메시지전달 프로그램의 디버깅을 위한 경합조건의 확장적 시각화*

배수연† 박미영†† 전용기†††
경상대학교 컴퓨터과학과
{javaio0, park, jun}@race.gsnu.ac.kr

Scalable Race Visualization for Debugging Message-Passing Programs

Su-Yun Bae0, Mi-Young Park, Yong-Kee Jun
Dept. of Computer Science, Gyeongsang National University

요 약

메시지전달 프로그램에서 가장 먼저 발생하는 경합인 최초경합은 다른 경합에 영향을 주므로 반드시 탐지되어야 한다. 기존의 최초경합 탐지기법은 첫 번째 수행에서 각 프로세스에서 처음으로 발생하는 경합의 위치를 탐지하고, 두 번째 수행에서는 그 위치에서 해당 프로세스를 정지하여 경합하는 메시지를 보고한다. 그러나 이 기법은 프로세스를 중단하여 다른 경합에 영향을 주는 메시지의 전송을 단절시키므로 탐지된 경합들간의 영향관계를 알 수 없게 한다. 본 논문에서는 기존 기법의 두 번째 수행이 종료될 때까지 각 프로세스에서 처음으로 발생하는 경합들간의 영향관계를 추적화일에 기록하며, 수행이 종료된 후에 이 정보를 이용하여 경합들간의 영향관계와 상호 영향을 미치는 경합들을 추상적으로 시각화하는 기법을 제안한다. 이 기법은 경합들의 집합을 추상적으로나 구체적으로 시각화함으로써 경합들간의 영향관계를 확장적이고 직관적으로 알 수 있게 한다. 따라서 본 기법은 최초경합을 수정함으로써 영향받은 경합들을 사라지게 할 수 있으므로 메시지전달 프로그램의 효과적인 디버깅을 가능하게 한다.

1. 서론

병렬컴퓨터에서 널리 사용되고 있는 병렬프로그래밍 모델은 메시지전달[10]이다. 메시지전달 프로그램에서는 두 개 이상의 송신자가 동시에 논리적으로 같은 채널을 통하여 수신자에게 메시지를 보낼 때 메시지경합[8, 11]이 발생된다. 경합은 의도하지 않은 비결정적인 수행결과를 초래하므로 반드시 탐지되어야하며, 가장 먼저 발생하는 경합인 최초경합은 다른 경합에 영향을 줄 수 있으므로 탐지하는 것이 디버깅에 효과적이다.

기존의 최초경합 탐지기법[9]은 사후추적 기법[11]과 수행중 기법[1, 2, 5, 8, 9]이 혼합된 형태로서, 경합탐지를 위해 프로그램을 두 번 수행한다. 첫 번째 수행에서는 각 프로세스에서 처음으로 발생하는 경합의 위치를 탐지하고, 두 번째 수행에서는 그 위치에서 해당 프로세스를 정지한 후에 수신된 메시지를 조사하여 각 프로세스에서 처음으로 발생한 경합을 보고한다. 이 기법은 프로세스의 중단으로 인하여 송신사건이 발생하지 않아 다른 경합에 영향을 주는 메시지의 전송을 단절시킨다. 이는 경합들 간에 존재하는 영향관계[9]를 알 수 없게 하므로 각 프로세스에서 탐지된 경합이 프로그램에서 처음으로 발생하는 경합임을 보장하지 못한다.

본 논문에서는 기존 기법과 달리 두 번째 수행이 종료될 때까지 매 송신사건에서 각 프로세스의 경합정보를 송신 메시지에 첨부하고, 수신사건에서는 수신된 경합정보를 이용하여 각 프로세스에서 처음으로 발생하는 경합에 영향을 미치는 경합들의 정보를 추적화일에 기록한다. 수행이 종료된 후에 이 정보를 이용하여 경합들간의 영향관계와 상호 영향을 미치는 경합[9]들을 추상적으로 시각화하는 기법을 제안한다. 이 기법은 경합들의 집합을 추상적으로나 구체적으로 시각화함으로써 경합들간의 영향관계를 확장적이고 직관적으로 알 수 있게 한다. 따라서 본 기법은 보고된 최초경합을 수정함으로써 영향받은 경합들을 사라지게 할 수 있으므로 메시지전달 프로그램의 효과적인 디버깅을 가능하게 한다.

본 논문에서 제시하는 기법은 메시지전달 병렬프로그램의 표준 라이브러리인 MPI[10] 병렬프로그램을 대상 프로그램으로 하며, 이 기법을 위한 개발 언어로는 자바를 사용한다. 본 기법이 수행되는 플랫폼은 네 개의 노드로 구성된 Alpha 클러스터 시스템으로서, MPI의 표준을 그대로 구현한 MPICH[4]가 설치된 환경이다. 이어지는 2절에서 메시지경합과 경합탐지를 위한 기존의 기법들을 소개한다. 3절에서는 추적화일 생성 및 분석을 통하여 경합조건의 확장적 시각화를 보인다. 그리고 마지막으로 4절에서 결론을 내린다.

2. 연구배경

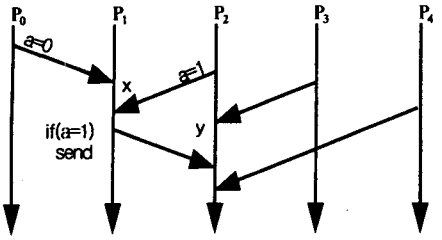
본 절에서는 메시지전달 프로그램에서 발생하는 메시지경합들 중에서 가장 먼저 발생한 최초경합에 대하여 설명하고, 이를 탐지하기위한 기존의 기법들에 대하여 살펴본다.

2.1 최초경합

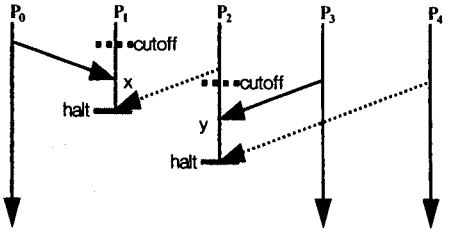
메시지전달[10] 프로그램의 의도하지 않은 비결정적 수행을 야기하는 주된 오류 중의 하나가 메시지경합[8, 11]이다. 경합은 두 개 이상의 송신자가 논리적으로 같은 채널로 메시지를 동시에 송신할 때 발생되며, 프로그램에서 가장 먼저 발생하는 경합을 최초경합이라 한다. 최초경합은 다른 경합에 영향을 주므로 반드시 탐지되어야 한다.

<그림 1>은 최초경합과 영향받은 경합이 함께 존재하는 메시지전달 프로그램의 부분수행이다. 그림에서 P_0, P_1, P_2, P_3, P_4는 병행하게 수행하는 프로세스를 나타내고, 화살표는 각 프로세스의 송수신 사건에 의한 메시지전달을 나타낸다. <그림 1>에서 P_0와 P_2가 P_1에게 메시지를 보내고, P_1, P_3, P_4가 P_2로 메시지를 전송한다. 이때 수신사건인 x, y에 수신되는 각각의 메시지들은 그들의 도착순서가 보장되지 않으므로 서로 경합한다. 또한, P_1에서는 x에서 발생하는 경합의 결과에 의해 (a=1) 조건을 만족하면 P_2로 메시지를 전송한다. 즉, P_2의 y에서 발생하는 경합은 x에서 발생한 경합으로부터 영향받기 때문에 x에서 발생하는 경합을 탐지하

* 본 연구의 일부는 해양수산부 수산특정연구개발 과제의 지원으로 수행되었음
† 학생회원: 경상대학교 석사과정
†† 학생회원: 경상대학교 박사과정
††† 종신회원: 경상대학교 컴퓨터과학과 교수
(교신저자, 컴퓨터·정보통신연구소 연구원)



<그림 1> 최초경합



<그림 2> 최초경합 탐지기법

여 디버깅하면 자연히 사라질 수 있는 경합이다. 그러므로 가장 먼저 발생하는 경합인 수신사건 x에서 발생하는 경합을 탐지하는 것이 디버깅에 효과적이다.

2.2 최초경합 탐지기법

병렬프로그램에서 동적으로 메시지경합을 탐지하는 기법으로는 사후추적 기법[11], 수행중 탐지기법[1, 2, 5, 8, 9] 등이 있다. 사후추적 기법은 프로그램 수행이 종료된 후에 추적화일을 분석하여 경합을 보고하는 기법으로서, 수행시간이 긴 병렬프로그램에서는 대용량의 기억장소를 요구하므로 비현실적이다. 반면에 수행중 탐지기법은 프로그램 수행 중에 경합을 탐지하는 기법으로, 수행 중에 불필요한 정보들은 제거되므로 사후추적 기법보다는 공간복잡도 측면에서 이점이 있다.

기존의 수행중 경합 탐지기법[1, 2, 5, 8, 9]은 프로그램 수행시 매 수신사건마다 수행되며, 이전 수신사건과 송신사건간의 병행성 검사[3]를 통해 경합을 탐지한다. 이 기법으로는 경합존재 탐지기법[8]과 최초경합 탐지기법[1, 2, 5, 9]이 있다. 경합존재 탐지기법은 프로그램에서 경합의 존재 여부만을 판단할 수 있는 기법이다. 반면 최초경합 탐지기법은 각 프로세스에서 경합이 발생하는 최초의 수신사건을 탐지하는 기법이다. 최초경합 탐지기법은 각 프로세스에서 처음으로 발생하는 경합을 탐지함으로써 이 경합에 영향받은 경합들을 사라지게 할 수 있으므로 디버깅에 효과적이다.

기존의 최초경합 탐지기법은 적용 가능한 프로세스의 수에 따라 한 번의 수행으로 한 프로세스에서 발생한 최초경합만을 탐지하는 기법[1, 2]과 모든 프로세스에서 발생하는 최초경합[5, 9]을 탐지하는 기법으로 분류할 수 있다. 또한 후자의 기법은 경합이 발생하는 프로세스의 모든 수신사건을 탐지하는 기법[5]과 오직 경합이 발생하는 첫 번째 수신사건만을 탐지하는 기법[9]이 있다. 경합이 발생하는 첫 번째 수신사건만을 탐지하는 기법은 경합이 발생하는 모든 수신사건을 탐지하는 기법에 비하여 시간적, 공간적 효율성이 높다.

이 기법은 사후추적 기법과 수행중 기법이 혼합된 형태로서 각 프로세스에서 처음으로 발생하는 경합을 탐지하기 위해 프로그램을 두 번 수행한다. 첫 번째 수행에서는 각 프로세스에서 처음으로 경합이 발생하는 위치를 탐지한다. 두 번째 수행에서는 그 위치 이후의 첫 번째 수신사건을 수행하고, 프로세스를 정지시킨 후 메시지 큐에 수신된 메시지들을 조사하여 경합을 보고한다. <그림 2>는 두 번째 수행을 나타낸 것으로 첫 번째 수행에서 탐지된 경합의 위치를 가리키는 cutoff 이후의 첫 번째 수신사건을 수행하고 프로세스를 정지한 것이다. 프로세스의 중단으로 인하여 <그림 1>의 P_1에서 발생했던 송신사건이 발생하지 않으므로 P_2에서 발생하는 경합은 P_1에서 발생한 경합에 영향받았는지 알 수 없으므로 영향받지 않는 경합으로 보고된다. 즉, 이 기법은 프로세스를 중단시켜 경합들 간에 존재하는 영향관계를 알 수 없게 하므로 각 프로세스에서 탐지된 경합이 영향받지 않은 경합임을 보장하지 못한다.

본 논문에서는 프로그램 수행 중에 발생하는 경합들 간의 영향관계와 상호 영향을 미치는 경합들을 추상적이나 구체적으로 시각화함으로써 프로그램에서 처음으로 발생하는 경합을 확장적이고 직관적으로 알 수 있게 한다. 이 기법은 프로그램의 수행 중에 나타나는 모든 송수신 사건이나, 발생하는 모든 경합을 시각화하는 기존의 기법[6, 7, 12]과 달리 각 프로세스에서 처음으로 발생하는 경합을 추상화하여 시각화함으로써 시각적 복잡도를 줄인다.

3. 경합조건의 확장적 시각화

본 연구에서는 프로그램이 종료될 때까지 각 프로세스에서 발생하는 경합들간의 영향관계를 추적하여 확장적으로 시각화함으로써 최초경합을 보고하는 기법을 제안한다. 본 절에서는 확장적 시각화가 수행되는 과정을 설명하고 이를 통해 최초경합을 탐지하는 예를 보인다.

3.1 경합조건의 추상화

경합조건의 확장적 시각화를 위한 수행과정은 첫 번째로 각 프로세스에서 처음으로 발생하는 경합들간의 영향관계 정보를 가진 추적화일을 생성하고, 두 번째는 추적화일을 통해 추적된 경합들간의 영향관계를 분석하여 추상적으로나 구체적으로 시각화한다.

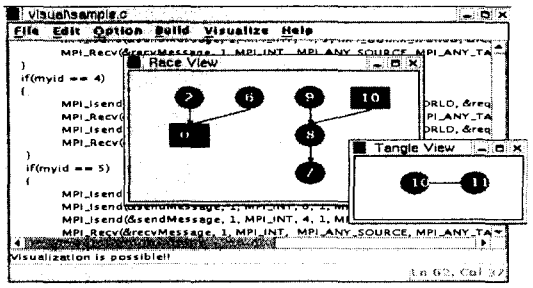
먼저, 추적화일을 생성하기 위해 프로그램을 두 번 수행한다. 첫 번째 수행은 기존의 최초경합 탐지기법과 동일하게 각 프로세스별 최초경합이 발생하는 위치를 파악하고, 두 번째 수행은 기존의 기법과 달리 프로그램이 종료될 때까지 탐지된 위치 이후의 첫 번째 수신사건에서 경합하는 모든 메시지를 수신한다. 이것은 해당 프로세스에서의 경합 발생 여부를 나타내는 경합정보와 발생한 경합에 영향을 미치는 경합이 발생한 프로세스들의 정보를 기록하는 Affecters 버퍼를 사용함으로써 가능하다. 각 프로세스는 매 송신사건마다 경합정보와 Affecters 정보를 메시지에 첨부하고, 이 메시지를 수신한 프로세스에서는 첨부된 경합정보의 값이 TRUE이면 송신 프로세스에서 경합이 발생한 것을 알 수 있다. 이것은 수신 프로세스에서 발생한 경합이 송신 프로세스에서 발생한 경합으로부터 직접 영향을 받은 것이므로 수신 프로세스는 송신 프로세스의 정보를 Affecters 버퍼에 저장하게 된다. 반면, 경합정보의 값이 FALSE이면 간접적으로 영향받은 것이므로 수신 프로세스는 수신된 메시지의 Affecters 정보의 값을 자신의 Affecters 버퍼에 포함시킨다. Affecters 정보는 해당 프로세스의 모든 송수신 사건이 종료된 후에 수신 프로세스의 정보와 함께 추적화일에 저장된다.

다음은 생성된 추적화일을 이용하여 경합들간의 영향관계와 상호 영향을 미치는 경합들을 확장으로 시각화하는 과정으로서, 경합들간의 영향관계에 따른 추상화와 구체화가 요구된다. <그림 3>은 경합조건의 추상적 시각화 알고리즘으로 크게 세 단계로 이루어져 있다. 첫 번째 단계는 추적파일을 읽어 영향관계에 따라 인접리스트를 구성하는 것으로 makeOriginalGraph 함수가 이에 해당한다. makeOriginalGraph 함수는 추적파일에서 수신 프로세스의 정보와 송신 프로세스들의 정보를 담고 있는 Affecters 정보를 이용하여 인접리스트 Graph를 생성한다. Graph는 프로그램 수행에서 발생한 경합의 수인 n만큼 생성되며, 영향관계에 따라 구성된다.

두 번째 단계인 makeAbstractGraph 함수는 생성된 Graph를 순회하면서 경합들간의 영향관계를 조사하여 상호 영향을 미치는 경합을 탐지하고, 이를 추상화하기 위해 인접리스트를 수정한다. 탐지된 상호 영향을 미치는 경합은 구체적으로 시각화하는 경우

```
1: Program RaceVisualization(TraceFile)
2:   n, pid := 0;
3:   initialize Check, Graph;  /* Graph is an adjacency list */
4:   makeOriginalGraph(TraceFile, n, Graph);
5:   for pid := 0 to n-1 do
6:     if Check(pid) = 0 then makeAbstractGraph(pid, Check, Graph);
7:   endfor
8:   displayAbstractGraph(Graph);
9: End RaceVisualization
```

<그림 3> 경합시각화 알고리즘



<그림 4 > 경합조건의 확장적 시각화

를 위해 Tangle 버퍼에 저장되며, 이러한 경합들 중에서 가장 작은 프로세스 번호를 가지는 경합으로 추상화된다. 인접리스트는 깊이 우선순위로 탐색되며, 한번 탐색한 노드에 대해서는 재탐색하지 않도록 하기 위해 탐색된 순서를 해당 Check 값으로 기록한다. 그리고 인접노드를 탐색하기 전에 Check값을 조사하여 재탐색하지 않도록 한다. 이러한 인접리스트에서는 자신의 Check값과 인접한 노드들의 Check값을 비교하여 상호 영향을 미치는 경합을 탐지하게 된다.

마지막 단계로 displayAbstractGraph 함수는 수정된 인접리스트를 순회하면서 경합을 노드로 표기하며, 경합들간의 영향관계는 방향성 있는 간선을 이용하여 시각화한다. 각 노드는 경합이 발생한 프로세스의 번호로 식별이 가능하고 상호 영향을 미치는 경합들은 추상화되어 일반 노드들과 달리 사각형으로 표현된다. 또한, 영향받지 않은 경합은 붉은 색으로 표현하고, 영향받은 경합은 파란색으로 표현한다. 추상화된 노드는 마우스 이벤트 발생시 Tangle 버퍼의 정보를 통하여 상호 영향을 미치는 노드들로 구체화된다.

3.2 확장적 시각화를 이용한 경합 탐지

이 기법은 각 프로세스에서 처음으로 발생하는 경합들을 추상적으로나 구체적으로 시각화하여 프로그램에서 처음으로 발생하는 경합들을 확장적이고 직관적으로 알 수 있게 하는 것으로 <그림 4>에서 적용 예를 보인다. 그림에서는 Build 메뉴를 이용하여 프로그램을 실행한 후 Visualize 메뉴를 통해 시각화되어진 경합들간의 영향관계를 Race View에서 보여주고 있다.

프로그램에서 처음으로 발생하는 경합인 최초경합은 가장 상위 수준의 붉은 노드로서 노드 2, 6, 9, 10이 이에 해당하며, 이들 노드에 들어오는 간선이 없으므로 영향받지 않은 경합임을 직관적으로 알 수 있다. 그리고, 본 기법에서는 이러한 경합들을 항상 최상위 수준에 표현함으로써 직관적 판단을 용이하게 한다. 노드 10은 다른 노드와 달리 사각형으로 표현된 추상화된 경합이다. Tangle View는 노드 10에 적용된 마우스 이벤트를 통해 생성된 것으로 노드 10을 구체적으로 표현한 것이다. 그림에서 노드 10과 11이 상호 영향을 미치는 경합임을 알 수 있으며. 나머지 파란색 노드들은 노드 2, 6, 9, 10으로부터 경로가 각각 존재하므로 영향받은 경합임을 쉽게 알 수 있다. 그러므로 이 기법은 최초경합을 직관적으로 탐지하여 수정할 수 있게 하므로 디버깅에 효과적이다.

4. 결론 및 향후과제

본 연구는 각 프로세스에서 처음으로 발생한 경합들 간의 영향관계를 추적하여 경합들간의 영향관계와 상호 영향을 미치는 경합을 추상화하여 시각화한다. 이 기법은 경합들의 집합을 추상적으로나 구체적으로 시각화함으로써 경합들간의 영향관계를 확장적이고 직관적으로 알 수 있게 한다. 그러므로 본 기법은 최초경합을 수정함으로써 영향받은 경합들을 사라지게 할 수 있으므로 메시지전달 프로그램의 효과적인 디버깅을 가능하게 한다. 향후 과제는 보다 효과적인 디버깅을 위해 시각화 정보와 프로그램 정보를 연계하여 사용자 인터페이스를 보완하는 것이다.

참고문헌

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

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

[3] Fidge, C. J., "Partial Orders for Parallel Debugging," *SIGPLAN/SIGOPS Workshop on Parallel and Distributed Debugging*, pp. 183-194, Madison, WI, May 1988.

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

[5] Kilgore, R., C. Chase, "Testing Distributed Programs Containing Racing Messages," *Computer Journal*, 40(8): 489-498, Cambridge Univ. Press, Aug., 1997.

[6] Kranzlmuler, D., S. Grabner, and J. Volkert, "Event Graph Visualization for Debugging Large Applications," *1st SIGMETRICS Symp. on Parallel and Distributed Tools (SPDT'96)*, pp. 108-117, ACM, Philadelphia, PA, May 1996.

[7] Kranzlmuller, D., and J. Volkert, "Why Debugging Parallel Programs Needs Visualization," *Workshop on Visual Methods for Parallel and Distributed Programming*, Seattle, WA, Sept. 2000.

[8] Netzer, R. H. B., and B. P. Miller, "Optimal Tracing and Replay for Debugging Message-Passing Parallel Programs," *Supercomputing 92*, pp. 502-511, Minneapolis, MN, Nov. 1992.

[9] Netzer, R. H. B., T. W. Brennan, and S. K. Damodaran-Kamal, "Debugging Race Condition in Message-Passing Programs," *SIGMETRICS Symp. on Parallel and Distributed Tools*, pp. 31-40, Philadelphia, PA, ACM, May 1996.

[10] Snir, M., S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra, *MPI-The Complete Reference: Volume 1, The MPI Core, 2nd Ed.*, Cambridge, MA, MIT Press, 1998.

[11] Tai, K. C., "Race Analysis of Traces of Asynchronous Message-Passing Programs," *17th Int'l Conf. on Distributed Computing Systems*, pp. 261-268, IEEE, May 1997.

[12] 박미영, 김영주, 김성대, 이승렬, 박소희, 전용기, "메시지전달 프로그램의 디버깅을 위한 경합조건의 시각화", 컴퓨터시스템 연구회 추계학술발표회논문집, pp. 47-56, 한국정보과학회, 2000. 9.