VIS를 이용한 RACE 프로토콜의 정형검증

엄 현 선[†] · 최 진 영[††] · 한 우 종[†††] · 기 안 도[††††] · 심 규 현[†††††]

요 약

다중 프로세서 시스템에서 각각의 프로세서에 할당되어 있는 지역 캐쉬에 데이터의 복사본이 분산 공유되어 있는 경우 데이터의 일관성 유지가 필요하다. 따라서 캐쉬 일관성 유지 프로토콜은 공유 메모리 다중 프로세서 시스템의 정확하고 효율적인 작동에 중요하다. 그러므로 시스템이 복잡해짐과 비례하여 현재 사용되고 있는 무작위적 테스트나 시뮬레이션은 프로토콜의 정확성을 확인하기에 충분하지 못하므로 보다 효율적이고 믿을 만한 검증 방법이 필요하다. 본 논문은 ETRI에서 개발된 캐쉬 일관성 프로토콜인 RACE(Remote Access Cache coherent Enforcement)프로토콜의 몇 가지 특성(property)들을 정형기법에 쓰이는 도구 중의 하나인 VIS(Verification Interacting with Synthesis)를 이용하여 검증한다.

Formal Verification of RACE Protocol Using VIS

Hyun-Sun Um[†] · Jin-Young Choi[††] · Woo-Jong Han[†††] ·
An-Do Ki[††††] · Kyu-Hyun Shim[†††††]

ABSTRACT

Caches in a multiprocessing environment introduce the cache coherence problem. When multiple processors maintain locally cached copies of a unique shared-memory location, any local modification of the location can result in a globally inconsistent view of memory. Cache coherence protocols are important to operate a shared-memory multiprocessor system with efficiency and correctness. Since random testing and simulations are not enough to validate correctness of protocols, it is necessary to develop efficient and reliable verification methods. In this paper we present our experience in using VIS (Verification Interacting with Synthesis), a tool of formal method, to analyze a number of property of a cache coherence protocol, RACE (Remote Access Cache coherent Enforcement).

1. 서 론

병렬 처리는 현대 컴퓨터 기술에서 고성능, 낮은 비용, 실제 응용에서의 생산성에 대한 요구가 많아짐에 따라 이를 해결할 수 있는 주요 기술로 개발되었다. 병렬 컴퓨터는 구조적으로 공유 메모리(shared memory)를 가지는가 아니면 비공유 분산 메모리(unshared distributed memory)를 가지는가에 따라 두 부류로 나눌 수 있다. 공유 메모리 다중 프로세서 모델은 메모리와 주변 자원들이 어떻게 공유 또는 분산되어 있는가에 따라 다시 UMA(Uniform-Memory-Access)모델, NUMA(Non Uniform-Memory-Access)모델, COMA (Cache-Only Memory Architecture)모델로 분류된다. 여기서 프로세서가 각각의 캐쉬와 연관되어 있을 때 UMA와 NUMA모델을 특히 CC-UMA와 CC-NUMA [14]라고 한다.

본 논문에서는 ETRI에서 개발된 디렉토리 기반의 CC-NUMA시스템의 캐쉬 일관성 프로토콜인 RACE프

※ 본 논문은 1998년 한국전자통신연구원 위탁과제 98241의 지원으로 연구 되었음
† 준 회 원 : 고려대학교 대학원 컴퓨터학과
†† 정 회 원 : 고려대학교 컴퓨터학과 교수
††† 정 회 원 : 한국전자통신연구원 병렬시스템연구팀장 책임연구원
†††† 정 회 원 : Dynalith Systems, Inc의 R&D Director(책임연구원)
††††† 정 회 원 : 한국전자통신연구원 선임연구원
논문접수 : 2000년 2월 22일, 심사완료 : 2000년 6월 19일

로토콜[22]을 정형기법을 이용하여 검증하고자 한다. 에러를 발견하고 나서 수정하는 것이 어렵고 비용 또한 많이 들기 때문에 시스템을 구현하기 전에 어떤 설계의 정확성을 확인하는 것은 중요하다. 현재 설계 단계에서 에러를 찾기 위해 사용하는 방법인 시뮬레이션은 단순한 개념이지만 불완전하다는 문제를 가지고 있다. 이와는 달리 정형기법[3]은 쉽게 찾아내기 힘든 불일치성, 애매모호함, 불완전성을 나타내 보임으로써 보다 완벽한 시스템을 구축 할 수 있게 한다. 특히 근래에 자동적으로 검증 할 수 있는 도구들이 개발되고 있어 학계에서 뿐만 아니라 산업계에서 설계하는데 이용할 수 있기 때문에 최근 들어 상당한 관심을 받아 오고 있다. 정형기법은 정형 명세와 정형 검증으로 나뉘고 정형 검증에서 최근 많이 사용되고 있는 모델 체킹은 시스템을 명세언어로 명세하고 명세한 시스템 모델에 대해 궁극적으로 검증하려는 특성이 만족하는지를 CTL[13]과 같은 논리식으로 표현하여 검증하는 기법이다. VIS[20]는 CTL모델 체킹[13] 과 language emptiness검사 등을 할 수 있는 정형 검증 도구이다.

본 논문에서는 VIS를 이용하여 캐쉬 일관성 프로토콜인 RACE 프로토콜의 몇 가지 특성들을 검증한 사례를 제시한다. 2장에서 CC-NUMA시스템에 대한 소개를 하고 3장에서는 관련 연구에 대한 설명, 4장에서는 VIS에 대해 설명하고 5장에서는 RACE프로토콜에 대한 대략적인 소개를 하고 6장에서 RACE프로토콜 검증에 대한 가정과 추상화 모델, 구현, 검증 결과에 대한 설명을 하고 7장에서 검증 결과에 대한 결론을 맺고 모델 체킹 검증 기법의 장점과 향후 과제를 제시한다.

2. CC-NUMA 시스템

UMA모델에서는 물리적 메모리는 모든 프로세서가 단일하게 공유한다. 그리고 모든 프로세서는 모든 메모리 위치에 대해 동일한 접근 시간을 갖는다. 각 프로세서는 독자적인 캐쉬를 가질 수 있다. 주변 자원 또한 여러 가지 방법으로 공유된다. 그러나 NUMA모델에서는 메모리 접근 시간이 메모리 워드의 위치에 따라 다르다. 각 프로세서가 공유메모리 주소 공간을 쪼개어 물리적으로 연관되어 있기 때문이다. a라는 메모리 주소를 포함하고 있는 메모리 모듈을 a의 홈 메모리(home memory)라고 한다. 그 홈 메모리에 해당되는 노드를 홈 노드(home node)라고 한다. 따라서 지역 프로세서에 의한 지역 메모리의 접근이 훨씬 빠르다. 다른 프로세서에 할당되어 있는 외부 메모리의 접근은 인터커넥션 네트워크를 거쳐 가야 하기 때문에 시간이 더 걸린다. 분산 메모리 뿐 아니라 전체적으로 공유된 메모리도 다중 프로세서 시스템에 사용될 수 있다. 이런 경우에 세가지 메모리 접근 형식이 있다. 가장 빠른 접근은 지역 메모리 접근(local memory access)이고 그 다음은 전역 메모리 접근(global memory access), 상대적으로 가장 느린 접근은 외부 메모리 접근(remote memory access)이다.

이러한 모델들은 미리 지정된 접근 권한을 가진 공유 메모리와 전용 메모리의 혼합된 형태로 쉽게 변형할 수 있다. 계층적으로 구성된 다중 프로세서 모델은 다음과 같다. 각 클러스터 자체는 UMA 또는 NUMA 다중 프로세서이다. 클러스터는 전역 공유 메모리(global shared memory)모듈과 연결 된다. 전체 시스템을 NUMA 다중 프로세서로 생각한다. 같은 클러스터에 속하는 모든 프로세서는 동일하게 클러스터 공유 메모리 모듈에 접근 할 수 있다. 모든 클러스터는 전역 메모리에 동일한 비율로 접근한다. 그러나 클러스터 메모리에의 접근 시간은 전역 메모리에의 접근 시간보다 짧다. 클러스터 메모리 접근 권한을 여러 가지 방법으로 지정할 수 있다. 위에서 규정한 UMA, NUMA 모델 이외에 다른 모델이 다중 프로세서를 위해 존재한다. 예를 들면 캐쉬 일관성 비단일 메모리 접근(CC-NUMA)모델을 분산 공유 메모리와 캐쉬 디렉토리로 규정 할 수 있다. CC-NUMA모델의 예로는 Stanford Dash 와 MIT Alewife가 있다.

3. 관련 연구

하드웨어나 소프트웨어 시스템은 불가피하게 규모, 기능 면에서 점점 더 복잡해지고 있다. 이러한 복잡성의 증가 때문에 작은 에러가 존재할 가능성 역시 커지고 있다. 이러한 복잡한 시스템에 대해서 과거와 같이 테스트나 시뮬레이션만으로는 에러를 찾아내기 힘들다. 또한 작은 에러들이 발생함으로써 전체 시스템에 엄청난 오류를 발생하게 하여 심각한 피해를 입는 사례들도 곳곳에서 나타나고 있다. 따라서 하드웨어 설계 회사들은 점차로 기존에 사용되던 시뮬레이션 방법을 대체할 방법으로 정형검증에 대한 관심을 갖고 있다.

정형기법은 수학과 논리학에 기반을 둔 방법으로 하

드웨어나 소프트웨어 시스템을 명세하거나 검증하는 것이다. 수학적 기호를 사용하여 시스템의 명세를 하고 검증할 특성 또한 논리로 기술하여 시스템이 특성을 만족하는지를 수학적 성질을 이용하여 검증하므로 자연어가 내포하는 애매모호함이나 불확실성을 최소한 줄일 수 있다. 따라서 복잡한 시스템에 어떤 특성이 만족하는지를 검증하여 최소한으로 검증된 특성에 대해서는 믿을 수 있고 사용할 수 있다.

정형 기법은 정형 명세와 정형 검증으로 나뉜다. 정형 명세는 시스템이 달성해야 할 요구사항과 그러한 요구사항을 완성 할 수 있는 설계를 묘사하는 데 목적이 있다. 그리고 정형 검증은 명세가 정확한지, 즉 설계가 요구사항을 만족하는지를 증명하는데 목적이 있다. 정형 기법은 쉽게 찾아내기 힘든 불일치성, 애매모호함, 불완전성을 나타내 보임으로써 보다 완벽한 시스템을 구축 할 수 있게 한다.

모델체킹은 정형 검증의 한 분야로 시스템을 어떤 입력 언어로 명세하고 검증하려는 특성을 CTL과 같은 논리식으로 표현한 후 명세 언어로 명세한 시스템 모델에 대해 검증하려는 특성이 만족하는지를 검증하는 기법이다. 모델체킹 방법은 완전히 자동화 될 수 있으며 이해가 쉽다는 장점이 있다. 그러나 모델체킹 방법은 상태 공간 폭발(state space explosion) 문제와 검증 시간이 많이 걸린다는 문제에 직면하고 있다. 따라서 많은 대학과 연구 기관에서 이러한 문제를 해결 하기 위한 연구가 계속 되고 있다. 많은 프로토콜들이 이 방법을 이용해서 그 정확성을 검증하고 있으며 캐쉬 일관성 프로토콜 역시 이 방법을 사용하고 있다.[1, 5--10, 12, 15-17] 캐쉬 일관성 프로토콜을 검증한 예 중에 몇 가지를 소개 해 보자면 다음과 같다.

SSM(Symbolic State Model)[4]은 USC(University of Southern California)의 Fong Pong이 개발한 툴로써 상태 나열 방법을 이용했지만 특별한 추상화 과정을 통해서 상태 공간 폭발 문제를 해결했다는 점에서 주목할 만한 툴이다. SSM으로 Illinois 프로토콜을 검증했으며 그 외에 다수의 버스 기반 일관성 유지 프로토콜을 검증했다. 또한 디렉토리 기반 프로토콜중에서는 USC에서 개발한 CC-NUMA 시스템의 Rapid Prototyping engine for Multiprocessor(RPM)프로토콜[15]과 chained 디렉토리 방식의 S3.mp프로토콜을 검증했다.

CMU의 K. L. McMillan에 의해 개발된 툴인 SMV (Symbolic Model Verifier)는 동기적인 시스템에서부터 비동기적인 시스템에 이르기 까지 SMV입력언어로 시스템을 모델링 한 후 검증하려는 특성을 CTL로 표현한 후 모델 체킹 방법을 사용하여 검증한다. [10, 13] CTL논리식은 safety, liveness, fairness, deadlock등과 같은 속성들을 표현할 수 있다. 1992년에 CMU의 Clarke는 SMV를 이용하여 IEEE Futurebus + Standard 프로토콜을 검증하였다. SMV를 이용하여 프로토콜의 설계 시 발견하지 못했던 에러들과 잠재적인 에러들을 발견하였다. 이는 아마도 자동 검증 도구를 사용하여 IEEE표준에서 에러를 찾아낸 첫번째 사례일 것이다.

Murφ는 Stanford 대학의 하드웨어 검증 그룹의 David L. Dill에 의해 개발된 툴이다. Dill은 1992년 Murφ를 이용하여 SCI(Scalable Coherent Interface, IEEE std 1596-1992)프로토콜을 검증하였다.[11, 21] Dill은 검증을 위해 먼저 프로토콜을 Murφ기술 언어로 표현한 후 캐쉬 일관성에 대한 명세를 작성하였다. 전체 상태의 수가 매우 커지므로 시스템의 아주 작은 부분만 정의 하였지만 변수 초기화부터 미묘한 논리적 에러까지 프로토콜에서의 에러를 발견하였다.

4. VIS (Verification Interacting with Synthesis)

4.1 VIS에 관한 소개

VIS는 정형 검증, 시뮬레이션, 유한 상태 하드웨어 시스템의 합성(synthesis)을 종합한 도구이다. 이 도구는 입력 언어로 Verilog[2]를 사용하고 fair CTL 모델 체킹, language emptiness 검사, 조합 순차 동등성 검사(combinational and sequential equivalence checking), 원형 기반 시뮬레이션(cycle-based simulation), 계층적 합성(hierarchical synthesis)을 지원한다. VIS는 버클리 대학과 콜로라도 대학(Boulder)이 협력하여 개발되고 있으며 1세대 도구인 HSIS나 SMV에 비교하여 보다 개선된 프로그래밍 환경을 제공하고 새로운 호환성을 제공하며 어떤 경우에는 성능 또한 향상 시켜줄 수 있다.

VIS는 BLIF-MV라는 중간 형태에서 작동하며 BLIF-MV 파일은 VL2MV라 불리는 컴파일러에 의해 생성된다. VIS에서 사용하는 Verilog가 일반적인 Verilog와 다른 점은 비결정성(nondeterminism)과 기호적 변수(symbolic variables)를 사용할 수 있다는 점이다. 비결정성 구성자인 $ND는 wire변수에서의 비결정성을 나타내기

위해서 Verilog에 추가 되었고 비결정성을 자연스럽게 사용할 수 있도록 한다. 이는 환경에 대한 명세를 할 때 필요하다. 또한 VL2MV가 Verilog에서 C에서 사용하는 것과 유사한 열거형을 사용하여 기호적 변수를 사용 할 수 있도록 하여 변수의 값을 기호적으로 명세하고 참조하는 것을 직접적으로 표현할 수 있게 하였다. 따라서 상위 레벨의 명세를 쉽게 하였다. BLIF-MV묘사가 VIS로 읽혀질 때 계층적 트리 형태로 저장된다. 이는 차례로 부모들(sub module)을 구성하고 모듈의 기능은 arbitrary functionality와 latch를 가진 게이트의 네트워크에 의해 나타내어진다. 이 계층에 대한 순회(traversal)는 UNIX의 디렉토리에 대한 순회와 유사하고 시뮬레이션과 검증 작업은 어느 계층에서나 가능하다.

4.2 VIS를 이용한 검증

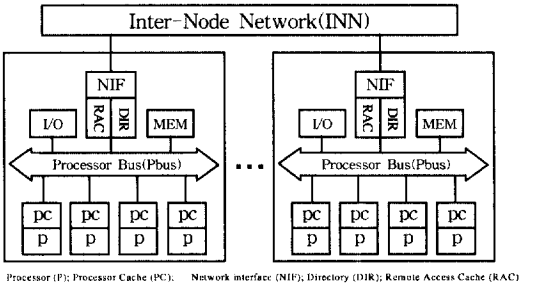
정형 검증은 디자인이 어떤 요구에 만족하는지를 체크하는 과정이다. 디자인을 정형적으로 검증하기 위해서는 먼저 검증 할 수 있는 형태로 바꿔야 한다. 상태와 상태 사이의 전이는 유한 상태 기계(finite state machine)로 구성 되고 완전한 시스템은 각 구성 요소와 연관된 유한 상태 기계를 구성함으로써 얻어지는 유한 상태 기계가 된다. 그러므로 검증할 때 첫번째 단계는 시스템을 유한 상태 기계로 나타내는 것이다. 현재 상태가 주어졌을 때 유한 상태 기계의 다음 상태는 현재 상태와 입력의 함수로써 만들어진다. 이러한 완전한 프레임워크는 이산함수(discrete function)의 하나이며 이산함수는 BDDs(Binary Decision Diagram)와 MDDs(Multi-valued Decision Diagram)에 의해 쉽게 나타낼 수 있다.[18-19] 유한 상태 기계로 나타낸 시스템을 VIS의 입력 언어인 Verilog로 설계 한 후 VIS를 이용하여 현재 자동 정형 검증에서 가장 많이 쓰이는 두 가지 방법인 language containment와 모델 체킹(model checking) 방법으로 검증한다.

5. RACE 프로토콜(Remote Access Cache coherent Enforcement Protocol)

CC-NUMA의 일종인 PDLSystem(Physically-Distributed but Logically-Shared memory system)[22]은 (그림 1)과 같이inter-node 네트워크를 통해 연결되어 있는 많은 프로세싱 노드(processing node)들로 구성되어 있다. 프로세싱 노드는 개인 캐쉬(private cache)를 갖는 프로세서와 전역 공유 메모리 공간(globally-shared memory space)의 일부분인 메모리 모듈, I/O, 외부접근 캐쉬(RAC)와 디렉토리를 갖는 네트워크 인터페이스로 구성된다.

프로세싱 노드 안의 프로세서 캐쉬들은 무효화 기반(invalidate-based) snoopy 캐쉬 일관성 유지 프로토콜에 의해 일관성을 유지한다. 프로세서 캐쉬에 없는 데이터의 접근 시 해당 캐쉬 라인을 가져 오기 위해서 프로세서 버스에 요청을 보낸다. 그 위치가 처리 노드 안의 지역 메모리에 있다면 메모리가 요청 데이터를 제공한다. 그렇지 않은 경우, 외부 접근 캐쉬가 그 요청을 처리한다. 외부 접근 캐쉬가 유효한 데이터를 갖고 있는 경우는 그 데이터를 제공하고 그렇지 않은 경우 네트워크를 통해서 그 위치가 있는 외부 메모리로부터 데이터를 가지고 온다. 처리 노드의 외부로부터의 접근은 공유 정보를 가지고 있는 디렉토리에 의해 관리된다.

다중 프로세서 시스템에서 캐쉬의 데이터 일관성을 위한 프로토콜인 RACE프로토콜은 한국 전자 통신 연구원의 컴퓨터 소프트웨어 기술 연구소에서 개발한 것으로 디렉토리 기반 일관성 유지 프로토콜이다.



(그림 1) PDLSystem

5.1 개요

프로세서가 캐쉬 미스를 발생하면 홈이 지역적(local)인지 외부적(remote)인지를 판단하기 위해 요청된 캐쉬 라인을 가지고 있는 메모리 블록의 주소를 참조한다. 지역적이면 지역 메모리가 프로세서 캐쉬에 데이터를 제공하고 아니면 외부 접근 캐쉬가 프로세서 캐쉬에 데이터를 제공한다. 요청된 캐쉬라인이 지역 메모리에 위치해 있더라도 디렉토리는 외부 노드의 외부 접근 캐쉬가 복사본을 가지고 있기 때문에 데이터 일관성을 유지하기 위해 참조 해야 한다. 외부 접근

캐쉬가 요청된 데이터를 제공할 수 있기 위해서는 외부 접근 캐쉬가 데이터를 가지고 있어야 한다. 그렇지 않다면 요청 노드에 있는 외부 캐쉬는 홈 노드에 요청을 보낸다. 홈 노드가 데이터의 소유권을 가지고 있다면 디렉토리는 그 지역 메모리로부터 그 블록을 받는다. 그렇지 않다면 홈 노드에 있는 디렉토리가 소유권을 가지고 있는 노드에게 요청을 전달하고 홈 노드에게 작업의 완료를 알린다. 그 다음 요청 노드의 외부 캐쉬가 그 데이터를 프로세서 캐쉬에 제공한다. 네트워크 관점에서 RACE 프로토콜을 설명하면 크게 지역적 접근(local access)과 외부적 접근(remote access)으로 나눌 수 있다.

5.2 가정

본 논문에서 검증할 RACE프로토콜은 다음과 같은 가정을 따른다.

- 외부 접근 캐쉬에서 일어나는 모든 외부 접근 미스는 우선 홈 노드로 전해진다.
- reply-forwarding이 사용된다.
- Inter-node 네트워크를 통한 데이터 전송 프로토콜은 한 노드에서 다른 노드로 보내는 메시지들의 순서를 보장한다.
- Ghost-sharing이 허용된다.

5.3 RACE 프로토콜의 operation

5.3.1 Intra-node network operation

- coherent read : 캐쉬 라인을 읽기 전용으로 요구하는 경우. 요구한 프로세서는 이 데이터를 수정하지 않는다.
- exclusive read : 캐쉬 라인을 쓰기 가능으로 요구하는 경우. 요구한 프로세서는 이 데이터를 수정 할 수 있다.
- invalidate : 다른 모든 캐쉬의 복사본을 무효화한다. 요구한 프로세서는 이 데이터를 수정 할 수 있다.
- writeback : 캐쉬 라인을 메모리에 쓴다. 프로세서 캐쉬의 replacement에 의해 발생한다.
- writethrough : 캐쉬 라인을 메모리에 쓴다. 프로세서 캐쉬가 그 데이터를 가지고 있을 수도 있고 가지고 있지 않을 수도 있다.

이 밖에도 locked read, locked write, uncached read, uncached write가 있다.

5.3.2 Inter-node network operation

- coherent read : 캐쉬 라인을 읽기 전용으로 요구하는 경우.
- exclusive read : 캐쉬 라인을 쓰기 가능으로 요구하는 경우.
- invalidate : 다른 모든 캐쉬의 복사본을 무효화한다.
- writeback : 캐쉬 라인을 메모리에 쓴다. 일반적으로 replacement에 의해 발생한다.

이 밖에도 uncached read, uncached write, input /output read, input/output write가 있다[22].

5.4 네트워크 관점에서의 RACE 프로토콜

5.4.1 지역적 접근

디렉토리가 uncached 상태에서 프로세서 버스 상의 coherent local request가 발생하는 경우 디렉토리는 그 요청을 처리할 필요가 없다. 디렉토리가 shared 상태에서 요청이 발생하는 경우에도 디렉토리는 그 요청을 처리할 필요가 없다. 그러나 디렉토리는 프로세서 캐쉬가 exclusive 상태에서 shared 상태로 바뀌었는지 확인해야 한다. 디렉토리가 uncached상태에서 프로세서 버스 상의 exclusive local request가 발생하는 경우 디렉토리는 그 요청을 처리 할 필요가 없다. 디렉토리가 shared 상태에서 exclusive local request가 발생하는 경우 디렉토리 컨트롤러는 무효화 메시지를 모든 공유 노드들에 보낸다. 이 무효화 메시지를 받자마자 각 공유 노드들은 복사본을 무효화 시키고 홈 노드에 확인 메시지를 보낸다. 그 다음 각 공유 노드는 외부 접근 캐쉬의 상태를 invalid로 바꾼다. 모든 무효화 확인 메시지가 도착되면 디렉토리는 uncached상태로 바뀐다. 최종적으로 요청 노드의 디렉토리는 쓰기 가능 독점권을 갖는다. 지역 접근에는 coherent local request, exclusive local request뿐 아니라, invalidate local request, writeback local request도 포함된다.

5.4.2 외부적 접근

외부 접근 캐쉬가 shared 상태에서 coherent remote

request가 발생하는 경우 외부 접근 캐쉬는 요청 데이터를 공급한다. 외부 접근 캐쉬는 modified 상태에서 coherent remote request가 발생하는 경우도 외부 접근 캐쉬가 요청 데이터를 공급한다. 외부 접근 캐쉬가 modified 상태에서의 경우 coherent remote request가 발생하는 경우 외부 접근 캐쉬가 요청 데이터를 공급한다. 외부 접근 캐쉬가 shared 상태에서 exclusive remote request가 발생하는 경우 외부 접근 캐쉬는 무효화 요청 메시지를 홈 노드에 보낸다. 홈 노드는 그 요청을 받자마자 모든 공유 노드에 무효화 요청을 보낸다. 또한 프로세서 캐쉬를 무효화 시킨다. 모든 공유 노드에 무효화 요청을 보낸 후에 디렉토리는 요청 노드로 하여금 쓰기 가능 독점권을 얻게 한다. 그러나 디렉토리는 모든 무효화 확인 메시지가 올 때까지 기다려야 한다. Exclusive remote request시 외부 접근 캐쉬 미스가 발생하면 쓰기 요청 메시지를 홈 노드에 보낸다. 홈 노드의 디렉토리 상태가 uncahed상태이면 디렉토리는 프로세서 캐쉬로부터 독점적 복사본을 받고 요청 노드에 그 복사본을 보낸다. 홈 노드의 디렉토리가 shared 상태이면 디렉토리는 모든 공유 노드에 무효화 요구 메시지를 보낸다. 동시에 홈 노드는 독점적 복사본을 받고 요청 노드에 그 복사본을 보낸다. 홈 노드의 디렉토리가 modified상태이면 디렉토리는 요청을 점유 노드에 전달한다. 점유 노드의 외부 접근 캐쉬가 그 요청을 받자마자 프로세서 캐쉬로부터 가장 최근의 수정 데이터를 가져와서 요청 노드에 보낸다. 동시에 점유 노드는 홈 노드에 동작이 완료 되었음을 알린다. 외부 접근 캐쉬가 shared 상태에서 invalidate remote request가 발생하는 경우 외부 접근 캐쉬는 무효화 요청 메시지를 홈 노드에 보낸다. 홈 노드가 그 요청을 받자마자 무효화 요청을 모든 공유 노드에 보내고 홈 노드의 프로세서 캐쉬를 무효화 시킨다. 모든 공유 노드에 무효화 요청을 보낸 후, 디렉토리는 요청 노드로 하여금 쓰기 가능 독점권을 얻게 한다. 외부 접근에는 coherent remote request, exclusive remote request뿐 아니라 invalidate remote request, writeback remote request도 포함된다.

5.4.3 Replacement of Cache blocks

캐쉬에서의 새로운 공간 확보를 위해서 replacement를 할 필요가 있다 shared 상태이거나 modified 상태인 캐쉬 블록이 replacement의 대상이 된다. 블록이 shared상태인 경우, 외부 접근 캐쉬는 그 블록을 제거하고 그와 같은 사실을 홈 노드에 알리지 않는다. 블록이 modified 상태인 경우 외부 접근 캐쉬는 그 블록을 제거하고 제거 사실을 홈 노드에 알린다. replacement가 일어나 블록은 invalid상태로 간다.

5.4.4 Concurrent accesses

요청이 같은 address에 대해 병렬적으로 일어날 때 concurrent 요청에 대하여 다음과 같이 작동한다.

1) Pending state에서 hit한것을 제외한 nak은 즉시 retry해야한다.
2) Pending state 에 hit한 CRDq와 ERDq같은 요청은 nak을 보내야 된다. 요청자가 DIR 이라면 즉시 원래 요청자에게 retry해야한다. 요청자가 RAC이라면 어느 정도 시간이 지난 후에 retry해야한다.
3) rac에서 pending state에 hit한 INVq는 보통으로 인식하고 요청자는 INVp를 받는다. DIR에서 pending stated에 hit한 invalidation은 nak을 보내야 한다.
4) DIR에서 hit한 WRBq 는 홈 메모리를 즉시 수정해야 하는 대신에 nak을 보내지 말아야 한다.

6. RACE 프로토콜의 검증

6.1 가 정

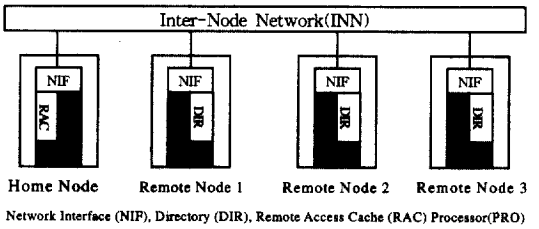
본 논문에서는 검증의 대상을 다음과 같이 제한한다.

1) 검증의 범위를 하나의 메모리 워드에 대한 검증만으로 제한한다. 서로 다른 메모리 워드는 서로의 상태 전이에 영향을 미치지 않기 때문에 하나의 메모리 워드에 대한 검증으로 이번 모델에 대한 검증은 충분하다.
2) 프로세스 캐쉬와 메모리는 추상화 시킨다.
3) 프로세스 캐쉬에서 미스 되어 나오는 요청들은 비결정적(non deterministic)으로 발생시킨다.

6.2 추상화 모델

모델 체킹은 상태 공간 폭발 문제 때문에 검증 할 때 시스템 자체를 직접 사용할 수 없다. 이 문제를 해결하기 위해 원래 시스템의 행위는 유지하면서 상태 공간을 줄이기 위해 추상화 모델을 만든다. 이러한 추상화 모델 또한 상태 공간 폭발 문제를 완전히 해결하

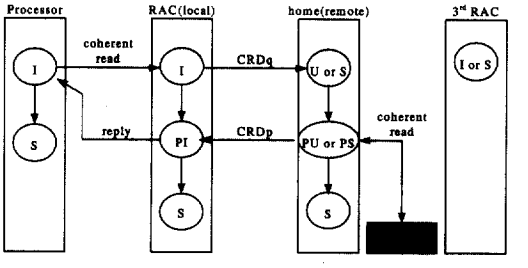
지는 못하지만 큰 시스템을 모델링 하는데 많은 도움이 된다. 캐쉬 일관성 시스템에서 추상화에 관해 여러 가능성이 존재한다. 모든 공유 메모리에서 데이터의 워드를 가지는 각 블럭을 구성하는 각 라인, L을 direct-mapped cache의 경우에 생각해보자. 캐쉬 일관성 특성을 만족하기 위해 캐쉬의 모든 라인에 있는 모든 블럭은 항상 일치해야 한다. 다시 말해 한 블럭이 일치 하지 않는다면 모든 시스템은 잘못 된 것이다. 따라서 line-block 추상화를 가능하게 한다. 예를 들어 이는 거대한 캐쉬를 모델링 할 때 캐쉬의 한 블럭을 모델링 하는 것만으로 충분하다 할 수 있다는 것이다. 다음으로 블럭에 포함된 데이타 그 자체로는 캐쉬 일관성 문제를 일으키지 않는다는 것이 쉽게 보여 질 수 있다. 그러므로 데이타 또한 추상화 될 수 있다. 데이타를 추상화함으로써 메모리 또한 추상화 될 수 있다. 예를 들어 메모리의 크기에 상관 없이 캐쉬 일관성 특성은 유효해야 한다. 시스템은 더 추상화 될 수 있다. 시스템의 완전한 행위를 모델링 하기 위해 추상화 모델에서 비결정성을 사용한다. 예를 들어 프로세서 캐쉬 시스템에서 프로세서는 요청을 하거나 안 할 수 있다. 요청이 있다면 이는 또한 read나 write를 할 수 있고 또한 hit나 miss할 수 있다. 그러므로 VIS에서 제공하는 비결정성을 사용하여 모델링한다. (그림 2)는 6.1에서 설명한 가정을 기반으로 한 추상화 모델이다. 하나의 메모리 워드만을 검증 대상으로 하기 때문에 그림에서 왼쪽의 노드를 홈 노드라고 하면, 나머지 세 개의 노드는 외부 노드1, 외부 노드2, 외부 노드3가 된다. 그러므로, 홈 노드의 PRO와 DIR을 나머지 세 개의 외부 노드에서는 RAC만을 구현하면 된다. 그리고 이들을 연결하는 Inter-Node Network(INN)을 구현한다.



(그림 2) 검증에 사용된 RACE 프로토콜의 추상화 모델

6.3 RACE 프로토콜의 명세

추상화 한 모델을 VIS를 이용하여 검증하기 위해서는 먼저 검증 할 수 있는 형태인 유한 상태 기계로 나타낸다. 유한 상태 기계로 나타낸 시스템을 VIS의 입력 언어인 Verilog로 구현 한다. Verilog 명세는 BLIF-MV로 변환되고 변환된 형식이 VIS로 읽혀진 후 현재 자동 정형 검증에서 가장 많이 쓰이는 두 가지 방법인 language containment와 모델 체킹(model checking)방법을 이용하여 검증한다. (그림 4)는 RAC의 외부 접근에 관한 상태 전이 중 한 경우인 RAC이 Invalid상태에서 coherent read가 발생한 경우 RAC을 Verilog로 명세한 일부분이다.

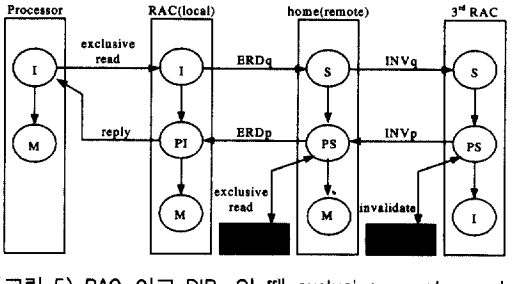


(그림 3) RAC$_i$이고DIR$_U$ 또는 DIR$_S$ 에서의 coherent remote read

```
module RAC
INVALID : begin
    if (pro_cmd == coh_read)
        begin
            rac_req = CRDq;
            rac_state = PI;
        end
    else if (pro_cmd == excl_read)
        begin
            rac_req = ERDq;
            rac_state = PIM;
        end
end
```

(그림 4) Verilog를 이용한 RAC 명세

(그림 6)은 DIR의 외부 접근에 관한 상태 전이 중 한 경우인 RAC이 invalid상태에서 exclusive read를 한 경우 DIR을 Verilog로 명세한 일부분이다.



(그림 5) RAC$_i$ 이고 DIR$_S$ 일 때 exclusive remote read

```
module DIR
SHARED : begin
    if (rac_req1 == ERDq && i_reply1 ==1 )
        begin
            if (s_reply2 == 1)
                begin
                 dir_req2 = INVq;
                 dir_state = PSM_ex;
            end
                else if (s_reply3 ==1)
                 begin
                 dir_req3 = INVq;
                 dir_state = PSM_ex;
                 end
        end
```

(그림 6) Verilog를 이용한 DIR 명세

6.4 VIS를 이용한 RACE프로토콜의 검증

5장에서 설명한 RACE프로토콜의 특성대로 정확하게 작동하는 지 검증한다. 다시 말해 RAC에서 발생할 수 있는 모든 요청이 의도한 대로 처리되는지를 검사한다. 예를 들어 (그림 7)의 CTL 명세 중에서 첫 번째는 RAC이 INVALID 상태에 있고, 버스에서 coherent read 요청이 들어 왔을 때, RAC에서 디렉토리로의 요청은 CRDq를 내보내고 모든 경우에 있어서 RAC의 상태가 SHARED가 된다는 것을 검증한 것이다. 두 번째는 RAC이 INVALID 상태에 있고, 버스에서 exclusive read 요청이 들어 왔을 때 RAC에서 디렉토리로의 요청은 ERDq를 내보내고 모든 경우에 있어서 RAC의 상태가 MODIFIED가 된다는 것을 검증한 것이다. 이 외에도 주어진 명세에 대하여 CTL을 이용하여 VIS로 검증한 결과 (그림 7)과 같이 RACE프로토콜이 필요한 특성을 만족함을 알 수 있었다.

또한 검증 과정 동안 특정한 상태에서 명세 상에 정의 되지 않은 메시지가 몇 가지 발견되었다. 예를 들어 홈 노드에서 coherent read 요청을 처리하고 있는 중간에 다른 노드로부터 writeback 요청이 들어 온 경우이다. 명세에서는 exclusive read 요청인 경우만 명세 되어 있는데 coherent read의 경우 역시 같은 처리가 필요하다는 것이다. 즉 점유 노드(1^{st} RAC)가 CRDq를 받았을 때 PM상태에 있으므로 홈 노드의 DIR에 NACK을 보낸다. 홈 노드의 DIR은 점유 노드로부터 NACK를 받기 전에 WRBq를 받으므로 메모리를 갱신하고 점유노드로 WRBp를 전달 하고 동시에 요청 노드에 CRDp를 보냄으로써 해결 할 수 있다. 이와 같은 경우는 명세자가 exclusive read 요청 중간에 write-back요청이 발생한 경우만을 명세하고 coherent read 요청이 발생하는 경우는 같은 방법으로 처리된다는 사실을 모호하게 기술함으로써 발생하는 경우라 하겠다. 설계자와 구현하는 사람이 동일 하거나 서로 간에 충분한 이해의 과정이 있는 경우는 큰 문제를 발생하지 않겠지만 그렇지 않은 경우는 문제를 유발 할 수 있다. 따라서 명세의 정확성 또한 요구 된다고 하겠다.

(그림 8)은 VIS에서 제공하는 invariant 검사를 이용한 검증 결과 이다. 여기서 명세한 특성은 이 시스템에서 절대 일어나지 않는 특성을 CTL로 명세하고 이

```
# MC : formula passed --- AG(rac1.rac_state = INVALID  * cmd1 = coh_read) -> AF(rac1.rac_state =SHARED))
# MC : formula passed --- AG(rac1.rac_state = INVALID  * cmd1 = excl_read -> AF(rac1.rac_state = MODIFIED))
# MC : formula passed --- AG((rac1.rac_state = SHARED  * cmd1 = excl_read) -> AF(rac1.rac_state = MODIFIED));
# MC : formula passed --- AG(rac1.rac_state = SHARED   * cmd1 = write_ba  -> AF(rac1.rac_state = INVALID));
# MC : formula passed --- AG(rac1.rac_state = MODIFIED * cmd1 = write_ba  -> AF(rac1.rac_state = INVALID));
```

(그림 7) VIS를 이용한 모델 체킹 결과

```
# INV: formula passed --- !(((rac1.rac_state=MODIFIED * rac2.rac_state=MODIFIED * rac3.rac_state = MODIFIED);
# INV: formula passed --- !((rac1.rac_state=MODIFIED  * (rac2.rac_state=SHARED  + rac3.rac_state = SHARED));
# INV: formula passed --- !(((rac1.rac_state=MODIFIED * rac2.rac_state=MODIFIED * rac3.rac_state = SHARED);
# INV: formula passed --- !(((rac1.rac_state=MODIFIED * rac2.rac_state=MODIFIED * rac3.rac_state = MODIFIED);
# INV: formula passed --- !((rac2.rac_state=MODIFIED  * (rac1.rac_state=SHARED  + rac3.rac_state = SHARED));
# INV: formula passed --- !(((rac2.rac_state=MODIFIED * rac3.rac_state=MODIFIED * rac1.rac_state = SHARED);
# INV: formula passed --- !(((rac1.rac_state=MODIFIED * rac2.rac_state=MODIFIED * rac3.rac_state = MODIFIED);
# INV: formula passed --- !((rac3.rac_state=MODIFIED  * (rac2.rac_state=SHARED  + rac1.rac_state = SHARED));
# INV: formula passed --- !(((rac1.rac_state=MODIFIED * rac3.rac_state=MODIFIED * rac2.rac_state = SHARED);
# INV: formula passed --- !((loc.loc_state=MODIFIED    * direc.dir_state=MODIFIED))
```

(그림 8) VIS를 이용한 invariant체킹 결과

것이 pass 되는지 확인함으로써 이 프로토콜이 명세한 내용에 대해 정확하다는 것을 검증 할 수 있었다. 예를 들어 RAC하나가 MODIFIED 상태에 있다면 다른 RAC 2개는 절대 SHARED 상태를 가질 수 없다. 이를 명세한 후 VIS를 이용하여 검증 해본 결과 이 프로토콜이 그러한 특성 들에 대하여 만족한다는 것을 알 수 있었다.

7. 결론 및 향후 과제

정형 기법은 정형 논리와 수학에 기반을 둔 방법으로 자연어가 내포하는 애매모호함과 불확실성을 배제할 수 있으며, 시스템이 어떤 특성을 만족하는지 검증하기 때문에 최소한 검증된 특성에 대해서는 완전히 믿을 수 있게 한다. 또한 시스템이 구현되기 전에 정확히 동작하는지를 검증 할 수 있으므로 시스템 개발 비용과 개발 시간의 측면에서 매우 효과적이다. 캐쉬 일관성 유지 프로토콜은 점점 복잡해지고 있는데 이는 지역적 접근을 위해 물리적인 메모리를 분산화 함으로써 공유 메모리를 논리적인 관점으로 프로그래머에게 제공하는 것이 어렵기 때문이다. 본 논문은 모델 체킹 기법을 사용하는 정형 검증 도구인 VIS를 이용하여 ETRI에서 개발한 캐쉬 일관성 프로토콜인 RACE 프로토콜을 검증함으로써 RACE 프로토콜이 정확하게 작동함을 확인 할 수 있었고 정형 기법의 적용 가능성을 제시하였다. 향후 과제로는 모델 체킹을 이용한 정형 검증 기법을 다양한 분야에 적용하는 것이 필요하다.

참 고 문 헌

[1] A. K. Nanda and L.N. Bhuyan, "A Formal Specification and Verification Technique for Cache Coherence Protocols," proc. of the 1992 International Conference on Parallel Processing, pp.I-22-I-26, 1992.

[2] D. E. Thomas, P.R. Moorby. "The Verilog Hardware Description Language," Kluwer Academic Publishers, Nowell, Massachusetts, 1991.

[3] E. M. Clarke and J. M. Wing, "Formal Methods : State of the Art and Future Directions" ACM Computing Surveys, pp.626-643, December 1996.

[4] F. Pong, "Symbolic State Model : A New Approach for the Verification of Cache Coherence Protocols," August 1995.

[5] F. Pong and M. Dubois, 1995. "A new approach for the verification of cache coherence protocols" IEEE Trans. Parallel Distrib. Syst. 6, 8 Aug, pp.773-787.

[6] F. Pong, and M. Dubois, "Formal verification of delayed consistency protocols," In Proceedings of the 10th International Parallel Processing Symposium Apr. IEEE Computer Society Press, Los Alamitos, Calif., pp.124-131, 1996.

[7] F. Pong and M. Dubois, "Formal Verification of Complex Coherence Protocols Using Symbolic State Models," Journal ACM on Computer Architecture, Vol.45, No.4, pp.557-587, July 1998.

[8] J. Archibald and J. L. Baer, "Cache Coherence Protocols : Evaluation Using a Multiprocessor Simulation Model, ACM Trans. on Computer Systems, Vol.4, No.4, pp.273-298, Nov. 1986.

[9] J. Archibald, "The Cache Coherence Problem in Shared-Memory Multiprocessors," PH.D Dissertation, University of Washington, Feb. 1987.

[10] J. Archibald, "The Cache Coherence Problem in Shared-Memory Multiprocessors," Ph.D Dissertation, University of Washington, Feb. 1987.

[11] James et al., "Scalable Coherent Interface," IEEE Computer, Vol.23, No.6, pp.71-82, June 1990.

[12] K. L. McMillan and J. Schwalbe, "Formal Verification of the Gigamax Cache Consistency Protocol," Proc. of the ISSM Int'l Conf. on Parallel and Distributed Computing, Oct. 1991.

[13] K. L. McMillan, "SYMBOLIC MODEL CHECKING," Kluwer Academic Publisher 1993.

[14] K. Hwang, "Advanced Computer Architecture," McGraw-Hill Book, 1996

[15] L. Barroso et al., "RPM : A Rapid Prototyping Engine for Multiprocessors," IEEE Computer, Feb. 1995.

[16] M Heinrich, The FLASH Protocol. Internal document, Stanford University FLASH Group, 1993.

[17] P. Loewenstein and , D. L. Dill, "Verification of a Multiprocessor Cache Protocol using Simulation Relations and Higher-Order Logic," Proc. of the 2nd Int'l Conf. on Computer-Aided Verification, Springer-Verlag, pp.302-311, June 1990.

[18] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," IEEE Transaction Computers, Vol.35, No.6, pp.677-691, Aug. 1986.
[19] R. K. Brayton et al. "HSIS : A BDD based system for formal verification." Proc. Of Design Automation Conference, 1994.
[20] T. Villa, G. Swamy, and T. Shiple. "VIS User's Manual"
[21] U. Stern and D. L. Dill, "Automatic Verification of the SCI Cache Coherence Protocol," IFIP WG10.5 Advanced Research Working Conference Proceedings, 1995.
[22] 기안도, 모상만, 김성운, 심규현, 심원세, 한우종, 한종석, "RACE Protocol : Remote Access Cache Coherency Enforcement Protocol" 컴퓨터 시스템 연구부, 컴퓨터 소프트웨어 기술 연구소, 한국전자통신연구원, TM-3100-1999-012, 1999.

엄 현 선

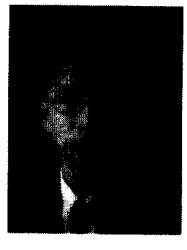
e-mail : hum@formal.korea.ac.kr
1993년 덕성여대 전산학과 입학
1998년 덕성여대 전산학과 졸업
1998년 고려대 컴퓨터학과 대학원 입학
2000년 고려대 컴퓨터학과 대학원 재학
관심분야 : 정형기법, CTL 모델체킹

최 진 영

e-mail : choi@formal.korea.ac.kr
1982년 서울대학교 컴퓨터공학과 졸업
1986년 Drexel University Dept. of Mathematics and Computer Science 석사
1993년 University of Pennsylvania Dept. of Computer and Information Science 박사
1993년~1996년 Research associate, University of Pennsylvania
1994년~현재 고려대학교 컴퓨터학과 부교수
관심분야 : 컴퓨터이론, 정형기법(정형명세, 정형검증), 실시간 시스템, 분산프로그래밍 언어, 소프트웨어 공학

한 우 종

e-mail : wjhan@etri.re.kr
1981년 고려대학교 전자공학과 학사
1984년 고려대학교 전자공학 석사
1995년 고려대학교 전자공학 박사
1985년 한국전자통신연구소 연구원
1986년~1988년 미국 AIT사 파견 연구원
1989년 전자계산기 분야 기술사
1997년~1988년 한국전자통신연구원 프로세서연구실장
1998년~현재 한국전자통신연구원 병렬시스템연구팀장 책임연구원
관심분야 : 컴퓨터구조, 마이크로프로세서 구조, 병렬처리 구조, 상호연결망, 계층메모리 구조 등



기 안 도

e-mail : adki@dynalith.com
1982년~1986년 한양대학교 전자공학과(공학사)
1986년~1988년 KAIST 전기 및 전자공학과(공학석사)
1994년~1997년 University of Manchester Dept. of CS (공학박사)
1988년~2000년 한국전자통신연구원
2000년~현재 Dynalith Systems, Inc의 R&D Director (책임연구원)
관심분야 : 컴퓨터구조, 병렬처리, 시스템레벨 모델링 및 검증

심 규 현

e-mail : khshim@etri.re.kr
1981년~1985년 서울대학교 전자공학과(공학사)
1989년~1991년 KAIST 전기 및 전자공학과(공학석사)
1991년~1997년 KAIST 전기 및 전자공학과(공학박사)
1985년~1989년 금성전기
1997년~현재 한국전자통신연구원 선임연구원
관심분야 : 시스템레벨 모델링 및 검증, ASIC 구현