내는 사건 히스토리가 어떻게 쓰일 것인가에 따라 결정이 되며, 그 양식은 쉽게 관심 있는 내용을 질의 할 수 있고 다른 형태로 데이타를 변경하기 편하게 특정한 형태로 저장된다. 사건 히스토리의 내용과 양은 브라우징, 재실행, 실행 에니메이션에 따라 다르다. 대부분 사건 히스토리에 기록된 데이타의 양은 매우 크기 때문에 사용자가 알고자 하는 정보를 찾아 내기가 어렵다. 이 사건 히스토리에서 필요한 정보를 뽑아 내는데 도움이 되는 여러 가지 기법이 개발되어 있으며 그 중의 하나는 데이타 베이스를 이용하여 정보를 기록하고 쉽게 관련된 정보를 추출할 수 있게 하는 것이다. 데이타를 저장하는 장소에 따라 하나의 파일에 모든 데이타를 기록하거나, 쓰레드나 프로세스 당 각각의 파일로 저장하는 방법이 있다. 그리고 저장 방법으로는 데이타 베이스를 이용하여 모든 데이타를 한 곳에 저장할 수 있으며, 또 데이타를 저장하는 절차로는 파일에 직접 저장하거나 IPC를 이용하여 저장을 위한 특별한 서버 프로그램에게 데이타를 전달하여 저장하는 방법이 있다. 데이타의 저장 방법으로 연결망을 이용하면 실제 프로그램의 실행, 데이타의 저장, 데이타를 보여 주는 도구의 사용을 서로 다른 컴퓨터에서 처리 할 수 있다. 즉 그래픽이나 사용자 인터페이스를 병렬 처리 시스템이 아닌 그래픽 워크스테이션에서 수행 할 수 있다. 그러나 데이타를 저장하는데 많은 부담이 지워지면 탐침 효과에 의해 프로그램의 실행에 큰 영향을 미치기 때문에 많은 주의를 기울여야 한다.

3.2.4 사건의 배열 및 논리적 클럭

프로그래머가 병렬 및 분산 프로그램을 디버깅하려고 할 때 프로세스의 상태나 실행 중에 생성되는 사건들의 발생 순서를 정확하게 알 수 있는 방법이 없다고 한다면 오류의 위치를 국부화 시키는데 많은 어려움이 따르게 된다. 물리적으로 분산되어 있는 프로세스들 상에서 발생된 프로세스의 생성, 종료, 메시지의 송/수신 등과 같은 사건의 발생을 검출하기 위해 관찰자 프로세스가 사건이 발생하였다는 메시지를 해당 프로세스로부터 수신한다. 그러나 망 지연 등과 같은 요인으로 인하여 메시지 전송 시간을 정확하게 예측할 수 없기 때문에 이러한 사건 발생 메시지들의 도착 시간은 실제로 사건이 발생한 순서와 일치하지 않는다. 이러한 문제를 해결하는 하나의 방법으로 사건을 발생시킨 프로세스에서 국부 물리적 클럭을 이용하여 사건 발생 메시지에 실제 시간을 기록하여 전송하는 방법을 생각해 볼 수 있다. 그러나 이 방법은 국부 물리적 클럭이 적절하게 동기화 될지라도 프로그램의 실행과는 무관한 사건들 때문에 CPU가 영향을 받을 수 있으므로 사용하기에는 어려움이 따른다. 물리적 클럭을 사용하여 사건을 순서화시키는 문제점을 해결하기 위하여 Lamport는 논리적 클럭(logical clock)의 개념을 도입하였다[6,13]. Lamport의 논리적 클럭은 메시지 전송으로 상호 통신하는 프로세스들이 실행 중에 발생시킨 사건들이 부분 순서(partial order)를 이룬다고 가정하였다.

4. 병렬 디버거 사례 : ParaDebug

ParaDebug[14]는 고속병렬컴퓨터(SPAX)의 병렬 프로그래밍 환경인 TOPS[15]의 구성 요소로서, TOPS를 구성하는 다른 구성 요소(성능 가시화기, 병렬 컴파일러)들과 유기적인 인터페이스를 가지고 있다. ParaDebug는 병렬 C 프로그램의 하나인 ParaC[16]로 작성된 병렬 프로그램을 소스 레벨에서 디버깅하는 심볼릭 디버거이다. 이 병렬 프로그램은 ParaC로 부터 변환(translated)된 형태로서 쓰레드와 MPI 루틴을 포함하고 있다.

4.1 ParaDebug 구조

병렬 디버거 ParaDebug의 설계는 비결정성

을 재실행 기법을 이용하여 결정적 실행 형태로 병렬 프로그램을 변경하는 방법으로 디버깅이 가능하게 하였다. 또한 재실행을 통하여 탐침효과를 효과적으로 감소시키도록 하였으며, 그래픽 사용자 인터페이스를 제공하여 병렬 프로그램의 디버깅을 쉽게 하였다. ParaDebug 구조는 그림 3과 같이 기본기능(debugger core), 재실행 구동기(replay driver), 그래픽 사용자 인터페이스의 세 부분으로 구성이 되어 그 상호간에 협력하며 디버깅을 수행한다. 기본 기능으로는 전통적인 디버거의 기능을 포함하며 프로세스/쓰레드를 처리할 수 있는 기능을 가지며, 재실행 구동기는 병렬 프로그램의 사건들의 인과 관계를 저장한 사건 히스토리를 참조하여 재실행 가능한 병렬 프로그램을 생성한다. 그리고 그래픽 사용자 인터페이스는 디버깅을 편리하게 할 수 있도록 그림의 형태로 병렬 프로그램의 동작 특성을 보여주며, 텍스트/그래픽 사상과 소스 브라우징을 제공한다. ParaDebug는 UnixWare 2.0 상에서 제공되는 디버거[17]를 기반으로하여 확장하여 설계 하였다. 다음은 ParaDebug의 각 블럭에 대한 주요 기능을 구체적으로 기술 하였다.

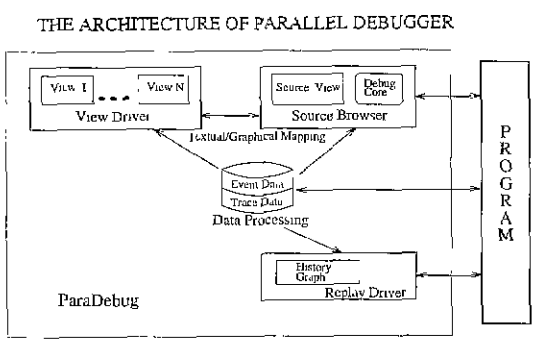


그림 3 병렬 디버거(ParaDebug) 구조

4.2 기본 기능

ParaDebug 코어(core)는 직접적으로 실행 객체를 제어하며, 위로는 사용자 인터페이스가 요구하는 내용을 추출하여 전달하는 기능을 가지고 있다. 디버깅을 위해 운영체제는 프로세스 화일 시스템(/proc) 인터페이스를 제공하여, 하나의 사용자 수준 프로세스가 다른 프로세스나 프로세스내의 각각의 LWP(Light Weight Process)를 직접 제어할 수 있도록 하였다. 그러나 디버거는 쓰레드의 생성 및 종료 시기, 쓰레드와 LWP의 관계가 변화하는 시기, 그리고 각 쓰레드에 관하여 식별자라든가, 스택의 경계, LWP에 현재 연결되어 있지 않은 쓰레드에 관한 문맥(context) 정보를 얻기 위해서 별도의 쓰레드 라이브러리 인터페이스를 필요로 한다.

ParaDebug는 프로그램, 프로세스, 쓰레드의 3단계 계층구조로 객체를 제어한다. 이 중에서 디버거가 직접적으로 처리해야 할 객체는 프로세스와 쓰레드이다. 대부분의 경우에 디버거가 이 객체에 적용해야 할 것들은 비슷하다. 이들은 run, step, stop, print a stack trace, dump the registers 등이다. 또한 새로운 프로세스를 생성 하거나 프로세스를 획득(grabbing)하는 기능도 제공한다.

4.3 재실행 구동기[18]

병렬 프로그램의 실행 경로의 구성은 사건을 기반으로 이루어진다. 사건의 정의는 각 처리기(프로세스 또는 쓰레드)들 간의 정보 교환 과정을 중심으로 이루어진다. 즉, 메시지 교환, 공유 메모리 접근 및 공유 자원 접근 등을 사건으로 정의한다. 프로그램의 초기 실행시 재실행에 필요한 부분 순서를 정의하기 위하여 최소한의 정보를 사건 기록으로 저장한다. 그러나 모든 사건을 기록하는 대신 인과관계(casual-link) 만을 기록해도 사건들 간의 종속 관계를 명확히 규정할 수 있다.

병렬 프로그램의 초기 실행이 끝난 후, 재실행을 위하여 발생한 사건들의 타임 스탬프를 생성한다. 사건 데이타의 수집은 실행 프로그램 안에 소프트웨어적으로 사건의 발생을 기록하도록 probe 코드를 삽입하여 이 코드가 사건 추적 정보를 외부로 전송함으로써 데이타를 기록한다. 인과 관계 재실행 기법을 구현하기 위하여 쓰레드 라이브러리를 사용하며, 현재의 제어를 다른 쓰레드에게 자발적으로 인계하는 *thr_yield( )* 함수를 사용한다. 재실행을 위하여 의미있는 사건의 앞뒤에 각각 프롤로그(prolog) 사건과 에필로그(epilog) 사건을 첨가한다.

4.4 사용자 인터페이스[19]

ParaDebug는 그래픽 사용자 인터페이스(Graphical User Interface : GUI)를 제공한다. 디버거로 부터 제공되는 각각의 명령어들은 윈도우 상에서 메뉴로 표시되고, 사용자는 이 메뉴를 이용하여 자신이 원하는 기능을 선택할 수 있다. 각각의 기능들은 기본적으로 하나의 디버거 요소창(pane)으로 구성되며, 이들은 디버거 윈도우 구성의 가장 기본적인 요소가 된다. 한편 ParaDebug에서는 디버깅되고 있는 정보를 텍스트 형태로 뿐만 아니라, 그래픽의 형태로도 보여줌으로써 디버깅을 쉽게 해준다. 특히, 디버깅되는 프로그램의 호출 그래프, 쓰레드-대-시간 다이어그램, 프로세스 실행 상태 다이어그램 등을 제공하여, 전체적으로 가시적인 디버깅 환경을 제공한다.

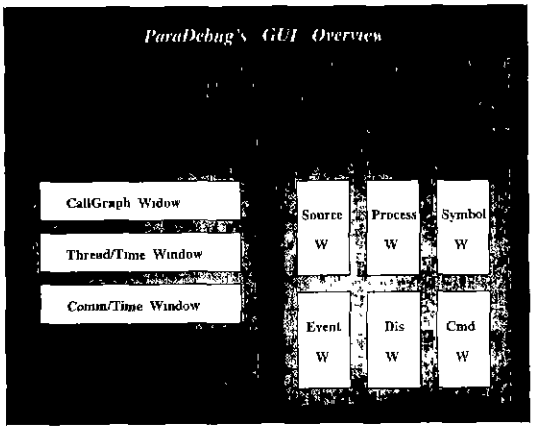


그림 4 ParaDebug의 GUI 구조

ParaDebug의 GUI 환경에서 텍스트/그래프 사상은 디버깅 정보들이 텍스트 형태로 보여지는 윈도우와 그래픽 형태로 제공되는 윈도우를 엮어주는 기능이다. 예를 들어, 텍스트 윈도우 상에서 정지점이 설정된 특정한 함수에 대하여, 전체 프로그램의 실행 경로 상에서 그의 실행 위치 및 상태를 그려 주는 그래픽 윈도우와 매치시킨다.

4.5 ParaDebug를 이용한 디버깅

재실행 기반의 병렬 디버거는 전통적인 순차 디버거와는 다른 사용 특성을 가지고 있다. 즉 재실행을 이용하기 위해 선행 실행 과정이 존재하고, 그 실행 후 재실행 구동기가 프로그램을 동일한 실행 순서로 수행 되도록 프로그램을 수정한 후 디버깅을 진행한다. 사용자가 ParaDebug로 재실행을 이용하여 디버깅을 하려면, ParaDebug 시스템은 자동적으로 프로그램을 참조(reference) 실행하여 재실행에 필요한 정보를 수집한 후 프로그램을 재 실행이 되게 수정하고 디버깅 상태로 들어가 사용자가 병렬 프로그램을 순차 프로그램과 같은 방식으로 디버깅할 수 있게 한다. 그림 5에서 점선으로 표시된 화살표가 실제 디버깅하는 흐름이고 실선으로 된 부분이 ParaDebug 시스템이 자동적으로 수행하는 부분이다.

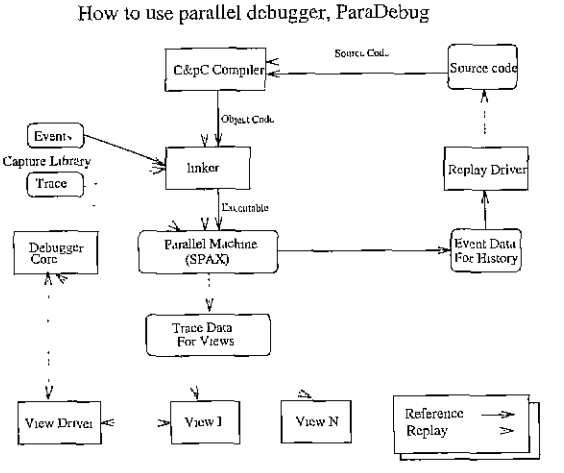


그림 5 ParaDebug의 디버깅 흐름

5. 결 론

본 고에서는 병렬 디버거의 여러 가지 기능들을 분석하고 정리하였다. 병렬 디버거는 순차 디버거와는 달리 경합 조건에 따른 탐침 효과 문제 및 동일한 입력과 실행에 대하여 서로 다른 출력이 나오는 비반복성 문제를 처리해야 하는데, 이러한 문제를 사건 기반형 재실행 기법을 이용하여 해결하는 방법에 대하여 기술하였다. 공유 메모리와 분산 메모리형 병렬 프로그램의 재실행을 위한 사건의 정의와 종류, 저장 방법에 관하여 기술하였고, 이러한 사건을 논리적 타임 스탬프 기법을 기반으로 순서화함으로써 비반복성 문제를 해결하는 과정을 설명

하였다. 이 경우에도, 재실행시 정지점을 설정하여 오류라고 판단되는 부분을 찾고 분석하여, 프로그램을 수정하는 작업을 반복적으로 수행하는 순환적 디버깅 방법의 사용이 가능하게 된다. 병렬 디버거의 사례로 소개한 ParaDebug는 순환 디버깅 기법과 재실행 기법을 기반으로 하는 소스 레벨 디버거로서, 디버깅되는 프로그램의 데이타를 텍스트와 그래픽 형태로 보여주는 텍스트/그래픽 사상 기반형 그래픽 사용자 환경을 제공한다. ParaDebug의 그래픽 사용자 인터페이스는 텍스트/그래픽 사상 기능을 제공함으로써, 디버깅되는 객체들에 대한 상호 관계 이해를 용이하게 하고, 각각의 디버깅 객체에 대한 데이타 값과 실행 상황을 동시에 제공한다. 이는 병렬 프로그램의 실행 오류 디버깅에 필요한 모든 데이타를 한 곳에서 검색할 수 있는 가시적인 통합 뷰를 형성하며, 병렬 프로그램의 오류와 실행 성능을 동시에 디버깅할 수 있는 새로운 차원의 병렬 프로그램 개발 환경을 제시한다.

참고문헌

[1] C.E. McDowell, D.P. Helmbold, Debugging Concurrent Programs, ACM Computing Surveys, Vol.21(4), pp.593-622, Dec. 1989.

[2] J. Yan, S. Sarukkai, P. Mehra, Performance Measurement, Visualization and Modeling of Parallel and Distributed Programs using the AIMS Toolkit, Software - Practice and Experience, Vol.25(4), pp. 429-461, April 1995.

[3] R. Zink, The Stuttgart Parallel Processing Library SPPL and the X Windows Parallel Debugger XPDB, 7th Intn'l Parallel Processing Symposium, Proc. Parallel Systems Fair, pp.1150-1155, Newport Beach, USA, April 1993.

[4] J.T. Stasko, E. Kraemer, A Methodology for Building Application - Specific Visualizations of Parallel Programs, Graphics, Visualization, and Usability Center, Georgia Institute of Technology, Technical Report GIT-GVU-92-10, June 1992.

[5] T.J. LeBlanc, J.M. Mellor-Crummey, Debugging parallel programs with Instant Replay, IEEE Transactions on Computers, C-36(4), pp.471-482, April 1987.

[6] L. Lamport, Time, Clocks and the Ordering of Events in a Distributed System, Communications of the ACM, Vol.21, No. 7, pp.558-565, 1978.

[7] J. Fowler, W.Zwaenepoel, Causal Distributed Breakpoints, Proceedings of the 10th International Conference on Distributed Computing Systems, pp.134-141, 1990.

[8] R.H.B. Netzer, Optimal Tracing and Replay for Debugging Shared-Memory Parallel Programs, Proceedings of the ACM/ONR Workshop on Parallel and Distributed Debugging, pp.1-11, San Diego, California, May 1993.

[9] R.H.B.Netzer and B.P.Miller, Optimal Tracing and Replay for Debugging Message-Passing Parallel Programs, Proceedings of the International Conference on Supercomputing, pp.502-511, MN, Nov. 1992.

[10] L. Gunaseelan, R.J. LeBlanc,Jr., Debugging Objects and Threads in a Shared Memory System, Proceedings of the USENIX Symposium on Experience with Distributed and Multiprocessor Systems, pp.175-194, USENIX Association, Sep. 1993.

[11] K.-C. Tai, R.H. Carver and E.E. Obaid, Debugging Concurrent Ada Programs by Deterministic Execution, IEEE Transactions on Software Engineering, Vol.17, No. 1, pp.45-63, Jan. 1991.

[12] Colin Fidge, Logical time in distributed computing systems, IEEE Computer, pp.28 -33, Aug. 1991.

[13] L. Gunaseelan, R.J. LeBlanc,Jr., Event ordering in a shared memory distributed system, Proceedings of the 13th International Conference on Distributed Computing Systems, pp.256-263, May 1993.

[14] 이범식, 온기원, 홍철의, 지동해, 분산 및 공유 메모리 병렬 컴퓨터용 디버거 설계, 한국정보처리학회 '95 추계학술발표 논문집 제2권 2호, pp.190-195, 10. 1995.

[15] 지동해, 임기욱, TOPS : 고속병렬컴퓨터(주전산기 IV)를 위한 병렬 프로그래밍 도구, UniExpo "95, pp.189-193, 11. 1995.

[16] 김진미, 우영춘, 이경석, 신동하, 이재경, 고속병렬컴퓨터에서 병렬프로그래밍을 위한 C 언어의 확장, 한국정보과학회 '95 추계학술발표 논문집 Vol.22, No.2, pp.861-864, 10. 1995.

[17] UNIX System Laboratories, Inc., UNIX Debugger - Design for Multi threaded Support, Oct. 1993.

[18] 홍철의, 이범식, 온기원, 지동해, MPI 병렬 프로그램 디버깅을 위한 재실행. 한국정보처리학회 '96 춘계학술발표 논문집, 3권 1호, pp.146-149, 4. 1996.

[19] 온기원, 홍철의, 이범식, 그래픽형 병렬 디버거(ParaDebug)를 위한 텍스트/그래픽 사상 기반형 사용자 인터페이스,한국정보처리학회 '96 춘계학술발표 논문집, 3권 1호, pp.150-155, 4. 1996.

온 기 원

1988 Universitat Dortmund, Computer Science, Germany(Vordiplom)
1992 Universitat Dortmund. Computer Science, Germany(Diplom)
1993～현재 한국전자통신연구소 컴퓨터연구단 연구원
관심분야 : parallel processing, hybrid intelligent system

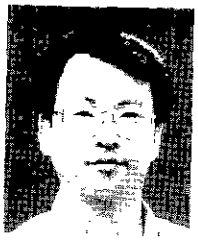
이 범 식

1988 경북대학교 전자공학과(공학사)
1993 연세대학교 전산과학과(공학석사)
1993～현재 한국전자통신연구소 컴퓨터연구단 연구원
관심분야 : 병렬 및 분산처리, 프로그래밍 환경

홍 철 의

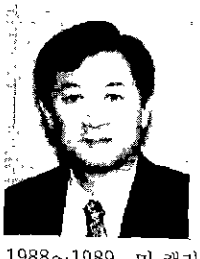
1985 한양대학교 공과대학(공학사)
1989 New Jersey Institute of Technology, Computer Science and Information (공학석사)
1992 University of Missouri, Computer Science(공학박사)
1992～현재 한국전자통신연구소 컴퓨터연구단 선임연구원
관심분야 : 최적화 이론, 병렬 및 분산처리, 프로그래밍 환경

지 동 해



1982 정희대학교(공학사)
1986 Iowa State University (공학석사)
1988 Iowa State University (공학박사)
1988～현재 한국전자통신연구소 병렬프로그램이연구실 실장
관심분야 : 병렬 및 분산처리, 객체지향 프로그래밍 환경

임 기 욱



1977 인하대학교 공과대학 전자공학과 졸업
1986 한양대학교 대학원 전자계산학 석사
1994 인하대학교 대학원 전자계산학 박사
1977～1983 한국전자기술연구소 선임연구원
1986～1988 한국전자통신연구소 시스템소프트웨어연구실장
1988～1989 미 캘리포니아주립대학(Irvine) 방문연구원
1989～현재 한국전자통신연구소 시스템연구부장
관심분야 : 소프트웨어 아키텍쳐, 실시간 데이터베이스시스템, 컴퓨터시스템구조