메시지전달 프로그램의 디버깅을 위한 메시지경합의 탐지

박미영* · 구금서** · 전용기***

1. 서 론

분산된 메모리를 가진 병렬 컴퓨팅 환경에서 메시지의 전달로써 프로세스간의 통신을 수행하는 병렬프로그램을 메시지전달 프로그램이라 한다. 이러한 프로그램에서 병행하게 수행하는 프로세스들이 송신하는 메시지들은 네트워크의 통신 지연이나 프로세스 스케쥴링에 의해서 도착하는 순서가 달라질 수 있다.

메시지전달 프로그램에서 동일한 채널로 두 개 이상의 메시지들이 도착순서가 보장되지 않고 동일한 수신자에게 전송될 수 있을 때 메시지경합이 발생된다. 이러한 경합의 존재는 결정적 수행결과를 보이도록 의도된 프로그램에서 메시지의 도착순서에 따라 서로 다른 수행 결과를 초래하므로, 프로그램의 디버깅을 위해서 우선적으로 탐지되어야 한다.

메시지전달 프로그램의 효과적인 디버깅을 위해서는 존재하는 경합들 중에서 영향을 받지 않은 경합의 탐지가 중요하다. 왜냐하면 영향받지 않은 경합은 영향받은 다른 경합들을 사라지게 하거나 또는 숨어있는 경합의 탐지를 가능하게 하기 때문이다. 영향받지 않은 경합을 효과적으로 탐지하는 방법은 가장 먼저 발생한 경합인 최초경합을 탐지하는 것이다. 왜냐하면 가장 먼저 발생하는 경합은 부분적 순서관계에 의해서 먼저 발생한 경합이 존재하지 않은 경합이기 때문이다.

경합을 탐지하는 기법으로는 경합 탐지정보를 수집하고 분석하는 시기에 따라 정적 분석 기법, 수행후 기법, 수행중 기법 등으로 나눌 수 있다. 정적 분석 기법은 프로그램의 시맨틱을 분석하여 모든 경합을 탐지하는 기법이고, 수행후 기법은 프로그램의 수행을 추적파일에 기록하고, 추적파일을 분석하여 경합을 탐지하는 기법이다. 그리고 수행중 기법은 프로그램 수행 중에 병행성 정보를 생성하고, 이를 이용하여 경합을 탐지하는 기법이다.

경합탐지를 위해 개발된 대표적인 도구로는 mdb, PDT, MAD등이 있다. 이들 도구는 프로그램 수행 중에 발생하는 경합을 탐지하여 보고하지만, 디버깅에 효과적인 최초경합을 탐지하지 못하는 단점을 가지고 있다. 또한 탐지 결과를 보고하는 형태는 텍스트 형태로 제공되거나, 단순히 송수신 사건만을 시각적으로 보여 주는 형태로 제공되어 경합 디버깅에 효과적인 도구는 아니다.

효과적인 경합 디버깅을 위해서는 최초경합을 탐지할 뿐만 아니라, 탐지 결과를 시각화하여 사용자가 경합의 발생을 이해하고 디버깅하는데 도움을 줄 수 있어야 한다.

본 연구는 한국과학재단 목적기초연구(R05-2003-000-12345-0) 지원으로 수행되었음.
* 경상대학교 자연과학대학 컴퓨터과학과 박사과정
** 경상대학교 자연과학대학 컴퓨터과학과 석사과정
*** 경상대학교 자연과학대학 컴퓨터과학과 교수

본 고의 2절에서는 표준화된 메시지전달 모델인 MPI를 소개하고, MPI의 스팩이 그대로 구현되어 널리 사용되고 있는 MPICH를 소개한다. 그리고 3절에서는 이러한 모델로 작성된 병렬프로그램에서 발생하는 메시지경합에 대해 소개하고, 그 종류를 분류해 본다. 4절에서는 경합 탐지를 위해 제시된 기법들의 특징을 살펴보고 5절에서는 실제적으로 구현된 도구들에 대해서 소개한다. 마지막으로 6절에서는 효과적인 경합 디버깅을 위해 제공하는 여러 시각화 기법들을 소개하고 7절에서 결론을 내린다.

2. MPI

병렬 컴퓨팅에서 병렬 연산 수행을 위한 다양하고 효율적인 패러다임은 메시지전달 모델이다. 대표적인 메시지전달 모델로는 PVM[7]과 MPI [14]이다. PVM은 MPI보다 먼저 개발되어 사용되고 있으나, 다른 메시지전달 라이브러리에 비해 성능이 뒤떨어지는 단점을 가지고 있다. 반면에 MPI는 높은 이식성과 성능으로 사용자에게 풍부하고 다양한 라이브러리를 제공한다.

MPI는 연구기관, 학교기관, 산업 기관들로 구성된 협회에서 만든 메시지전달 라이브러리의 표준으로서, 현재 과학자와 엔지니어링들 사이에서 가장 보편화된 고수준 메시지전달 시스템이다. MPI는 프로세스가 메시지를 보냄으로써 서로 통신하는 명시적인 메시지전달 패러다임을 가지고 있다. 첫 번째 버전 MPI-1은 1994년에 발표되었고, 기능을 보강한 MPI-2는 1997년에 발표되었다.

MPI는 프로세스간의 통신을 위해서 블록킹(blocking)과 넌블록킹(nonblocking) 등과 같은 통신 타입[2]에 따라 다양한 일대일 통신 함수를 제공한다. 일대일 통신 함수에는 크게 송신 사건과 수신사건으로 구성되는데, 수신사건으로는 블로킹 과 넌블록킹 두 가지가 제공된다. 일대일 통신 함수에서 블록킹과 넌블록킹의 구분은 송수신 사건의 복귀에 대한 송수신의 버퍼 재사용를 알리는 신호발생 여부에 따라 이루어진다. 송신사건의 모드로는 표준 모드(standard mode), 버퍼 모드(buffered mode), 동기화 모드(synchronous mode), 준비 모드(ready mode) 등이 제공된다.

표준 모드는 대응되는 수신이 먼저 발생하지 않아도, 송신을 시작하거나 완료될 수 있다. 그러나 이 모드에서 성공적인 송신의 여부는 대응되는 수신 사건의 발생 여부에 의존적이다. 표준 모드에 해당하는 블록킹 송신함수는 MPI_Send()이고 넌블록킹 송신함수는 MPI_Isend()이다. MPI_Send()는 메시지가 전송될 때까지 MPI_Send 함수에서 블록킹되는 반면에 MPI_Isend()는 블록킹되지 않고 송신 이후의 명령을 수행하게 된다.

버퍼 모드에서의 송신도 수신 발생 여부와 상관없이 송신을 시작하거나, 수신 발생 이전에 송신을 완료할 수도 있다. 이 모드에서 성공적인 송신의 여부는 나가는 메시지의 지역적 버퍼화 여부에 결정된다. 버퍼 모드에 해당하는 블록킹 송신함수는 MPI_Bsend()이고 넌블록킹 송신함수는 MPI_Ibsend()이다.

준비 모드의 송신은 오직 대응되는 수신이 발생된 후에만 송신이 시작되며, 수신이 발생하지 않으면 이 송신 사건은 에러로 취급된다. 준비 모드에 해당하는 블록킹 송신함수는 MPI_Rsend()이고 비블록킹 송신함수는 MPI_Irsend()이다.

동기적 모드에서의 송신은 대응되는 수신의 발생 여부와 상관없이 송신을 시작하지만, 오직 대응되는 수신이 발생되어야만 성공적인 송신을 완료하게 된다. 동기 모드에 해당하는 블록킹 송신함수는 MPI_Ssend()이고 비블록킹 송신함수는 MPI_Issend()이다.

2.1 MPICH

MPI는 병렬컴퓨터 공급업체, 라이브러리 개발자, 애플리케이션 전문가들로 구성된 MPI 포럼에서 정의한 메시지전달 라이브러리에 대한 표준안이다. 현재까지 다수의 MPI가 구현되었으나 MPI 스팩을 그대로 구현한 것이 MPICH[8]이다. MPICH는 이식성과 고성능을 목표로 구현된 MPI 라이브러리이며, 'Chameleon'을 나타내는 MPICH의 'CH'는 환경의 적응성 또는 이식성을 의미한다.

MPICH는 이전에 존재하던 안정적인 라이브러리를 바탕으로 매우 빠르고 안정적으로 개발되었다. 현재 MPICH는 송신의 취소를 포함한 MPI의 모든 표준안을 구현하였으며, TCP/IP 프로토콜을 사용하여 분산 메모리 시스템뿐만 아니라 SMP-cluster 시스템도 지원한다.

그림 1은 구현된 MPICH의 계층적 구조를 나타내고 있다. 그림에서와 같이 MPICH는 세 개의 계층, 즉 MPI 라이브러리, ADI, 채널 인터페이스로 구성된다. MPICH가 소프트웨어적으로 이식성과 성능을 동시에 높일 수 있는 기본 메카니즘은 ADI(Abstract Device Interface)이다. MPICH는 MPI에서 필요한 기본 기능들만 추출하여 ADI라는 인터페이스를 만들고, 이 ADI에서 더욱 기본적인 기능들만 추출하여 채널 인터페이스(channel interface)를 정의하는 계층적인 구조로 구현되었다. 즉, 채널 인터페이스의 적은 루틴들을 컴퓨터에 종속적인 루틴들로 구현하므로, 구조적으로 독립적인 부분을 다시 구현하지 않고도 MPICH를 쉽게 이식할 수 있다. MPICH의 하위 계층으로서 구현된 채널 인터페이스는 최소한 5개의 함수를 가지고 있는 매우 작은 인터페이스이며 MPICH가 새로운 환경에 가장 빠르게 포팅(porting) 되도록 해준다.

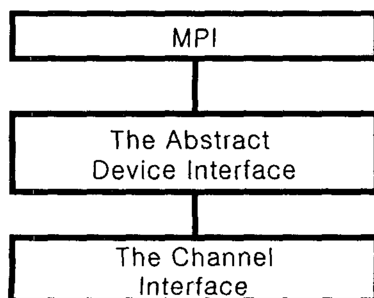

그림 1. MPICH 구조

MPI에서 수행되는 병렬 프로세스들은 양의 정수로 식별되는데, 만약에 P개의 프로세스가 존재한다면, 그들의 식별 값은 0, 1, ..., P-1이다. 모든 MPI 프로그램은 #include "mpi.h" 문을 포함해야 하는데, 이때 mpi.h 파일은 MPI 프로그램을 컴파일할 때 필요한 정의, 매크로, 함수 원형 등을 가지고 있다. 그리고 다른 MPI 함수를 부르기 전에 반드시 MPI_Init 함수를 처음에 한번은 호출해야 한다. 왜냐하면, MPI_Init 함수는 MPI 프로그램을 수행하기 위한 환경을 초기화하기 때문이다. 마지막으로 MPI 프로그램을 마칠 때에는 MPI_Finalize 함수를 호출해야 한다.

그림 2는 MPICH로 작성된 간단한 예제 프로그

```
#include <stdio.h>
#include "mpi.h"

int main( argc, argv )
int argc;
char **argv;
{
   int rank, size;
   MPI_Init( &argc, &argv );
   MPI_Comm_size( MPI_COMM_WORLD, &size );
   MPI_Comm_rank( MPI_COMM_WORLD, &rank );
   printf( "Hello world from process %d of %dWn", rank, size );
   MPI_Finalize();
   return 0;
}
```

그림 2. MPI 프로그램 예

램이다. 이 프로그램은 수행 중인 각 프로세스가 "Hello world from process i of n" 메시지에 자신의 식별 값(i)인 정수를 출력하는 프로그램이다. 그림 3은 예제 프로그램의 컴파일과 수행 결과를 보여 주고 있다. C 언어로 작성된 MPICH 병렬프로그램을 컴파일 할 때의 명령은 "mpicc"이다. 그림 3에서 "mpicc -o helloworld helloworld.c" 명령은 helloworld.c 파일을 컴파일하여 실행파일은 helloworld로 생성하라는 의미이다. 그리고 MPICH에서 실행파일을 병렬로 수행하기 위한 명령은 "mpirun"이다. 그림 3에서 "mpirun -np 4 helloworld"는 helloworld 실행파일을 4개의 프로세스에서 수행하라는 의미이다. 수행결과는 그림 3에서 살펴볼 수 있다.

```
% mpicc -o helloworld helloworld.c
% mpirun -np 4 helloworld
Hello world from process 0 of 4
Hello world from process 3 of 4
Hello world from process 1 of 4
Hello world from process 2 of 4
%
```

그림 3. MPI program의 컴파일과 수행 예

3. 메시지 경합

메시지전달 프로그램의 수행은 병행하게 수행하는 프로세스들이 동시에 메시지를 전송하기 때문에, 프로세스의 스케줄링이나 네트워크의 지연시간 등에 영향을 받아서 의도되지 않은 비결정적인 수행을 보이기도 한다. 이러한 비결정적 수행의 주된 원인 중의 하나가 메시지경합(message-races)[6,9,13]이다. 메시지경합은 두 개 이상의 송신자가 논리적으로 같은 채널을 통해 수신자에게 메시지를 보낼 때 발생된다.

메시지전달 프로그램의 비결정적 수행을 야기하는 메시지경합의 예를 그림으로 나타내면 그림 4와 같다. 그림에서 P_1, P_2, P_3은 병행으로 수행중인 프로세스들을 나타내며, 각 정점은 송신과 수신 사건들을 나타내고 프로세스간의 화살표는 메시지 전달을 나타낸다. 그림 4에서 P_1과 P_3의 두 송신사건들은 서로 병행하게 수행되기 때문에 전송 중인 메시지들은 P_2에 있는 하나의 수신사건으로 서로 경합하게 되며, 스케줄링이나 네트워크의 지연시간 등에 따라 P_2에 도착하는 메시지의 도착순서가 비결정적이다. 이때 P_2에 도착하는 메시지 순서에 따라 프로그램 수행결과도 다르게 나타날 수 있다.

메시지전달 프로그램에서 발생되는 메시지경합을 영향관계 측면, 프로세스관계 측면, 순서관계 측면에서 분류해 볼 수 있다. 먼저, 발생한 메시지 경합들간의 영향관계에 따라 영향받은 경합(affected race)과 영향받지 않은 경합(unaffected race)으로 분류할 수 있다. 영향받은 경합은 먼저 발생한 경합으로부터 영향을 받아 발생되는 경합으로서, 이전에 발생한 경합이 디버깅되면 자동적으로 사라질 수 있는 경합들이다. 반면에 영향받지 않은 경합은 다른 메시지경합으로부터 영향을 받지 않고 발생되는 메시지경합이며 비결정적 수행의 직접적인 원인이다.

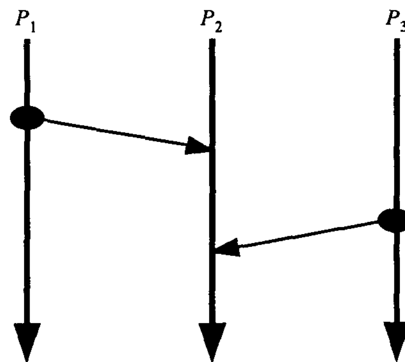

그림 4. 메시지 경합

그림 5는 영향받은 경합과 영향받지 않은 경합의 예를 보이기 위한 메시지전달 프로그램의 부분 수행을 보인다. 이 프로그램의 올바른 수행결과를 위해서는 프로세스 P_3의 메시지가 프로세스 P_2에 먼저 도착하고, 다음으로 프로세스 P_1의 메시지가 도착해야 한다고 하자. 그러나 P_1과 P_3은 같은 채널을 통해 P_2로 메시지를 송신하고, P_2의 수신 사건은 임의의 메시지를 수신한다고 했을 때, P_2의 도착하는 메시지들의 순서는 보장되지 않는다. 만약에 P_1에서 보낸 메시지가 먼저 도착하고 P_3에서 보낸 메시지가 나중에 도착하는 경우에는 (x = 1)의 조건이 성립되어 P_2에서 송신사건이 발생하여 P_3에서 경합이 발생하게 된다. 그러나 의도한 순서대로 P_3의 메시지가 도착한 후에 P_1의 메시지가 도착하는 경우에는 P_2에서 송신사건이 발생하지 않아 P_3에서는 경합이 발생하지 않는다. 즉 P_2에 존재하는 경합은 다른 경합으로부터 영향을 받지 않은 경합으로서, 매 수행시마다 나타나는 오류이므로 반드시 디버깅이 요구된다. 그러나 P_3에서 발생하는 경합은 P_2에서 발생한 경합으로부터 영향을 받은 경합으로서, 이러한 경합은 P_2의 경합이 디버깅되면 자연히 사라지는 경합이므로 탐지가 필요하지 않다.

따라서 영향받은 경합과 영향받지 않은 경합들 중에서 디버깅을 위해서 반드시 탐지가 요구되는 경합은 영향받지 않은 경합이다. 영향받지 않은 경합을 가장 효율적으로 탐지하는 방법은 모든 경합들 중에서 가장 먼저 발생한 경합을 탐지하는 것이다. 왜냐하면 가장 먼저 발생하는 경합은 부분적 순서관계에 의해서 먼저 발생한 경합이 존재하지 않는 경합이기 때문이다.

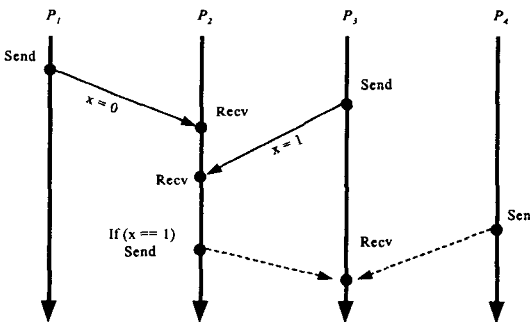

그림 5. 메시지 경합

메시지전달 프로그램에서 가장 먼저 발생하는 경합인 최초경합은 프로세스간의 관계에 따라 지역적 최초경합과 전역적 최초경합으로 분류할 수 있다. 지역적 최초경합은 한 프로세스 내에서 가장 먼저 발생한 경합으로서, 동일한 프로세스 내에서 발생한 경합들 중에서 가장 먼저 발생한 경합이다. 이러한 지역적 최초경합은 해당 프로세스 내에서는 영향받지 않은 경합이지만, 메시지 송수신으로 형성되는 다른 프로세스와의 관계에 의해서 다른 프로세스에서 발생한 경합으로부터 영향을 받을 수도 있다. 전역적 최초경합은 수행 중인 모든 프로세스간의 관계를 고려해서 가장 먼저 발생한 경합으로, 이 경합은 영향받지 않은 경합임을 보장할 수 있다.

예를 들어, 그림 5에서 지역적 최초경합으로 P_2, P_3에서 발생한 경합이고, 이들 중에서 P_3에서 발생한 지역적 최초경합은 P_2로부터 영향받은 경합이다. 그리고 그림 5에서 전역적 최초경합으로는 P_2에서 발생한 경합만 해당한다.

메시지전달 프로그램에서 어떤 경합은 다른 프로세스에서 발생한 경합으로부터 영향을 받으면서 자신도 그 경합으로 영향을 주기도 한다. 이와 같이 경합들이 서로 영향을 주고받는 경우는 그들간의 순서관계가 존재하지 않으므로 이들을 하나의 얽힘(tangle)[12]으로 탐지해야 한다. 얽힘을 구성하는 경합의 수에 따라 단일 얽힘과 다중 얽힘으로 분류할 수 있다. 단일 얽힘은 하나의 경합만으로 구성되는 얽힘으로서 앞서 보인 예에 나타

난 경합들이 이에 해당한다. 반면에 여러 개의 경합들로 구성되는 얽힘을 다중 얽힘이라 하는데 이는 그림 6과 같은 경우이다. 그림 6에서 프로세스 P_2와 P_5에 메시지경합이 존재하는데, 이 두 경합이 서로 영향을 주고받으면서 발생한다. P_2에서는 P_1과 P_3으로부터의 메시지들이 경합하고, P_5에는 P_4와 P_6으로부터의 메시지들이 경합한다. 두 경합이 서로에게 영향을 미치는 형태는 다음과 같다. 메시지경합이 있는 P_5에서 P_3으로의 송신사건 이후에, P_3의 송신으로 P_2에서 메시지경합이 발생된다. 그리고 경합이 있는 P_2에서 P_4로의 송신사건 이후에, P_4에서 P_5로의 송신으로 P_5에서 메시지경합이 발생된다.

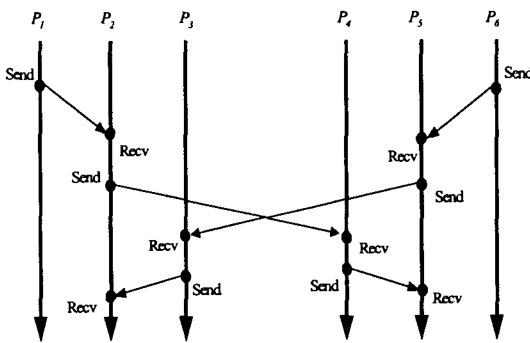

그림 6. 얽힌 경합

4. 경합 탐지 기법

메시지전달 프로그램에서 동적으로 경합을 탐지하는 기법은 탐지시점에 따라서 수행후(post-mortem) 탐지기법[15,16]과 수행중(on-the-fly) 탐지기법[3,13]으로 나눌 수 있다. 수행후 탐지기법은 프로그램을 수행하면서 수행정보를 추적파일에 기록하고, 프로그램의 수행이 끝난 후에 추적파일을 분석하여 경합을 탐지한다. 이 기법은 한 번의 수행에서 발생한 모든 경합을 탐지하지만, 소규모의 병렬프로그램에서도 추적파일을 위한 기억공간의 요구가 비현실적으로 크다. 수행중 탐지기법은 프로그램의 수행을 감시하면서 수신사건을 수행할 때마다 경합의 발생여부를 검사한다. 이 기법은 감시되는 프로그램에 경합이 존재한다면 적어도 한 개의 경합은 반드시 탐지하며, 경합탐지를 위해 사용되는 기억공간을 수행 중에 재사용하므로 대규모의 병렬프로그램에서도 적용할 수 있는 현실적인 기법이다.

수행중 탐지기법은 탐지되는 경합의 성격에 따라서 경합의 존재 여부를 검증하는 기법[3,13]과 영향받지 않은 경합을 탐지하는 기법[4,6,9,12]으로 나눈다. 경합검증 기법은 매 수신사건마다 이전 수신사건과 현재 송신사건간의 병행성을 검사하여 경합을 탐지한다. 그러나 이러한 기법은 경합들간의 영향관계를 고려하지 않고 수행 중에 존재하는 모든 경합을 대상으로 하기 때문에 디버깅에는 효과적이지 않다. 영향받지 않은 경합을 탐지하는 기법은 매 수신사건마다 경합의 발생여부를 검사하고, 경합이 발생한 경우에는 해당 프로세스에서 가장 먼저 발생한 지역적 최초경합인지를 검사한다. 탐지된 경합은 그 프로세스내에서 발생된 경합으로부터는 영향받지 않은 경합임을 보장하므로 경합 디버깅에 효과적이다.

메시지전달 프로그램의 수행 중에 영향받지 않은 경합을 탐지하고자 하는 기존의 기법은 프로그램 감시작업의 병렬성에 따라 OtOt(One-thread-at-One-time) 기법[4,6]과 MtOt (Multi-threads-at-One-time) 기법[9, 12]으로 구분된다. OtOt 기법은 한 번의 수행에서 하나의 프로세스만을 감시하여 지역적 최초경합만을 탐지하기 때문에, 적어도 프로세스 수 만큼을 반복해서 수행해야 한다. 그러나 MtOt 기법은 한 번의 수행으로 모든 프로세스를 감시하여 지역적 최초경합을 탐지하는 기법으로서, 필요한 수행 횟수에 따라서 1-Pass 기

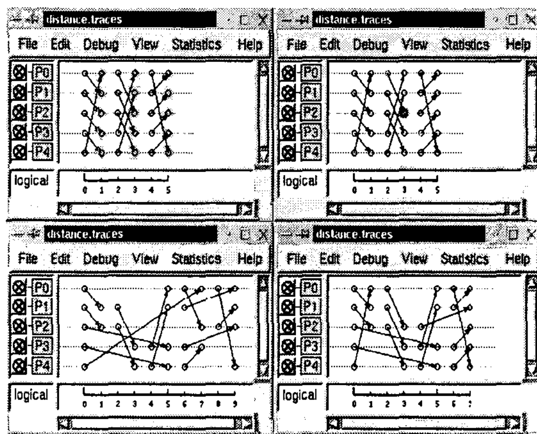

그림 7. MAD

법[9]과 2-Pass 기법[12] 기법으로 분류된다. 1-Pass 기법은 매 수신사건마다 이전에 발생한 모든 수신사건을 검사하여 관련된 경합들을 모두 탐지하기 때문에, 메시지 수에 비례하는 복잡성을 보이므로 비현실적이다. 반면에 2-Pass 기법은 첫 수행에서 프로세스별로 가장 먼저 발생하는 경합인 지역적 최초경합(locally-first race)를 위한 정보를 탐지하고, 두 번째 수행에서는 지역적 최초경합이 발생한 수신사건에서 해당 프로세스의 수행을 중단하여 경합을 탐지한다. 이러한 2-Pass 기법은 메시지의 수와 무관한 시간 및 공간 복잡도를 가지므로, 1-Pass 기법에 비해서 효율적이다.

5. 메시지경합 탐지 도구

메시지 경합을 탐지하는 도구로는 mdb, PDT, MAD등이 있다. 각 도구별 특징을 살펴보면 다음과 같다.

Damodaran-Kamal이 제안한 mdb[5]는 PVM 병렬프로그램에서 최초로 수행 중에 메시지경합을 탐지한 도구이다. 이 도구는 메시지경합을 탐지하기 위해 제어된 수행(controlled execution) 기법을 사용한다. 제어된 수행 기법은 병렬로 수행 중인 프로세스들 중에서 하나의 프로세스 수행을 감시하면서 수신 큐에 저장된 메시지들을 비교하여 경합을 보고한다. mdb는 최초로 메시지경합을 탐지한 도구이지만, 최초경합을 탐지하지 못하며, 문자 위주의 기법을 이용한 텍스트 형태 시각정보를 제공한다. 따라서 효율적인 디버깅할 수 있는 시각화 기능을 제공하지 못하기 때문에 메시지경합의 디버깅 부담이 크다.

Christian이 제안한 PDT[1]는 HPF(High Performance Fortran)과 MPI 을 이용한 병렬프로그램에서 메시지경합을 탐지하는 도구로서, 통합환경 도구인 Annai에 내장되어 있다. 이 도구는 메시지경합 탐지를 위해 Netzer와 Miller가 제안한 알고리즘[13]을 사용하고 있으나, 이 알고리즘은 메시지경합의 직접적인 정보를 제공하지 않고 프로그램의 재 수행을 위한 보조정보의 확보에 목적이 있다. 따라서 PDT 디버거 도구도 최초경합을 탐지하지 못하며, 효과적인 시각화 기능을 제공하지 못한다.

Kranzlmuller 외 2인이 제안한 MAD[10,11]는 MPI 병렬프로그램에서 메시지경합을 탐지하는 도구이다. 이 도구는 경합을 탐지하기 위한 어떠한 알고리즘도 사용하지 않고, 하나의 수신사건으로 경합하는 모든 메시지들을 시각화한다. 즉, MAD는 직접적으로 경합을 탐지하여 보고하지는 않지만, 이러한 시각화 기능을 통해서 간접적으로 프로그래머가 경합 및 최초경합을 판별하도록 유도하는 디버거 도구이다. 그림 7은 MAD가 제공하는 시각화의 예이다. 그림 7에서 알 수 있듯이 MAD는 시간흐름도 기법을 통해서 각 프로세스별 송수신 사건의 관계를 나타내고 있다.

6. 메시지경합의 시각화

시각화는 프로그램의 복잡한 수행에 대한 이해

를 돕고, 문제를 빠르게 인식할 수 있도록 수행형태를 그림으로 표현하는 기법으로 과학 및 공학분야에서 널리 사용되고 있다. 특히 병렬 프로그래밍 분야에서도, 복잡한 프로세스들의 수행을 이해하고 효과적으로 디버깅하기 위해서 시각화 기법을 사용한다. 그러나, 병렬프로그램의 시각화는 이전의 순차프로그램의 시각화보다 매우 어렵다. 왜냐하면, 컴퓨터 과학 분야에서 프로세스와 같은 대부분의 개념들은 추상적이기 때문에 사람에게 쉽게 받아질 수 있는 형태로 시각화하는 것은 어렵기 때문이다.

특히 시각화에서는 화려하게 시각적으로 나타내기보다는 입력된 정보로부터의 정확한 시각적 표현과 프로그램 수행에 대한 이해 등이 더욱 중요하다. 이러한 시각화로서 요구되는 조건은 다음과 같다. 첫째, 시각화는 이미 이해된 내용을 표현할 수 있을 뿐만 아니라 프로그래머가 미처 알지 못한 내용을 발견할 수 있도록 유도할 수 있어야 한다. 둘째, 확장성이 있어야 한다. 셋째, 시각화에서 나타나는 색(color)도 정보의 형태로 사용되어야 한다. 넷째, 시각화는 사용자와 상호작용이 가능해야 한다. 다섯째, 표현된 시각화에는 유용한 텍스트 형태의 설명이 덧붙여져야 한다. 여섯째, 시각화 정보를 사용하는 사용자 인터페이스가 간단해야 한다.

디버거의 시각화는 프로그래머가 프로그램의 수행을 이해하고, 시각정보를 통해 오류를 발견하도록 도와준다. 병렬프로그램에서의 일반적인 시각화 모델은 문자 위주(textual) 기법, 시간 흐름도(time-line) 기법, 그래프(graph) 기법, 애니메이션 기법, 다중 윈도우 기법 등이 있다. 문자위주 기법은 제어흐름 정보의 표시나 디버깅에 필요한 자료를 단순하게 문자 형태로 제공하는 것이며, 시간 흐름도 기법은 시간과 프로세스로 구성된 이차원 형태의 시각화를 통해 프로세스들의 수행 과정을 보여주는 기법이다. 그리고 수행 중인 프로세스들의 관계와 그들간의 통신 형태를 보이기 위해서는 그래프 기법 등을 이용한다. 애니메이션 표현은 디스플레이의 이차원에 의해 공간적으로 표현된다. 이때 하나의 단일 인스턴스(instance)는 프로그램 상태의 단편(snapshot)에 해당하며, 이것은 연속적으로 또는 단편적으로 하나씩 보게 할 수도 있다. 다중 윈도우 표현은 프로그램이 디버깅되고 있는 화면을 동시에 여러 화면으로 보는 것을 가능하게 한다.

7. 결 론

본 고에서는 메시지전달 프로그램에서 발생하는 메시지경합과 이러한 경합을 탐지하기 위한 기법 및 도구에 대해서 소개하였다.

메시지 경합은 동일한 채널로 두 개 이상의 메시지들이 수신자에게 도착순서가 보장되지 않고 전송될 수 있을 때 발생한다. 이때 경합하는 메시지들의 비결정적인 도착순서는 프로그램의 의도되지 않은 비결정적인 수행 결과를 초래하므로, 효과적인 디버깅을 위해서 메시지경합은 반드시 탐지되어야 한다.

경합을 탐지하는 기법은 크게 정적 분석 기법, 수행후 탐지 기법, 수행중 탐지 기법등으로 구분될 수 있으며, 이들 중에서 수행중 탐지 기법이 가장 효율적이다. 왜냐하면 프로그램 수행 중에 경합을 탐지하기 때문에 실제적인 경합을 탐지할 뿐만 아니라 경합이 존재한다면 최소한 하나의 탐지는 보장하기 때문이다.

경합을 탐지하는 도구로는 mdb, PDT, MAD등이 있지만, 영향받은 경합과 영향받지 않은 경합에 대한 구분 없이 탐지하므로 효과적인 디버깅 도구로의 역할은 부족하다. 효과적인 경합 디버깅

을 위해서는 실제적인 오류인 영향받지 않은 경합을 탐지하고, 병렬 프로그램의 복잡한 수행형태 및 경합 탐지 결과를 효과적으로 시각화하는 도구가 요구된다.

참 고 문 헌

[1] Clemencon C., J. Fritscher, M. J. Meehan and R. Ruhl, *An Implementation of Race Detection and Deterministic Replay with MPI*, TR-95-01, CSCS, January 1995.

[2] Cypher, R., and E. Leu, "The Semantics of Blocking and Nonblocking Send and Receive Primitives," *8th IEEE Intl. Parallel Processing Symp.*, pp. 729-735, IEEE, Apr. 1994.

[3] Cypher, R., and E. Leu, "Efficient Race Detection for Message-Passing Programs with Nonblocking Sends and Receives," *7th IEEE Symp. on Parallel and Distributed Processing*, pp. 534-541, IEEE, San Antonio, Texas, 1995.

[4] Damodaran-Kamal, S. K., and J. M. Francioni, "Nondeterminacy: Testing and Debugging in Message Passing Parallel Programs," *ACM/ONR Workshop on Parallel and Distributed Debugging*, Sigplan Notices, 28(12): 118-128, ACM, Dec. 1993.

[5] Damodaran-Kamal S. K. and J. M. Francioni, "mdb: A Semantic Race Detection Tool for PVM," *Scalable High-Performance Computing Conference*, pp. 702-709, IEEE, May 1994.

[6] Damodaran-Kamal, S. K. and J. M. Francioni, "Testing Races in Parallel Programs with an OtOt Strategy," *Int'l Symp. on Software Testing and Analysis*, pp. 216-227, ACM, Aug. 1994.

[7] Geist, A., A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. "PVM: Parallel Virtual Machine," *A Users' Guide and Tutorial for Networked Parallel Computing*, Cambridge, MIT Press, 1994.

[8] Gropp, W. and E. Lusk, *User's Guide for Mpich, A Portable Implementation of MPI*, TR-ANL-96/6, Argonne National Laboratory, 1996.

[9] Kilgore, R. and C. Chase, "Re-execution of Distributed Programs to Detect Bugs Hidden by Racing Messages," *30th Annual Hawaii Int'l. Conference on System Sciences (HICSS)*, Vol. 1, pp. 423-432, Jan. 1997.

[10] Kranzlmuller D., N. Stankovic, J. Volkert, "Debugging Parallel Programs with Visual Patterns," *Symp. on Visual Languages*, IEEE, September, 1999.

[11] Kranzlmuller D., C. Schaubschlager, J. Volkert, "A Brief Overview of the MAD Debugging Activities," *4th Intl. Workshop on Automated Debugging*, August 2000.

[12] Netzer, R. H. B., T. W. Brennan, and K. D. Suresh, "Debugging Race Conditions in Message-Passing Programs," *SIGMETRICS Symp. on Parallel and Distributed Tools (SPDT)*, ACM, May 1996.

[13] Netzer, R. H. B., and B. P. Miller, "Optimal Tracing and Replay for Debugging Message-Passing Parallel Programs," *Supercomputing*, pp. 502-511, IEEE/ACM, Nov. 1992.

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

[15] Tai, K. C. "Reachability Testing of Asynchronous Message-Passing Programs," *Int'l. Symp. on Software Engineering for Parallel and Distributed Systems*, IEEE, pp. 50-61, IEEE, May 1997.

[16] Tai, K. C. "Race Analysis of Traces of Asynchronous Message-Passing Programs," *Int'l. Conf. Distributed Computing Systems (ICDCS)*, pp. 261-268, IEEE, May 1997.

박 미 영

- 1999년 동서대학교 컴퓨터공학과 공학사
- 2001년 경상대학교 컴퓨터과학과 공학석사
- 2001년 ~ 경상대학교 컴퓨터과학과 박사과정
- 관심분야 : 운영체제, 병렬 프로그램 디버깅, 클러스터 컴퓨팅
- E-mail: park@race.gsnu.ac.kr

구 금 서

- 2003년 경상대학교 컴퓨터과학과 학사졸업
- 2003년 ~ 경상대학교 컴퓨터과학과 석사과정
- 관심분야 : 운영체제, 병렬 프로그램 디버깅, 클러스터 컴퓨팅
- E-mail: goodman@race.gsnu.ac.kr

전 용 기

- 1980년 경북대학교 컴퓨터공학과 학사
- 1982년 서울대학교 컴퓨터공학과 석사
- 1993년 서울대학교 컴퓨터공학과 박사
- 1982년 ~ 1985년 한국전자통신연구원 연구원
- 1995년 ~ 1996년 미국 캘리포니아 주립대(UCSC) 컴퓨터과학과 연구원
- 1985년 ~ 경상대학교 컴퓨터과학과 교수
- 관심분야 : 병렬 및 분산 처리, 운영체제, 프로그래밍 환경
- E-mail: jun@gsnu.ac.kr