느슨한 메모리 동시성 모델

서울대학교 ▌허충길

이 글에서는 느슨한 메모리 동시성(Relaxed-Memory Concurrency)이 무엇인지, 그리고 이러한 동시성을 제대로 모델링하는 문제에 관해 소개한다. 이 문제는 40년이 넘은 프로그래밍 언어 분야에서 가장 중요한 난제 중 하나[2]였고, 최근 우리 연구실에서 해결하였다[4].

1. 메모리 모델

동시성 프로그램(Concurrent Program)에서 메모리는 어떻게 동작할까? 크게 보면 순차적 메모리 (Sequentially Consistent Memory)와 느슨한 메모리 (Relaxed Memory)가 있다.

1.1 순차적 메모리

순차적 메모리는 순차적 프로그램(Sequential Program)에서와 같은 방식으로 동작하는 메모리이다. 이는 직관적으로 가장 쉽게 이해할 수 있지만, 현실에서 이러한 방식으로 동작하는 시스템은 거의 없다. 주로 동시성 프로그램을 이론적으로 연구할 때 문제를 단순화시키기 위해 사용하는 모델이다.

예를 들어 아래와 같은 프로그램을 생각해보자.

Initially: X = Y = 0

```
[ Thread 1 ]        ||        [ Thread 2 ]
1:  X = 42                    3:  Y = 42
2:  a = Y                     4:  b = X
```

이 프로그램은 두 개의 스레드로 구성되어 있고, 0으로 초기화 되어있는 X, Y라는 공유 메모리 변수(대문자로 표시)를 통해 데이터를 주고받는다. a, b는 스레드 지역 변수(소문자로 표시)이다.

순차적 메모리 동시성에서는 두 스레드가 임의의 순서로 번갈아 가며 한 줄씩 실행이 되고, 공유 변수에서 값을 읽으면 그 변수에 마지막으로 저장된 값을 읽게 된다. 위 프로그램에서는 두 스레드가 실행되는 순서에 따라 최종적으로 [a = b = 42], [a = 0, b = 42] 또는 [a = 42, b = 0]의 결과를 얻을 수 있다. 다시 말해 어떤 순서로 수행되더라도 [a = b = 0]의 결과를 얻을 수는 없다.

1.2 느슨한 메모리

순차적 메모리와는 다르게 현실을 반영한 메모리 구조를 느슨한 메모리라 부른다. 실제 현실에서는 하드웨어(즉, CPU)와 컴파일러가 수행하는 최적화 때문에 순차적 메모리에서는 관찰할 수 없는 행동들, 즉 "느슨한" 행동들이 추가적으로 나타날 수 있다. 예로 현실에서 앞선 예제 프로그램을 실행하면 실제로 [a = b = 0]의 결과를 가끔 관찰할 수 있다.

2. 느슨한 메모리 모델이 허용해야 하는 행동들

제대로 된 느슨한 메모리 모델이 만족해야 할 첫 번째 조건은 실제 현실에서 나타날 수 있는 모든 느슨한 행동들을 포함하는 것이다. 이번 절에서는 하드웨어와 컴파일러 최적화에 의해 나타날 수 있는 느슨한 행동들의 대표적인 예를 소개한다.

2.1 쓰기 미루기 (Store Buffering)

쓰기 미루기는 말 그대로 쓰기(Store)를 나중에 수행하는 것이다. 이러한 행동은 x86, ARM, Power 하드웨어에서 모두 나타난다.

쓰기 미루기는 앞선 예제 프로그램을 통해 쉽게 이해할 수 있다. 실제로 앞 예제에서 [a = b = 0]이 관찰되는 이유가 쓰기 미루기 때문이다. 예를 들어 앞 예제의 Thread 1을 수행할 때 하드웨어는 첫 번째 명령어 X = 42 와 두 번째 명령어 a = Y 의 순서를 바꿔서 수행할 수 있다. 즉, 쓰기 (X = 42)를 미뤄서 수행하는 것이다. 이 경우 2 → 3 → 4 → 1 의 순서로 명령어가 수행될 수 있고, 이 때 최종 결과는 [a = b = 0]이 된다.

이렇게 쓰기 미루기를 할 수 있는 경우는 실행 순

서가 바뀌는 명령어가 서로 다른 메모리 영역을 참조하고, 또한 서로 데이터나 컨트롤 의존성(Data or Control Dependence)이 없을 때이다.

2.2 읽기 미루기 (Load Buffering)

읽기 미루기는 비슷하게 읽기(Load)를 나중에 수행하는 것이다. 이러한 행동은 x86에서는 발생하지 않지만 ARM과 Power에서는 나타난다.

좀 더 구체적으로 아래 예제 프로그램을 통해 설명하겠다.

Initially: X = Y = 0

```
[ Thread 1 ]          ||    [ Thread 2 ]
1:  a = X                   3:  b = Y
2:  Y = 42                  4:  X = b
```

이 프로그램은 순차 메모리에서는 어떤 순서로 실행하더라도 최종적으로 [a = b = 42]라는 결과를 얻을 수 없다. 하지만 Thread 1에서 읽기 미루기가 발생하면 그러한 행동이 나타날 수 있다. 즉, 읽기(a = X)가 미뤄져서 다음 명령어 Y = 42가 먼저 수행되면, 2 → 3 → 4 → 1 의 순서로 실행될 수 있고 이 경우 최종 결과는 [a = b = 42]가 된다.

앞서와 마찬가지로 읽기 미루기를 할 수 있는 경우는 실행 순서가 바뀌는 명령어가 서로 다른 메모리 영역을 참조하고, 또한 서로 데이터나 컨트롤 의존성이 없을 때이다.

2.3 컴파일러 최적화에 의한 느슨함

하드웨어뿐만 아니라 컴파일러 최적화에 의해서도 느슨한 행동이 발생할 수 있다. 아래의 예를 보자.

Initially: X = Y = 0

```
[ Thread 1 ]          ||    [ Thread 2 ]
1:  a = X                   3:  b = Y
2:  Y = a + 42 - a          4:  X = b
```

이 프로그램은 앞선 프로그램에서 Thread 1의 2번째 명령어만 살짝 바꾼 것이다. 하지만 이 경우 어떠한 하드웨어도 읽기 미루기를 수행하지 않는다. 그 이유는 두 번째 명령어가 첫 번째 명령어의 결과(즉, a에 저장된 값)를 사용해 둘 사이에 데이터 의존성이 생겼기 때문이다. 따라서 하드웨어에 의해서는 [a = b = 42]를 관찰할 수 없다.

하지만 컴파일러 최적화 후에는 하드웨어에서 [a = b = 42]를 관찰할 수 있다. 위 프로그램이 소스 프로그램일 경우 거의 모든 컴파일러는 Y = a + 42 - a 를 Y = 42 로 최적화 한다. 그 이유는 a가 스레드 지역변수라 다른 스레드에 의해 값이 바뀔 가능성이 없어 a + 42 - a 의 계산 결과는 항상 42가 되기 때문이다. 이렇게 최적화 된 이후에는 이전 예제 프로그램과 정확히 일치하므로 하드웨어에 의해 읽기 미루기가 발생하고, 따라서 [a = b = 42]가 관찰 가능하다.

이와 같은 이유로 [a = b = 42]가 위 예제 프로그램의 느슨한 행동으로 허용되어야 한다.

3. 느슨한 메모리 모델이 허용하지 않아야 하는 행동들

제대로 된 메모리 모델은 하드웨어나 컴파일러에 의해 발생할 수 있는 느슨한 행동을 모두 포함해야 하지만, 그렇다고 난데없는(out of thin air) 행동을 포함해서는 안 된다. 즉, 우리가 상식적으로 기대하는 행동들만 포함해야한다.

대표적으로 데이터 경쟁(Data race)이 없는 프로그램에서 느슨한 행동이 나타나는 것은 난데없는 행동이다. 그 이유는 스레드 간 경쟁이 없는데도 불구하고 느슨한 행동이 나타나는 것은 비상식적이기 때문이다. 예를 들어 보자.

Initially: X = Y = 0

```
[ Thread 1 ]          ||    [ Thread 2 ]
1: a = X                    4: b = Y
2: if (a != 0)              5: if (b != 0)
3:    Y = 42                6:    X = 42
```

이 프로그램을 “순차적 메모리”에서 수행해보면, 어떤 순서로 수행하더라도 두 스레드가 공유 메모리 변수(즉, X 또는 Y)를 동시에 참조하려는 상태는 나타나지 않는다. 이러한 프로그램을 데이터 경쟁이 없는 프로그램이라고 부른다. 이렇게 데이터 경쟁이 없는 프로그램에서는 느슨한 행동(즉, 순차적 메모리에서는 나타나지 않는 행동)이 나타나지 않아야 한다. 이 프로그램의 경우 순차적 메모리에서는 어떤 순서로 수행되더라도 항상 최종 결과로 [a = b = 0]만 얻는다. 따라서 이외의 다른 결과(예로 [a = b = 42])를 얻는 것은 난데없는 행동이고 이는 느슨한 메모리에서도 허용되지 않아야 한다.

실제 현실에서도 데이터 경쟁이 없는 경우는 느슨한 행동이 나타나지 않는다. 예로 위 프로그램의 경우 컨트롤 의존성 때문에 하드웨어에서 읽기 미루기가

발생하지 않는다. 좀 더 구체적으로, Thread 1의 경우 Y = 42를 수행할지 결정하기 위해서는 a의 값을 알아야 한다. 따라서 a = X 와 Y = 42 사이에 컨트롤 의존성이 있다. 또한 어떠한 컴파일러 최적화에 의해서도 이 컨트롤 의존성은 없어지지 않는다. 같은 이유로 Thread 2에서도 읽기 미루기가 발생하지 않는다. 따라서 위 프로그램을 어떠한 하드웨어에서 컴파일해서 실행하더라도 항상 [a = b = 0]만 관찰된다.

이와 같이 순차적 메모리에서 데이터 경쟁이 없는 프로그램은 느슨한 메모리에서의 행동이 순차적 메모리에서의 행동과 일치해야 한다. 이를 무경쟁(Data Race Freedom, 짧게 DRF)정리라 부른다. 이러한 성질 덕분에 프로그래머가 데이터 경쟁이 없는 프로그램을 작성할 때는 느슨한 행동에 대한 걱정 없이 순차적 메모리에서만 생각하면 된다.

4. 느슨한 메모리 모델에 관한 연구

지금까지 느슨한 메모리 모델에 요구되는 가장 기본적인 성질들만 알아보았다. 실제로는 동기화를 위한 더 많은 기능들을 지원해야 하고, 또한 이 글에서 소개하지 않은 더 많은 성질들을 만족해야 한다.

이와 같은 요구조건을 모두 만족시키는 제대로 된 느슨한 메모리 모델을 개발하는 것은 40년이 넘은 오래된 중요한 난제 중 하나였다. 다음은 Batty el. al.의 논문[2]에서 취한 인용문이다.

> Disturbingly, 40+ years after the first relaxed-memory hardware was introduced (the IBM 370/158MP), the field still does not have a credible proposal for the concurrency semantics of any general-purpose high-level language that includes high-performance shared-memory concurrency primitives. This is a major open problem for programming language semantics.

실제로 현재 Java 언어를 위한 공식적인 메모리 모델[5]은 Java 컴파일러 최적화에 의해 발생하는 느슨한 행동을 모두 포함하지 못하는 문제를 가지고 있다[6]. 반면에 C/C++ 언어를 위한 공식적인 메모리 모델[1]은 난데없는 행동을 포함하는 문제를 가지고 있다[3].

최근 우리 연구실에서 주도해 이러한 문제를 모두 해결하는 획기적인 메모리 모델을 개발했고, 요구되는 성질을 모두 만족한다는 것을 Coq 증명보조기를 이용해 대부분 증명했다[4]. 또한 우리가 개발한 모델을 기반으로 Java 공식 메모리 모델을 재정의 하는 방향으로 Java 메모리 모델 위원회와 현재 협의 중에 있다. 우리가 개발한 메모리 모델에 관한 내용은 아래의 논문에 자세히 나와 있다.

> Jeehoon Kang, Chung-Kil Hur, Ori Lahav, Viktor Vafeiadis, Derek Dreyer. A promising semantics for relaxed-memory concurrency. In POPL, 2017.

이 연구에 의해 느슨한 메모리 모델의 가장 어려운 난제(즉, 난데없는 행동 없이 모든 느슨한 행동을 포함하는 문제)는 해결되었다.

이제는 몇몇 빠진 기능을 더 추가하고, 또한 이 모델 위에서 프로그램 검증을 위한 논리 및 분석 도구를 개발하는 방향으로 연구를 진행 중이다.

참고문헌

[1] ISO/IEC 14882:2011. Programming language C++, 2011.

[2] Mark Batty, Kayvan Memarian, Kyndylan Nienhuis, Jean Pichon-Pharabod, Peter Sewell. The problem of programming language concurrency semantics. In ESOP, 2015.

[3] Hans-Juergen Boehm, Brian Demsky. Outlawing ghosts: Avoiding out-of-thin-air results. In MSPC, 2014.

[4] Jeehoon Kang, Chung-Kil Hur, Ori Lahav, Viktor Vafeiadis, Derek Dreyer. A promising semantics for relaxed-memory concurrency. In POPL, 2017.

[5] Jeremy Manson, William Pugh, Sarita V. Adve. The Java memory model. In POPL, 2005.

[6] Jaroslav Ševčík, David Aspinall. On validity of program transformations in the Java memory model. In ECOOP, 2008.

약 력

허충길

2000 KAIST 전산학과 및 수학과 졸업 (학사)
2010 영국 University of Cambridge 졸업 (박사)
2009~2010 프랑스 Laboratoire PPS 박사후 연구원
2010~2012 독일 Max Planck Institute for Software Systems 박사후 연구원
2012~2013 영국 Microsoft Research Cambridge 박사후 연구원
2013~현재 한국 서울대학교 컴퓨터 공학부 조교수
관심분야: 소프트웨어 검증, 프로그래밍 언어, 컴파일러
Email: gil.hur@sf.snu.ac.kr