공유메모리와 메시지전달이 혼합된 병렬 프로그램의 디버깅을 위한 경합탐지

박 소 희*

1. 서 론

병렬프로그래밍은 처리해야 하는 작업의 대형화로 인해 단일 프로세서가 처리하기 힘들거나, 작업의 처리 속도가 저하되는 것을 극복하기 위해 요구되어진 방법이다. 즉, 하나의 작업을 처리하기 위해 동시에 여러 개의 프로세서를 사용하여 작업의 처리 속도를 높이고자 하는 것이 목적이다. 병렬 프로그램의 모델은 크게 공유메모리(shared-memory)기반[5,13]과 분산메모리(distributed-memory)기반[8]의 프로그램으로 나눌 수 있으며, 최근에는 이를 혼합하여 분산공유메모리(distributed shared memory)[2-4,10,16]에서 프로그래밍하는 것이 주된 경향이 되고 있다.

공유메모리 프로그램 유형의 대표적인 예는 산업 표준인 OpenMP(Open Multi-Processing)가 있으며, 메시지전달 프로그램 유형의 대표적인 예는 산업 표준인 MPI(Message Passing Interface)가 있다. OpenMP는 구현이 쉽고, 작은 단위의 병렬성(fine-grain parallelism)을 가지는 프로그램의 수행 시에 효율적이지만, 공유메모리 환경을 기반으로 개발되어 분산메모리 시스템에서 사용할 수 없는 단점이 있다. MPI는 분산메모리 환경을 기반으로 개발되어, 각 노드의 지역메모리 자료를 메시지 송·수신을 통하여 복사해야 하므로 지나치게 작은 단위의 병렬성을 가지는 프로그램 수행 시에 통신 부담이 매우 큰 단점이 있다. 그러므로 최근에는 병렬프로그램의 수행 성능을 개선하기 위해서 SMP(Symmetric Multi-Processor) 클러스터에서 MPI와 OpenMP를 혼합한 형태[2-10,16](mixed mode)로 프로그래밍하는 것이 일반적이다. 이러한 SMP 클러스터 시스템은 분산메모리를 사용하면서 각 노드 내에서는 공유메모리를 사용하게 된다. OpenMP/MPI의 혼합형 프로그램 모델은 MPI와 OpenMP로 각각 프로그래밍하여 수행한 것보다 효율적이며, 특히 MPI 프로그램이 부하 불균등을 보이거나, 작은 단위의 프로그램 수행으로 인해 확장성이 저하되거나, 자료의 복사 혹은 MPI 단위 프로세스 수의 제약으로 인해 메모리의 한계를 겪는 경우에 효율적인 장점이 있다.

공유메모리와 메시지전달(message-passing)을 혼합하여 사용하는 병렬 프로그램은 논리적인 오류 외에도 경합조건(race conditions) 등의 병렬 오류로 인한 비결정적인(non-deterministic) 수행을 초래하므로 기존의 순차프로그램보다 디버깅이 어렵다. 이러한 비결정적인 수행 결과의 주된 원인인 경합은 병렬로 수행되는 스레드들이 적절한 동기화 없이 공유메모리에 적어도 하나 이상의 쓰기 사건으로 동시에 접근하거나, 혹은 스레드들 간의 메시지전달 시에 결정적인 순서화를 보장하지 않는 경우에 발생한다. 이러한 병렬

* 경성대학교 교양과정부 전임강사

프로그램 오류를 공유메모리 프로그램의 경우에 자료경합(data race)[1,6]이라 하고, 메시지전달 프로그램의 경우에 메시지경합(message race)[8, 11]이라 한다. 경합 조건에 의한 프로그램의 비결정적인 수행은 프로그래머가 의도하지 않은 결과를 초래할 수 있으므로 병렬프로그램의 디버깅에 있어서 중요한 문제이다.

경합 조건을 탐지하기 위한 현실적인 기법으로, 프로그램 수행 중에 경합탐지에 필요한 정보만을 유지하는 수행중 경합탐지 기법(on-the-fly race detection)[1,6,11,12]이 있다. 수행중 경합탐지 기법은 수행 중에 경합을 탐지하여 보고하기 위한 경합탐지 프로토콜[6,12]과 경합탐지 프로토콜의 적용 시에 병행사건 간의 논리적인 병행성 여부를 판단하기 위한 병행성 정보 생성 기법[1,6]으로 구성된다. 각 스레드들은 병행성 정보에 해당하는 유일한 값인 레이블(label)이 있어야만 수행사건 발생 시에 스레드들 간의 논리적 병행성을 검사하여 경합을 탐지하고 보고할 수 있다. 레이블은 수행 중에 동적으로 생성 가능해야 하며, 어떠한 프로토콜을 적용하더라도 경합탐지가 가능해야 한다.

본 논문에서는 공유메모리와 메시지전달이 혼합된 병렬 프로그램에서 디버깅을 위한 경합탐지에 대해 살펴본다. 먼저, 그러한 병렬프로그램의 모델에 대해 소개하고, 각 모델에 따른 경합조건의 유형을 보인다. 그리고, 그러한 각 경합조건을 탐지하기 위해 제안된 기존의 기법들을 그 효율성과 함께 소개한다. 마지막으로 그러한 프로그램 모델의 경합을 효율적으로 탐지하기 위해서 개발된 탐지 기법의 사례를 소개한다.

2. 병렬 프로그램의 모델

병렬프로그램의 모델은 크게 공유메모리 기반과 분산메모리 기반의 프로그램으로 나눌 수 있으며, 최근에는 이를 혼합하여 프로그래밍하는 것이 주된 경향이 되고 있다. 공유메모리를 기반으로 하는 병렬프로그램의 대표적인 예는 OpenMP와 PCF(Parallel Computing Forum)를 들 수 있다. 본 논문에서는 현재 공유메모리 기반에서의 병렬 프로그램에 대한 산업표준으로서 널리 사용되고 있는 OpenMP에 대해서 설명한다. 메시지전달을 기반으로 하는 병렬프로그램의 대표적인 예는 MPI와 PVM(Parallel Virtual Machine)을 들 수 있다. MPI는 메시지전달 라이브러리에 대한 표준 명세로서, 사용자에게 풍부하고 다양한 라이브러리를 제공하고 있으며, 현재는 MPI의 높은 이식성과 성능으로 사용자들에게 보편화되고 있다. 본 논문에서는 메시지전달 라이브러리의 표준으로 채택되고 있는 MPI에 대해서 설명한다.

2.1 공유메모리 병렬 프로그램

OpenMP[5,13]는 공유 메모리를 사용하는 병렬 프로그램 모델로서, 표준 C/C++와 Fortran 77/90을 확장하는 디렉티브(directives)와 라이브러리들의 집합이라 할 수 있다. 공유 메모리 병렬 프로그래밍 모델인 POSIX 스레드와는 달리 OpenMP는 병렬화 디렉티브로 순차적 프로그램을 병렬화하기가 용이하다. 또한 OpenMP는 큰 단위의 병렬성(coarse-grain parallelism)을 구현하기위해 orphan 디렉티브 개념을 제공한다. orphan 디렉티브는 어휘적으로 병렬 영역(parallel region) 외부에 선언된 디렉티브를 말하는 것으로, 병렬 프로그래밍의 확장성(extension)을 높여 준다.

OpenMP에서 제공하는 디렉티브는 새로운 스레드 그룹을 생성하는 병렬화 디렉티브, 현재 병행하는 스레드들 간에 작업을 나누어 수행하도록 하는 작업공유 디렉티브, 병렬 영역 외부에 정의된 변수의 공유를 제어하는 데이터 환경을 위한

디렉티브, 병렬 수행하는 스레드간의 수행 순서를 제어하는 동기화 디렉티브 등이 있다. 병렬화 디렉티브로는 PARALLEL, PARALLEL DO 등이 있고, 작업 공유 디렉티브로는 Do, Section 등이 있고, 데이터 환경을 위한 디렉티브로는 Thread Private와 데이터의 접근 범위를 지정하는 Private, Shared 등이 있으며, 동기화를 위한 디렉티브로는 Master, Critical, Barrier 등이 있다. OpenMP는 마스터 스레드(master thread)라 불리는 하나의 스레드로 수행을 시작하며, 생성(fork)과 합류(join)에 의한 병렬화 모델을 사용한다. 다중 스레드에 의해 병행 수행되는 병렬 영역은 OpenMP의 기본적인 병렬화 디렉티브인 PARALLEL에 의해 정의되며, 마스터 스레드가 병렬 영역을 만나면 다중 스레드를 가지는 하나의 그룹을 생성하여 병렬 영역을 수행한다. OpenMP는 구현이 쉽고, 작은 단위의 병렬성(fine grain parallelism)을 가지는 프로그램 수행 시에 효율적이지만, 공유메모리 환경을 기반으로 개발되어 분산메모리 시스템에서 사용할 수 없는 단점[2]이 있다.

2.2 메시지전달 병렬 프로그램

MPI[8]는 메시지전달 라이브러리의 표준이며, 가장 일반적인 고수준 메시지전달 시스템이다. MPI는 프로세스가 메시지를 보냄으로써 서로 통신하는 명시적인 메시지전달 패러다임을 가지고 있다. MPI-1은 1994년에 발표되었고, MPI-2는 1997년에 발표되었으며 MPI-1 보다 많은 부분이 개선되었다. 예를 들면, MPI-2는 병렬 입출력과 동적인 프로세스 관리, 프로그래밍 언어들간의 상호운용성 등을 포함하고 있다. 하지만, MPI는 분산메모리 환경을 기반으로 개발되어, 각 노드의 지역메모리 자료를 메시지 송·수신을 통하여 복사해야 하므로 지나치게 작은 단위의 병렬성을 가지는 프로그램 수행 시에 통신 부담이 매우 큰 단점[2-4,10,16]이 있다. 그러므로 MPI는 여러 SMP 클러스터사이에 넓게 분산되어진 작업 즉, 큰 단위의 병렬성(coarse-grain parallelism)에 사용된다.

MPI는 다중 프로세스들을 생성(spawn)하고 난 뒤, TCP/IP를 통해 통신하게 되는데 이들 프로세스들은 같은 주소 공간을 공유하지 않는다[16]. MPI의 일대일 통신은 송·수신 사건의 복귀(return)에 의해 송·수신의 버퍼 재사용 여부를 알리는 신호발생 여부에 따라 통신 모드에 따른 다양한 함수가 제공되고 있다. 송신 모드의 종류로는 표준 모드(standard mode), 버퍼 모드(buffered mode), 동기화 모드(synchronous mode), 준비 모드(ready mode) 등의 네 가지가 있으며, 수신 모드의 종류로는 표준 모드가 있다. 각 송·수신 모드는 해당 메시지의 수행사건이 완료되어 다음의 메시지 수행이 가능하게 되었을 때만 다음 명령을 수행하게 되는 블럭킹(blocking) 타입과 해당 메시지의 수행사건이 발생된 것을 확인했을 때 즉시 다음 명령을 수행하게 되는 넌블럭킹(nonblocking) 타입으로 구분하게 된다.

2.3 혼합된 병렬 프로그램

최근에는 병렬프로그램의 수행 성능을 개선하기 위해서 SMP 클러스터에서 MPI와 OpenMP를 혼합한 형태로 프로그래밍하는 것이 주된 경향이 되고 있다. 이러한 SMP 클러스터 시스템은 분산메모리를 사용하면서 각 노드 내에서는 공유메모리를 사용하게 된다. <그림 1>은 MPI와 OpenMP 프로그램을 혼합한 프로그램 모델을 계층적 구조를 보여주고 있으며, SMP 클러스터 시스템에서의 프로그램 수행과정을 나타내고 있다. MPI의 병렬성은 n개의 프로세스를 가지는 상위 수준에서 발생하고, 그 하위 수준에서 m개의 스레드들

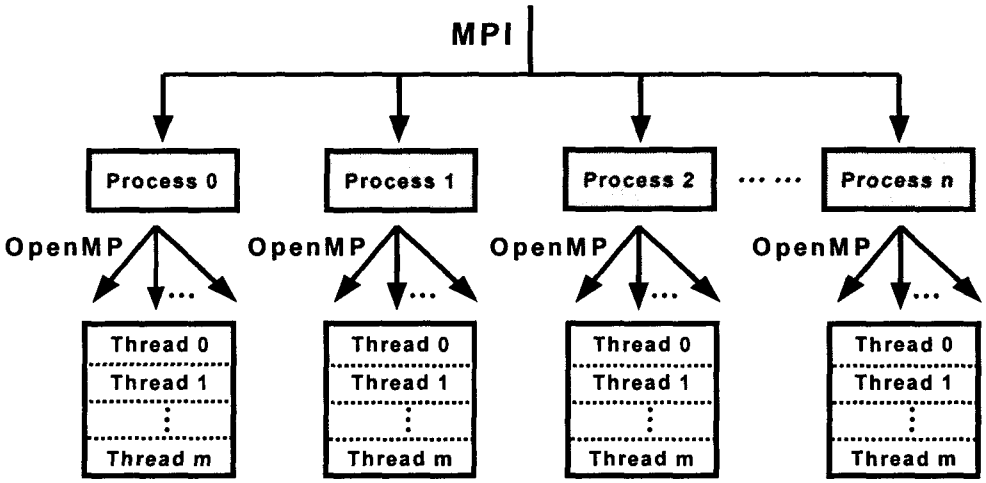

그림 1. 혼합된 OpenMP/MPI의 계층적 구조

이 수행하는 OpenMP의 병렬성이 발생하고 있음을 알 수 있다. 하지만 실제 OpenMP의 병렬성 내에는 메시지전달이 없는 경우를 원칙으로 한다. 즉, 모든 MPI 호출은 OpenMP의 병렬구간을 제외한 곳이나 마스터 스레드 (OMP MASTER)에 의해행되는 곳에서만 가능하다. 게다가 OpenMP의 병렬구간이 아니더라도 OMP SINGLE에서는 MPI 호출을 하지 않는 것이 좋다. 예를 들며, OpenMP/MPI의 혼합형 프로그램 모델은 OpenMP와 MPI로 각각 프로그래밍하여 수행한 것보다 효율적이며, OpenMP와 MPI의 장점들을 결합한 것이라고 볼 수 있다. 특히, MPI 프로그램이 부하불균등을 보이거나, 작은 단위의 프로그램 수행으로 인해 확장성이 저하되거나, 메모리의 한계로 인해 자료 코드에 대해 복사를 해야 하거나, 혹은 MPI 단위 프로세스 수의 제약이 있는 경우에 효율적인 장점[2-4,10,16]이 있다.

<그림 2>는 MPI 프로그램에 OpenMP 디렉티브를 삽입하여 작성한 혼합형 예제 프로그램이다. 그림에서 MPI 프로그램은 MPI_Init()에 의해 시작되고 MPI_Finalize()에 의해 종료하게 된다. MPI 프로그램에 삽입된 OpenMP 디렉티브인

```
#define Comm MPI_Comm_World
int i1, i2, i3, i4, myid;
MPI_Status st;
MPI_Request rq;
MPI_Init();
MPI_Comm_size(Comm, &numprocs)
MPI_Comm_Rank(Comm, &myid);
#pragma omp parallel for
for ( i1=1; i1<=2; i1++ )
 if ( i1==1 ) {
   MPI_Recv(MPI_Char, myid, 2, Comm, &st);
   MPI_Isend(MPI_Char, myid, 1, Comm, &rq);
   for(flag=0; !flag;) MPI_Test(&rq, &flag, &st); }
 else if ( i1==2 ) {
   #pragma omp parallel for
   for ( i2=1; i2<=3; i2++ ) {
     MPI_Isend(MPI_Char, 0, 2, Comm, &rq);
     for (flag=0; !flag;) MPI_Test(&rq, &flag, &st); }
   #pragma omp parallel for
   for ( i3=1; i3<=2; i3++ )
     if ( i3==1 ) {
      MPI_Recv(MPI_Char, myid, 1, Comm, &st);
      MPI_Recv(MPI_Char, myid, 3, Comm, &st); }
     else if ( i3==2 )
      #pragma omp parallel for
      for ( i4=1; i4<=2; i4++ )
       if ( i3==1 ){
        MPI_Isend(MPI_Char, myid, 3, Comm, &rq);
        for (flag=0; !flag;)MPI_Test(&rq, &flag, &st);}}
MPI_Finalize();
```

그림 2. 혼합된 OpenMP/MPI의 예제 프로그램

#pragma omp parallel for 영역은 스레드들을 병렬적으로 생성(fork)하고, 합류(join)하도록 하는 컴파일러 디렉티브이다. 각 스레드들 간의 메시지 전달은 해당 메시지의 수행사건이 완료되어 다음의 메시지 수행이 가능하게 되었을 때만 다음 명령을 수행하게 되는 블럭킹(blocking) 타입과 해당 메시지의 수행사건이 발생된 것을 확인했을 때 곧바로 다음 명령을 수행하게 되는 넌블럭킹(non-blocking) 타입으로 구분된다. <그림 2>에서의 예제 프로그램과 같이 하나의 프로세스에 의해 수행되는 경우에, 송신 명령어로 블럭킹 타입인 MPI_Send()를 사용하면, 대응되는 수신 명령어가 수행 완료되기 전의 경우이므로 해당 송신 메시지가 안전하게 수신완료 되었는지를 확인할 수 없기 때문에 교착상태(deadlock)가 발생된다. 그러므로 송신 메시지가 송신버퍼에 버퍼링된 것만을 확인하고 곧바로 다음을 수행하게 하는 넌블럭킹 타입의 MPI_Isend() 함수를 사용해야 하며, 그 후에 버퍼링된 송신 메시지가 송신되었음을 확인하기 위해 MPI_Test() 함수를 사용한다. 수신 명령어로 블럭킹 타입인 MPI_Recv() 함수를 사용하면 해당 수신 메시지가 수신되지 않는 경우에 블럭킹되며, 문맥교환(context switching)을 통해서 대응되는 송신명령을 포함하는 수행을 계속하여 진행할 수 있다. 넌블럭킹 타입인 MPI_Irecv() 함수를 사용하면 MPI_Isend() 함수와 마찬가지로 MPI_Test() 함수를 사용하여 해당 수신 메시지가 수신버퍼에 수신된 것을 확인할 수 있으므로 순차 수행의 경우에는 MPI_Recv() 함수와 동일한 효과를 가진다. 본 예제에서는 MPI_Recv() 함수를 사용한다.

3. 경합조건의 유형

공유메모리와 메시지전달을 혼합하여 사용하는 병렬 프로그램은 논리적인 오류 외에도 경합조건 등의 병렬오류로 인한 비결정적인 수행을 초래하므로 기존의 순차프로그램보다 디버깅이 어렵다. 이러한 비결정적인 수행 결과의 주된 원인은 병렬로 수행되는 스레드들이 적절한 동기화 없이 공유메모리에 적어도 하나 이상의 쓰기 사건으로 동시에 접근하거나, 혹은 스레드들 간의 메시지전달에 결정적인 순서화를 보장하지 않는 경우에 나타난다. 이러한 병렬프로그램 오류를 공유메모리 프로그램의 경우에 자료경합이라 하고, 메시지전달 프로그램의 경우에 메시지 경합이라 한다.

3.1 자료경합

자료경합은 병렬로 수행하는 스레드들이 적절한 동기화없이 하나의 공유 변수에 대해 하나 이상의 쓰기 사건으로 접근할 때 발생한다. 병렬로 수행되는 두 스레드들에서 발생되는 공유 변수에 대한 접근은, 두 접근사건이 모두 쓰기 접근사건이거나, 적어도 하나는 쓰기 접근사건인 경우이다. 병렬 프로그램에서 순차적 명령열(instruction sequence)들로 구성된 스레드는 발생하는 명령들에 의해 각각 읽기와 쓰기 접근사건의 집합으로 구성된다. <그림 3>을 보면, 병렬로 수행되는 스레드들 간에 적절한 동기화 없이 스레드 1과 스레드 2가 하나의 읽기 사건과 하나 이상의 쓰기 사건으로 동시에 공유변수 X에 접근하고 있다. 이때, 스레드 1이 먼저 수행하게 되면 Z의 결과 값은 4가 되고, 스레드 2가 먼저 수행하게 되면 Z의 결과 값은 3이 되어 비결정적인 결과 값을 가지게 된다. 자료경합에 의한 프로그램의 비결정적인 수행은 프로그래머가 의도하지 않은 결과를 초래하므로 병렬프로그램의 디버깅을 위해 반드시 탐지되어야 한다.

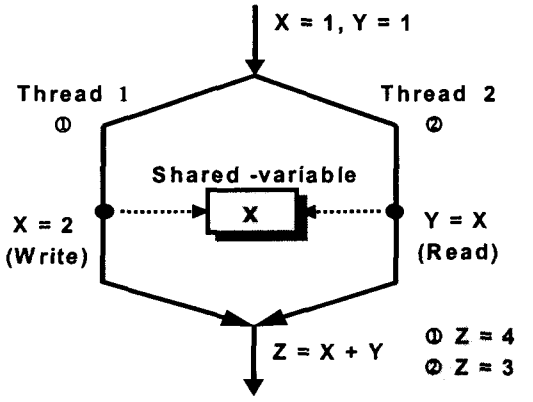

그림 3. 자료경합의 예

3.2 메시지경합

메시지경합은 병렬로 수행하는 스레드들이 결정적인 순서화 없이 하나의 스레드에 대해 메시지 전달을 하는 경우에 발생한다. <그림 4>를 보면, 메시지경합은 4개의 프로세서가 병렬로 수행되어, 프로세스 0과 프로세스 2가 프로세스 1에 대해 결정적인 순서화를 보장하지 않고 동시에 메시지 전달을 하는 경우에 발생한다. 마찬가지로, 프로세스 1과 프로세스 3이 프로세스 2에 대해 결정적인 순서화를 보장하지 않고 동시에 메시지전달을 하므로 메시지경합이 발생되고 있다. 여기서, 프로세스 2에서 발생된 메시지경합은 x = 1일 경우에만 발생되는 메시지경합이다. 이러한 메시지경

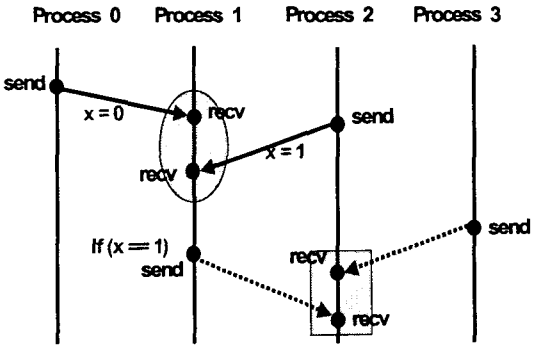

그림 4. 메시지경합의 예

합을 영향 받은 경합(affected race)이라고 하며, 프로세스 1에서 발생된 메시지경합을 탐지하여 디버깅하면 발생하지 않을 수도 있다. 일반적으로, 수행 시에 처음으로 발생되는 경합이 중요하다. <그림 4>와 같이 처음 발생한 경합의 영향에 의해 그 후에 나타나는 경합들이 인위적 경합(artifact race)이거나, 탐지되지 못한 불투명한 경합(hidden race)일 수 있다. 수행중 탐지 기법에서 보고되는 첫 경합은 실제로 발생된 첫 경합이 아닐 수 있으나, 동일한 입력으로 연속적인 재수행을 통해서 처음 발생되는 경합을 탐지해 낼 수 있다. 메시지경합 역시 프로그래머가 의도하지 않은 비결정적인 결과를 초래하므로 디버깅을 위해 반드시 탐지되어야 한다.

4. 병렬 프로그램의 수행중 경합 탐지

수행중 경합탐지 기법은 경합탐지 프로토콜과 프로토콜을 적용 시에 병행한 스레드 간의 논리적인 병행성 여부를 판단하기 위한 병행성 정보 생성 기법으로 구성된다. 수행중 경합탐지 프로토콜은 공유메모리를 위한 프로토콜[6]과 메시지전달을 위한 프로토콜[12]로 구분된다. 공유메모리를 위한 프로토콜의 경우에는 이전 접근사건과 현재 접근사건 사이의 논리적인 병행성을 결정하고 다음에 발생될 자료경합을 탐지하기 위해 접근역사를 유지한다. 그리고 메시지전달을 위한 프로토콜의 경우에는 이전 수신사건과 현재 수신사건에 대한 송신사건과의 병행성 여부를 결정하고, 다음에 발생될 메시지경합을 탐지하기 위해 메시지역사를 유지하게 된다. 경합탐지 프로토콜을 수행하기 위해 각 스레드들은 병행성 정보를 생성해야만 병행한 스레드들 간의 논리적 병행성을 검사하여 경합을 탐지하고 보고할 수 있다. 이러한 병행성 정보는 수행 중에 동적으로 생성 가능해야 하고

어떠한 프로토콜을 적용하더라도 경합탐지가 가능해야 한다.

4.1 경합탐지 기법의 수행

수행중 경합탐지 기법은 먼저, 프로그램 수행 동안 정보를 수집하기 위하여 프로그램을 변형(instrument)한 뒤, 프로그램 수행과 함께 분석이 진행되는 기법이다. 수행중 경합탐지 기법은 프로그램 전체의 수행정보를 저장하지 않고 수행이 진행됨에 따라 불필요한 정보는 제거하므로 기억공간의 부담이 없다. 하지만, 프로그램에 대한 모든 분석이 수행 중에 진행되므로 수행 중에 유지해야하는 정보를 위한 기억장소와 수행시간의 부담을 가지고 있다. 수행중 경합탐지 기법의 특징은 많은 경합을 탐지하지는 못하지만, 탐지 대상인 프로그램에 경합이 존재한다면 각 공유변수에 관련된 적어도 한 개의 경합은 반드시 탐지한다는 것이다.

수행중 경합탐지 기법의 효율성은 병행성 정보 생성 효율성과 생성된 병행성 정보의 적용 효율성으로 크게 나눌 수 있으며, 각각 공간 및 시간복잡도로 나타낸다. 병행성 정보 생성 효율성의 공간복잡도는 병행성 정보인 레이블을 저장하기 위한 공간이며, 시간복잡도는 레이블을 생성하기 위한 시간이다. 생성된 병행성 정보의 적용 효율성의 공간복잡도는 경합을 탐지하기 위해 유지하게 되는 접근역사와 메시지역사에 대한 공간이며, 시간복잡도는 이전 수행사건과 현재 수행사건 사이의 논리적인 병행성을 결정하기 위한 시간이다. 여기서, 병행성 정보 생성 효율성의 공간복잡도는 감시하는 프로그램의 최대 병렬성과 임의의 병행성 정보의 스레드정보와 부가정보를 곱한 것이고, 시간복잡도는 병행성 정보의 크기와 병행성 정보의 생성 시에 요구되는 공유 자료구조의 경쟁시간을 곱한 것이다. 병행성 정보 적용 효율성의 공간복잡도는 감시하는 공유변수의 수와 접근역사 혹은 메시지역사를 유지하는 수행사건 항목의 수 그리고, 접근역사 혹은 메시지역사 내의 각 항목의 크기를 곱한 것이고, 시간복잡도는 공유 자료구조인 접근역사에 대한 경쟁시간과 접근역사 혹은 메시지역사 내에 유지하는 수행사건 항목의 수, 그리고 한 항목의 크기를 곱한 것이다.

그러므로 수행중 경합탐지 기법의 효율성은 병행성 정보 생성에 소요되는 시간과 공간복잡도와 접근역사 혹은 메시지역사를 유지하기 위한 시간 및 공간복잡도에 크게 영향을 받는다. 하지만 기존의 병행성 정보 생성 기법들[1,6]은 병행성 정보 적용 시에 효율적인 시간복잡도를 가지지만 병행성 정보 생성 시에 공유 자료구조를 사용하여 심각한 병목현상을 발생시키는 경우[6]이거나, 개별 자료구조를 사용하여 병행성 정보 생성 시에 병목현상은 발생하지 않지만 병행성 정보 적용 시에 내포병렬성에 의존하는 비효율적인 시간복잡도를 가지는 경우[1]이다. 효율성에 영향을 주는 인수들은 프로그램의 최대 병렬성(T), 내포 깊이(N)와 감시하는 공유 변수들의 수(V) 등이다.

4.2 경합탐지를 위한 프로토콜

수행 중에 경합을 탐지하기 위한 프로토콜의 유형에는 경합 검증(race verification) 방식과 최초경합 탐지(first-race detection) 방식이 있다. 경합 검증 기법에서는 만약 대상으로 하는 병렬 프로그램에서 경합이 존재한다면 적어도 하나 이상의 경합은 보고된다는 것을 보장하지만, 처음으로 탐지된 경합이 그 병렬 프로그램에서 처음으로 발생한 경합이라는 것을 보장하지 못한다. 특히, Mellor-Crummey의 탐지 프로토콜에서는 접근역사내의 가장 최근의 쓰기 접근과 읽기 접근 중에서, 가장 오른쪽에서 발생한 읽기 접근과 가장

왼쪽에서 발생한 읽기 접근만을 선택하여 유지함으로써, 수행중 적어도 하나의 경합을 탐지할 수 있는 기법이다. 이 방식은 기억 공간의 절약을 위해서 프로그램 모니터링 중에 수집한 정보를 삭제할 수 있는데, 이러한 삭제는 접근역사에 각 스레드에서 경합에 참여하는 두 개의 읽기와 하나의 쓰기 접근만을 저장한다. 이 때문에 각 쓰레드의 처음으로 발생하는 접근에 의한 경합을 보고하 지 못하는 경우가 발생할 수 있다는 단점이 존재한다.

최초경합 탐지 방식에서는 수행 중에 최초로 발생한 경합을 탐지하는 기법이다. 기존의 기법들에 적용되는 프로그램은 제한된 모델에만 적용된다는 단점이 있다. 확장성을 가지고 있는 최초경합 탐지 기법은, 프로그램 수행 중에 각 접근들을 검사하여 접근역사 내에 있는 다른 접근들과 비교하여 최초경합을 탐지하는 방식이다. 이는 최대 2개의 접근으로 구성된 접근역사가 각 스레드들에 분산되어 있고, 이를 이용하여 수행되기 때문에 기존의 공유 변수들의 접근에서 야기되는 병목현상을 감소시키는 장점을 가진다. 따라서 쓰레드들의 수가 증가하더라도 서로 분산되어 있는 접근들을 비교하기 때문에 시간 복잡도 면에서는 유리한 기법이다. 하지만 스레드들 간의 동기가 없는 비내포 병렬 프로그램만 대상으로 하기 때문에 그 적용 범위가 한정된다고 할 수 있다. 이 기법과는 달리 동기가 있는 내포 병렬성을 갖는 프로그램에서 최초경합을 탐지할 수 있는 기법은 최초경합에 관련된 키(key) 접근 개념을 사용하여 좀 더 일반성이 있는 방법을 제공한다. 수행 중에 경합을 탐지하기 위한 프로토콜을 구분하는 또 다른 유형으로는 기본적으로 순차 수행만을 위한 기법과 병렬 수행으로 탐지하는 기법이 있는데, 경합탐지 능력을 강화하기 위해서 두 번의 병렬 수행을 필요로 하는 기법과 순환적 디버깅을 위한 기법도 있다.

4.3 경합탐지를 위한 병행성 정보 생성

각 스레드들은 병행성 정보에 해당하는 유일한 값인 레이블이 있어야만 수행사건 발생 시에 스레드들 간의 논리적 병행성을 검사하여 경합을 탐지하고 보고할 수 있다. 수행중 경합탐지 기법에서의 병행성 정보 생성의 조건은 수행 중에 동적으로 생성 가능해야 하며, 임의의 프로토콜에 대해 적용하더라도 경합을 탐지할 수 있어야 한다. 병행성 정보를 생성하는 기법으로는 Offset-Span (OS), Nest-Region(NR), English-Hebrew(EH), Task-Recycling(TR), Breadth-Depth(BD) 레이블링[1] 등이 있다. 병행성 정보를 생성하는 기법에 있어서 병렬성의 정도에 따른 확장성 여부를 살펴보면, 레이블을 생성할 때 공유 자료구조 혹은 개별 자료구조를 사용하느냐에 따라 나눌 수 있다. 병행성 정보를 생성하기 위해 공유 자료구조를 사용하는 TR 레이블링 기법의 경우는 모든 스레드에 대한 병행성 정보를 생성하는데 있어서 심각한 병목현상이 발생한다. 이러한 병목현상은 프로그램의 병렬성이 증가할수록 상대적으로 증가하게 된다. 예를 들어, 병렬로 병행성 정보를 생성하는 시점에서 공유 자료구조에 순차적으로 접근하여 병행성 정보 생성에 필요한 정보를 참조하게 되면 심각한 성능 저하를 야기시키게 된다. 그러므로 프로그램의 병렬성이 증가하더라도 병행성 정보 생성 시에 지역자료 구조를 사용함으로써 병목현상이 발생하지 않는 확장성을 제공하는 것이 필요하다. 확장성을 제공하는 레이블링 기법들로는 OS, EH, NR, BD 레이블링 기법 등이 있다.

하지만 확장성이 제공되는 레이블링 기법들 중에서 OS 레이블링 기법과 EH 레이블링 기법은 병행성 정보의 생성에 따른 시간 및 공간 복잡도

에 대한 효율성이 낮다. 왜냐하면 접근역사에 저장되는 병행성 정보의 크기가 내포 깊이만큼 증가하여 공간 부담이 크며, 경합 탐지 프로토콜의 적용 시에 접근역사에 저장되는 병행성 정보와의 병행성 비교는 내포 깊이만큼의 시간 부담이 크기 때문이다. 하지만 기존의 레이블링 기법들 중에는 확장성을 제공하면서도 시간과 공간에 대한 효율성이 제공되는 경우도 있다. 시간과 공간에 대한 효율성이 제공되는 기법들로는 NR 레이블링 기법과 BD 레이블링 기법이다. NR 레이블링 기법과 BD 레이블링 기법은 병행성 정보 생성시 지역 자료 구조를 사용함으로써 병목현상이 발생하지 않으며, 접근역사에 저장하는 병행성 정보의 크기가 상수 값을 갖는 공간적 효율성을 제공한다. 또한 경합 탐지 프로토콜의 적용 시에 BD 레이블링 기법의 경우 부가정보가 갖는 내포 깊이만큼 시간적 부담을 갖게되고, NR 레이블링 기법은 부가정보가 정렬된 리스트이므로 이진탐색으로 병행성 정보를 비교하게 되는 시간적 효율성이 제공된다. 하지만 NR 레이블링 기법의 경우 적용 대상 프로그램 모델에 있어서 동기화가 있는 공유메모리 프로그램에서 적용이 불가능하다.

그러므로 병행성 정보 생성 시에 개별 자료구조를 사용함으로써 병목현상을 제거하여 병행성 정보를 확장적으로 생성하며, 생성된 병행성 정보의 적용 시에 시간적 효율성을 제공하는 레이블링 기법이 필요하다.

5. 효율적인 새로운 기법 소개

공유메모리 프로그램에서의 병행성 정보 생성 기법으로는 OS, EH, TR, NR, BD 레이블링 기법 등이 사용 가능하다. 즉, 벡터 타임스탬프 기법을 제외한 대부분의 레이블링 기법들은 공유메모리 모델인 OpenMP 프로그램 모델에서만 적용 가능하다. 메시지전달 프로그램에서의 병행성 정보 생성 기법으로는 기본적으로 벡터 타임스탬프 기법이 사용 가능하다. 하지만 벡터 타임스탬프 기법도 수행 중에 새로운 스레드를 생성하는 MPI-2에서는 사용할 수 없다. 공유메모리 프로그램에 사용 가능한 레이블링 기법 중에서도 동기화가 있는 경우에 사용 가능한 기법들은 메시지전달 프로그램에서도 사용 가능한 특징이 있다.

최근에는 병렬프로그램의 수행 성능을 개선하기 위해서 SMP 클러스터에서 공유메모리와 메시지전달이 혼합된 형태로 프로그래밍하는 것이 주된 경향이 되고 있다. 하지만 기존의 병행성 정보 생성 기법들은 최근의 프로그래밍 경향을 고려하지 않았기 때문에 공유메모리와 분산메모리의 각 프로그램 모델 특성에 따라 서로 다른 기법을 사용해야 한다. 그러므로 공유메모리 기반의 프로그램 모델과 분산메모리 기반의 프로그램 모델 모두 적용 가능한 새로운 병행성 정보 생성 기법의 개발이 필수적이다. 특히, 공유메모리 프로그램의 산업표준인 OpenMP와 메시지전달 프로그램의 산업표준인 MPI가 혼합된 프로그램 모델이 대부분이므로, OpenMP와 같은 공유메모리 프로그램 혹은, MPI와 같은 메시지전달 프로그램에서도 적용이 가능할 뿐만 아니라 OpenMP와 MPI가 혼합된 병렬 프로그램에서도 적용 가능한 새로운 병행성 정보 생성 기법을 소개한다.

또한 기존의 병행성 정보 생성 기법은 병행성 정보 생성 시에 개별 자료구조를 사용하여 병목현상은 발생하지 않지만 병행성 정보 적용 시에 내포 병렬성에 의존하는 비효율적인 시간복잡도를 가지거나, 병행성 정보 적용 시에 효율적인 시간복잡도를 가지지만 병행성 정보 생성 시에 공유자료구조를 사용하여 심각한 병목현상이 발생하게 된다. 그러므로 병행성 정보 생성 시에 병목현

상을 제거하여 확장성을 제공하면서도 병행성 정보 적용 시에 효율적인 시간복잡도를 제공하는 새로운 병행성 정보 생성 기법을 소개한다.

5.1 새로운 병행성 정보 생성 기법

DV(Distributed Vectors) 레이블링 기법의 병행성 정보는 스레드 정보와 부가정보로 구성되며, 두 정보를 구분하는 구분자(delimiter)는 §로 나타낸다. 스레드 정보에는 접근역사 혹은 메시지역사에 저장되는 스레드 레이블인 $<t, v>$와 t를 구하기 위해 사용되는 s가 있다. 여기서, t는 각 스레드의 식별자 의미하며, 새로 생성되는 각 스레드마다 고유한 정수값을 가진다. v는 각 스레드 식별자의 재사용에 대한 횟수인 스레드 버전(version)을 의미하며, 스레 드의 생성과 합류, 그리고 메시지전달 시에 스레드 식별자가 재사용될 때마다 1씩 증가하게 된다. s는 동시에 발생 가능한 최대 병행 스레드 수인 스레드 공간(space)을 의미하며, 스레드 인덱스의 상한값(upper bound)으로 계산한다. 부가정보에는 이전에 수행한 스레드와의 병행성 여부를 검사하기 위한 버전벡터(version vector)인 M이 있다. M은 현재 시점까지 스레드가 알고 있는 모든 스레드들의 버전 값을 나타내며, 현재 시점까지 스레드가 알지 못하는 스레드들의 버전 값은 0이 된다.

DV 레이블링 기법은 병행성 정보를 생성하는 알고리즘과 병행성 정보의 병행성 여부를 검사하는 알고리즘으로 구성된다. 병행성 정보를 생성하는 알고리즘에는 초기화 알고리즘, 스레드 생성 알고리즘, 스레드 합류 알고리즘, 메시지전달 시의 송신·수신 알고리즘 등이 있다.

5.2 새로운 병행성 정보 생성 기법의 효율성

DV 레이블링 기법에 대해 병행성 정보의 생성 효율성과 병행성 정보의 적용효율성을 최악의 경우(worst-case)의 공간 및 시간복잡도로 구분하여 나타낼 수있다. 병행성 정보 생성효율성에서의 병행성 정보를 저장하기 위한 공간복잡도와 스레드들의 병행성 정보 생성 시에 소요되는 시간복잡도로 나눈다. 병행성 정보 적용효율성에서도 병행성 여부를 결정하기 위해 병행성 정보와 비교하게 되는 스레드 레이블의 공간복잡도와 병행성 여부를 결정하는데 소요되는 시간복잡도로 나눈다. 이러한 효율성에 영향을 주는 인수들은 프로그램의 최대 병렬성(T)과 내포깊이(N) 등이다.

먼저, 병행성 정보 생성효율성 측면에서 공간복잡도의 경우를 보면, DV 레이블링 기법은 병행성 정보 ($<t, v>$, s)§M이 최악의 경우에 $O(T)$개만큼 생성되며, 각 스레드들이 다른 스레드들과의 송·수신한 메시지전달을 모두 고려한 M의 저장공간을 $O(T)$만큼 필요로 하므로, 전체적으로 최악의 경우에 $O(T^2)$의 공간복잡도를 가진다. DV 레이블링 기법의 시간복잡도는 하나의 스레드가 병행성 정보를 생성할 때, 다른 스레드들과의 송·수신한 메시지전달을 모두 고려한 M을 생성하는데 소요되는 시간이 최악의 경우에 $O(T)$의 복잡도를 가진다. 병행성 정보의 적용효율성 측면에서 공간복잡도의 경우를 보면, DV 레이블링 기법은 임의의 두 스레드에 대한 병행성 정보 비교 시에, 현재 스레드는 생성된 병행성 정보 중에서 부가정보인 M이 필요하며, 이전 스레드는 병행성 정보 중에서 상수 크기의 스레드 레이블 $<t, v>$만을 필요로 하므로 병행성 여부를 결정하기 위해 유지되는 이전 스레드에 대한 공간복잡도는 $O(1)$이 된다. DV 레이블링 기법의 시간복잡도의 경우는 임의의 두 스레드를 L_x, L_y라 할 때, L_x의 스레드 버전값 v_x와 L_y의 버전벡터 M_y 중에서 L_x의 스레드 식별자에 해당하는 버전값 $M_y[t_x]$를 비

교하여 병행성 여부를 검사할 수 있기 때문에 $O(1)$이 된다.

6. 결 론

병렬프로그램의 수행 성능을 개선하기 위해서 SMP 클러스터에서 MPI와 OpenMP를 혼합한 형태로 프로그래밍하는 것이 일반적이다. 이러한 SMP 클러스터 시스템은 분산메모리를 사용하면서 각 노드 내에서는 공유메모리를 사용하게 된다. OpenMP/MPI의 혼합형 프로그램 모델은 MPI와 OpenMP로 각각 프로그래밍하여 수행한 것보다 효율적이다.

하지만 공유메모리와 메시지전달을 혼합하여 사용하는 병렬 프로그램은 논리적인 오류 외에도 경합조건 등의 병렬오류 즉, 자료경합과 메시지경합이 발생하게 된다. 이러한 병렬 오류들을 디버깅하기 위한 효율적인 기법들이 요구되고 있으나, 현재로는 OpenMP/MPI의 혼합형 프로그램 모델에서의 수행속도 향상에 따른 연구가 주류를 이루고 있다. 본 논문에서는 OpenMP/MPI의 혼합형 프로그램 모델에서의 디버깅을 위한 새로운 기법을 소개하였다. 소개된 DV 레이블링 기법은 크게 2가지 측면에서 그 의미를 지닌다. 첫째는 기존의 기법들과 비교해 볼 때, 병행성 정보 생성 시에 병목현상을 제거하여 확장성을 제공하면서도 병행성 정보 적용 시에 효율적인 시간복잡도를 제공하고 있으며, 둘째는 공유메모리와 메시지전달 프로그램 각각에 대해서 적용가능 할 뿐만 아니라 이를 혼합하여 사용하는 병렬 프로그램에서도 병행성 정보를 동적으로 생성함으로써 효율적인 수행중 경합 탐지가 가능하다. 그러나 앞으로 병행성 정보 생성에 따른 시간 부담을 줄이는 연구가 필요하다.

참 고 문 헌

[1] Audenaert, K., "Maintaining Concurrency Information for On-the-fly Data Race Detection," *Parallel Computing*, pp. 1-8, North-Holland, Sept. 1997.

[2] Mark, B. and L. Smith, "Programming Next Generation HPC Systems: the Mixed-mode Model," *Next Generation HPC Systems and the Grid*, The Univ. of Edinburgh Conf. and Training Centre, Edinburgh, U.K., Sept. 2001.

[3] Cappello, F. and D. Etiemble, "MPI versus MPI +OpenMP on the IBM SP for the NAS Benchmarks," *Supercomputing*, Dallas, TX, 2000.

[4] Caubet, J., J. Gimenez, J. Labarta, L. DeRose and J. Vetter, "A Dynamic Tracing Mechanism for Performance Analysis of OpenMP Applications," *Int'l Workshop on OpenMP Applications and Tools,* LNCS, 2104: 53-67, West Lafayette, Indiana, July 2001.

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

[6] Dinning, A. and 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.

[7] Fidge, C. J., "Partial Orders for Parallel Debugging," *1st Workshop on Parallel and Distributed Debugging*, pp. 183-194, ACM, May 1988.

[8] Gropp, W., E. Lusk and A. Skjellum, Using MPI: Portable Parallel Programming with the Message-Passing Interface, *The MIT Press*, 2nd Edition, 1999.

[9] Gropp, W. and E. Lusk, User's Guide for mpich, a Potable Implementation of MPI Version 1.2.2,

Univ. of Chicago Press, 1996.

[10] Hoeflinger, J., B. Kuhn, W. Nagel, P. Petersen, H. Rajic, S. Shah, J. Vetter, M. Voss and R. Woo, "An Integrated Performance Visualizer for MPI/OpenMP Programs," *Workshop on OpenMP Applications and Tools*, LNCS, 2104: 40-52, West Lafayette, Indiana, July 2001.

[11] Neyman, M., M. Bukowski and P. Kuzora, "Efficient Replay of PVM Programs," *6-th European PVM/MPI User's Group Meeting*, pp. 83-90, Barcelona, Spain, Sept. 1999.

[12] Netzer, R. H. B. and Miller, B. P., "Optimal Tracing and Replay for Debugging Message-Passing Parallel Program," *The J. of Supercomputing*, 8(4): 371-388, Oct. 1994.

[13] OpenMP Architecture Review Board, OpenMP Fortran Application Program Interface, Ver. 2.0, Nov. 2000.

[14] Park, S, M. Park and Y. Jun, "A Comparison of Scalable Labeling Schemes for Detecting Races in OpenMP Programs," *Int'l Workshop on OpenMP Applications and Tools*, LNCS, 2104: 68-80, West Lafayette, Indiana, July 2001.

[15] Sato, M., S. Satoh, M. Kusano and Y. Tanaka, "Design of OpenMP Compiler for an SMP Cluster," *1st European Workshop on OpenMP*, Lund, Sweden, Sept. 1999.

[16] Smith, L. and J. M. Bull, "Development of Mixed Mode MPI/OpenMP Applications," *Int'l Workshop on OpenMP Applications and Tools*, San Diego, July 2000.

박 소 희

- 1989년 2월 경상대학교 전산통계학과 학사 졸업
- 1996년 2월 경남대학교 컴퓨터공학과 석사 졸업
- 2002년 8월 경상대학교 컴퓨터과학과 박사 졸업
- 2003년 9월~현재 경성대학교 교양과정부 전임강사
- 관심분야 : 분산 및 병렬처리, 운영체제, 멀티미디어
- e-mail: heeya@star.ks.ac.kr