링 연결구조 기반의 멀티코어 프로세서를 위한 캐시 일관성 유지 기법

(An Efficient Cache Coherence Protocol for Multi-Core Processors with Ring Interconnects)

박 진 영 † 최 린 ††
(Jinyoung Park) (Lynn Choi)

요 약 SOC 기술의 발전과 더불어 최근 여러 개의 프로세서를 단일 칩에 집적한 멀티코어 프로세서가 기존 슈퍼스칼라 프로세서 구조에 비하여 보다 에너지 효율적으로 성능을 증가시키는 방안으로 채택되고 있다. 이에 온 칩 프로세서간 캐시 일관성 유지 문제가 시스템의 안정성과 성능에 큰 영향을 미치는 요소로 부각되고 있다. 본 논문에서는 단 방향 링 연결구조의 노드 순서와 데이타 전달 순서를 이용하여 캐시 일관성 유지 요청의 순서를 결정하는 RING-DATA ORDER를 제안하여 기존 GREEDY-ORDER 방식의 단점인 재 요청을 최소화 하고 RING-ORDER의 단점인 토큰 관리의 부담을 없애면서 두 방식의 장점을 모두 가지는 캐시 일관성 유지 기법을 제안한다. RING-DATA ORDER는 기존의 공용 버스에 집중되는 일관성 유지 요청을 단 방향 링을 이용하여 각 노드에 골고루 배분 함으로써 유효 대역폭을 높이고 네이타 전송 순서에 기반하여 간단하게 처리 순서를 결정할 수 있으므로 멀티코어에 쉽게 적용 가능한 캐시 일관성 유지 기법이다.

키워드 : 링, 캐시, 일관성, 멀티코어

Abstract Today's microprocessor normally includes several processing cores to reduce the energy consumption without losing performance. In this paper, data transfer ordering mechanism can be efficiently used for cache coherence solution in unidirectional ring interconnect. RING-DATA ORDER combines the simplicity of GREEDY-ORDER and the performance of RING-ORDER. RING-DATA ORDER can be easily applicable to multi-core processor with unidirectional ring interconnect.

Key words : ring, cache, coherence, multicore, ring-data order

1. 서 론

SOC 기술이 빠르게 발전하면서 고성능 프로세서의 필요성이 더욱 커지고 있다. 기존의 SOC는 대부분 하나의 고성능 프로세서를 통해 동작하는데, 성능을 더욱 높이기 위해 동작 속도를 개선하는데 주력해 왔다. 그러나, 최근 동작 속도의 증가로 인한 소비 전력의 증가가 큰 문제로 대두되기 시작 하였으며, 이에 대한 대안으로 여러 개의 프로세서를 채택하여 성능은 유지하고 소비 전력은 절감하는 방안이 고성능 프로세서는 물론 내장형 프로세서 설계에서도 채택되기 시작하였다[1]. 이에 따라 캐시 일관성 유지 문제가 시스템의 안정성과 성능에 큰 영향을 미치는 요소로 부각 되고 있다. 또한, 종래의 공용 버스 기반의 온-칩 연결구조가 대역폭 향상을 위한 점대점(point-to-point) 구조로 전환되고 있다. 따라서, 이러한 새로운 온 칩 연결구조에 구현 가능한 멀티코어 프로세서에서의 캐시 일관성 프로토콜에 대한 지속적인 연구가 필요하다.

본 논문에서는 종래의 공용 버스가 가지는 문제점과 이를 극복하기 위해 단 방향 링 연결구조 기반의 멀티코어 프로세서에 적합한 캐시 일관성 유기 기법을 제시하고자 한다. 본 논문에서는 단 방향 링 연결구조에서 데이타 전송 순서에 따라 캐시 일관성 유지 요청의 순서가 결정되는 RING-DATA ORDER 기법을 제안하여, 종래 기술인 ORDERING-POINT 기법[2-5]의 긴 평균 지연시간 문제, GREEDY-ORDER 기법[6-9]의 캐시 일관성 유지 재 요청 문제, RING-ORDER 기법[10]의 토큰 관리 문제에 따른 성능 저하를 방지하고 GREEDY-ORDER와 RING-ORDER의 장점을 모두 가지도록 하였다.

본 논문의 내용은 다음과 같다. 2장에서는 종래의 멀티프로세서 구성과 그에 따른 캐시 일관성 유지 기법 등 관련 연구에 대하여 살펴보고, 3장에서는 단 방향 링 연결구조에 적용 가능한 RING-DATA ORDER와 그 특징을 기술하고 종래 연구 성과와 본 논문에서 제시한 방법의 복잡도에 대하여 분석하였다.

· 이 논문은 KCC2008에서 '링 연결구조 기반의 멀티코어 프로세서를 위한 캐시 일관성 유지 기법'의 제목으로 발표된 논문을 확장한 것임
· 이 논문은 2004 년도 한국학술진흥재단의 지원에 의하여 연구되었음 (KRF-2004-041-D00669)".

† 정 회 원 : 고려대학교 전자전기공학과
o2id@korea.ac.kr
†† 종신회원 : 고려대학교 전자전기공학과 교수
lchoi@korea.ac.kr

논문접수 : 2008년 8월 28일
심사완료 : 2008년 10월 20일

2. 관련 연구

일반적으로 멀티프로세서 시스템은 공용 버스를 기반으로 설계되어 모든 노드가 버스를 모니터하며 일관성 유지 문제를 해결하는 스누핑 프로토콜이 현재까지도 널리 사용되고 있다. 스누핑 프로토콜에서는 버스에 인가되는 모든 요청을 아비터(arbiter)가 우선 순위를 결정하기 때문에 원리는 간단한 반면, 노드의 개수가 많아질 경우 노드당 유효한 대역폭이 감소하고, 서로 버스를 점유하기 위한 경쟁을 하게 되어 성능 저하가 발생한다. 최근에는 공용 버스 대신 1개의 스누핑 처리 장치에 여러 개의 내장형 프로세서를 점대점으로 연결하여 트리 형태로 구성하여 2개 이상의 프로세서가 동시에 캐시 일관성 유지 요청을 할 수 있도록 하고 있으나, 스누핑 처리 장치에 모든 캐시 일관성 유지 요청이 집중되고, 동시에 여러 개의 요청이 처리될 수 있도록 하기 위하여 처리 장치가 복잡해지는 문제가 있다.

버스의 사용으로 인한 한계로 인해 스누핑 프로토콜의 성능이 제한되자, 버스뿐 아니라 다른 형태의 연결구조에서도 사용 가능한 디렉터리 프로토콜(directory-based protocol)이 제안 되었다. 디렉터리 프로토콜은 보다 고속의 연결구조에서도 사용가능 하지만, 단점으로 요청에 대한 지연시간이 버스 기반의 스누핑 프로토콜 보다 상대적으로 길며, 디렉터리 정보를 유지하기 위한 메모리가 필요하다. 대신 디렉터리 프로토콜은 다양한 형태의 연결구조에도 적용가능 하므로 스누핑 프로토콜에 비하여 상대적으로 확장성(scalability)이 뛰어나다. 그러나, 노드의 개수가 너무 많아지면, 메모리 블록보다 디렉터리 정보가 더 커질 수도 있는 문제가 있다.

디렉터리 프로토콜은 다양한 연결구조에 적용가능 한데, Barroso[3]는 풀맵 디렉터리 기반의 확장 가능한 칩 멀티프로세서 구조를 제안하였다. Jeffery A. Brown [11]은 4×4 메쉬 타일 기반의 Proximity-Aware 디렉터리 프로토콜을 제안 하였는데, 요청이 홈 노드에 도착하면 디렉터리 정보를 참고하여 요청한 노드와 가장 가까운 공유 노드에게 요청을 전달하고 디렉터리를 갱신하도록 하여 평균 지연시간을 개선하였다.

Milo M. K. Martin[12]은 토큰 일반성(Token Coherence) 기법을 제안하여 일관성 유지 기법을 정확성과 성능 정책으로 분리하였으며, 성능 정책으로 다중 브로드 캐스팅을 적용한 토큰B와 디렉터리와 유사한 구조를 가진 토큰D로 구성 하였으며 이 둘을 조합하여 토큰M을 제안하였다. 토큰 일관성 기법은 성능 정책에 따라 스누핑 프로토콜의 지연시간과 디렉터리 프로토콜의 대역폭 효율성을 연결구조에 구애 받지 않고 적용할 수 있도록 하였다. Michael R. Marty[10]는 토큰 일반성 기법을 기반으로 링 기반 칩 멀티프로세서에 사용 가능한 캐시 일관성 유지 기법을 제안 하였다. 이 밖에도 로컬 시스템과 글로벌 시스템으로 계층을 구분하여 서로 다른 연결 방식을 혼합 적용하기도 한다. 가령, 로컬 시스템은 버스 기반의 디렉터리 프로토콜을 사용하면서, 글로벌 시스템은 GREEDY-ORDER에 따른 링 연결구조를 적용하기도 한다[9].

3. RING-DATA ORDER 프로토콜

단 방향 링은 현재 노드로부터 다음 노드까지 단 방향의 데이타 및 요청을 전송 할 수 있는 점대점 링크로 구성된다. 단 방향 링은 그 구조 상 하나의 노드가 캐시 일관성 유지 요청을 보내면 링을 한 바퀴 돌아 자기 자신에게 도착하면 그 요청이 종료된다. 또한, 여러 개의 요청이 동시에 링을 통해 전송 가능하다.

3.1 단 방향 링 연결구조 기반의 RING-DATA ORDER

단 방향 링에서 링에 동일한 메모리 블록에 대한 요청이 동시에 발생할 경우에는 데이타 전송이 이루어 지는 순서대로 요청이 수행 완료 되도록 각각의 노드는 요청 보관 버퍼(request pending buffer)를 가지고 있으며 캐시 일관성 유지 기법은 MOESI 상태를 사용한다. 각 노드는 진행 중인 요청 목록(issued request table (IRT))을 가지고 있어서, 자기가 링에 보낸 요청 중에서 아직 수행 종료되지 않은 요청들의 정보를 가지고 있다.

각 노드는 링으로부터 도착하는 요청을 자신의 IRT와 비교하는데, 도착하는 요청의 요청한 노드 ID가 자기 자신의 노드 ID와 같으면 IRT에서 삭제하고 해당요청은 수행 완료된다. 도착하는 요청의 요청자 노드 ID와 자신의 노드 ID가 다를 경우 R_Type이 둘 다 Write이

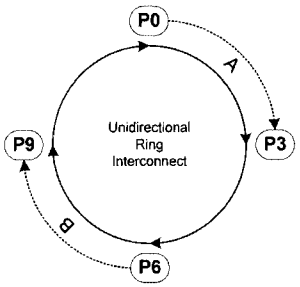

그림 1 단 방향 링 연결구조
(요청 A와 요청 B는 동시에 링에서 전송될 수 있으며, 각각의 요청은 링을 한 바퀴 되돌아 오면 수행 완료 된다.)

표 1 IRT(Issued Request Table)

R_Type	Address
...	...
R_Type	Address

고, 데이타 전송을 동반하면서(D=1), 메모리 블록의 주소가 동일 하면 링으로부터 도착한 요청을 펜딩 버퍼(pending buffer)에 기록하였다가 자신이 보낸 요청이 도착 할 때 자기의 Write 요청을 먼저 수행 완료 하고, 펜딩 버퍼에 기록된 요청을 갱신된 데이타와 함께 링에 내보내고 자신은 Invalid상태가 된다.

그림 2와 같이 노드 P0는 SHARED 상태, 노드 P6는 OWNED 상태일 때 노드 P3와 노드 P9이 동시에 Write 요청을 링에 내보낼 경우 노드 P0에 도착한 WriteReq;P9에 의해 노드 P0는 SHARED → INVALID로 상태 전환을 하고 WriteReq;P9를 노드 P3에게 전달한다. 또, 노드 P6에 도착한 WriteReq;P3에 의해 노드 P6는 OWNED → INVALID로 상태 전환을 하고 WriteReq;P3에 데이타를 덧붙여 P9에게 전달한다. 이후 노드 P3에 WriteReq;P9가 도착 하지만 데이타를 동반하지 않은 요청이므로 노드 P6에게 전달한다. 또한, 노드 P9에 WriteReq;P3가 도착하면 데이타가 있고 노드 P9의 IRT에 매치(match)되는 엔트리가 있으므로 WriteReq;P3를 펜딩 버퍼에 저장하고 IRT의 WriteReq;P9이 노드 P9에 도착하기를 기다린다. WriteReq;P9이 노드 P6을 지나 노드 P9에 도착하면, IRT와 매치되어 노드 P9은 데이타를 갱신하고, 갱신된 데이타를 펜딩 버퍼의 WriteReq;P3에 덧붙여 링에 전달한 후 노드 P9은 INVALID 상태가 된다. 이후 노드 P0를 거쳐 WriteReq;P3가 노드 P3에 도착하면 노드 P3는 데이타를 갱신하고 MODIFIED 상태가 된다.

그림 3과 같이 ReadReq;P3가 링에서 P6를 거치며 데이타를 가지고 있을 때 이 요청이 P9에 도착하면 P9은 자신의 IRT에 있는 WriteReq;P9와 주소가 같은 요청이므로 P9의 펜딩 버퍼에 ReadReq;P3를 저장하고, WriteReq;P9이 링을 한 바퀴 돌아 P9에 도착하면 P9은 데이타를 갱신하고, 펜딩 버퍼에 있는 ReadReq;P3와 갱신된 데이타를 링으로 내보내고, P9은 OWNED 상태가 된다. ReadReq;P3가 갱신된 데이타를 포함하여 P3에 도착하게 되면, P3는 SHARED 상태가 된다.

캐시 일관성을 유지하기 위해서는 싱글 Writer 조건을 만족하여야 하는데, 모든 Write 요청은 링의 모든 노드를 INVALID로 만들고 MODIFIED나 EXCLUSIVE 또는 OWNED 상태인 단 하나의 노드로 부터 데이타를 전송하게 된다. 이때 데이타를 전송중인 Read요청이 있으면 Write요청이 있는 노드에 펜딩 되거나, Read요청이 수행 완료된 후 Write요청에 의해 INVALID 상태로 전환되고, 데이타를 전송중인 Write요청은 경쟁 노드에 펜딩되거나 자기 노드에 도착하여 수행 완료되므로 RING-DATA ORDER기법은 싱글 Writer 조건을 만족한다.

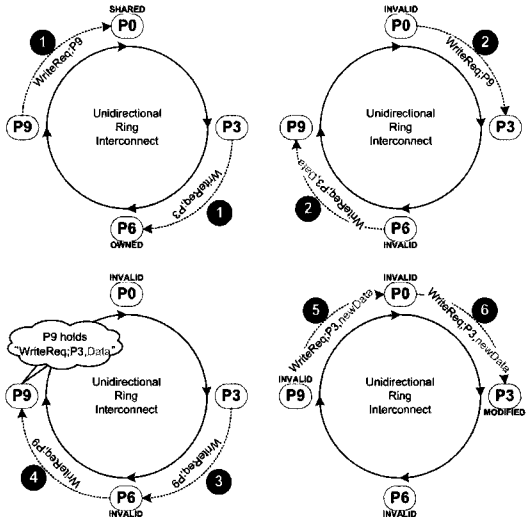

그림 2 단 방향 RING-DATA ORDER 기법 예1

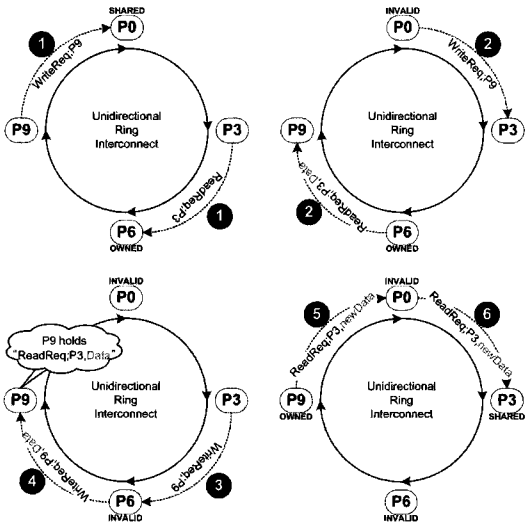

그림 3 단 방향 RING-DATA ORDER 기법 예2

3.2 단 방향 링 기반 캐시 일관성 유지의 복잡도 및 성능 비교

종래의 단 방향 링 기반 캐시 일관성 유지 방법으로 ORDERING-POINT[2-5]와, GREEDY-ORDER방식[6-9], RING-ORDER[10] 방식이 있다.

링에 존재하는 전체 노드가 N개일 경우, ORDERING-POINT 기법의 경우 링 상의 특정 위치에 삽입된 ORDERING-POINT까지 각 노드의 요청이 도달하기 위한 평균 거리는 N/2이다. ORDERING-POINT에 도달한 요청은 ORDERING-POINT를 시작으로 링을 한 바퀴 돌아 다시 ORDERING-POINT에 도착하여 각

노드에 요청을 전달하게 되며, 이때 N개의 노드를 거치게 된다. 만약 ORDERING-POINT가 각 노드의 요청이 완료 되었다는 확인 메시지(Acknoledgement)를 보내게 되면 이때 평균적으로 N/2 만큼의 노드를 거치게 된다. 따라서, ORDERING-POINT 방법의 경우 캐시 일관성 유지를 위해 N/2 + N + N/2 = 2N 만큼의 노드를 거치게 되어 평균적으로 링을 두 바퀴 돌아야 한다. ORDERING-POINT의 삽입은 간단한 반면 평균 지연시간이 긴 단점이 있다.

GREEDY-ORDER방법은 링에서 경쟁 관계에 있는 요청들 중 데이타를 먼저 획득한 요청이 링을 한 바퀴 돌아 수행되면, 경쟁 관계에 있는 다른 요청은 데이타 획득에 실패할 경우 데이타 없이 링을 한 바퀴 돌게 되어 요청이 종료되지 못하고 해당 노드는 링에 다시 요청을 내 보내어 데이타를 획득할 때 까지 재 요청하게 된다.

표 2 성능 비교, N = 노드 개수
(성능 : 값이 작을수록 좋음)

종 류	성능	장점	단점
ORDERING-POINT	2N	간단한 구조	긴 지연시간
GREEDY-ORDER	N(1+M)	대역폭 효율	재 요청 (M)
RING-ORDER	N	높은 대역폭 효율	토큰 관리 필요
RING-DATA ORDER	N	높은 대역폭 효율	충분한 버퍼 필요

RING-ORDER의 경우 링에서 특수 토큰을 먼저 획득한 노드가 해당 요청에 대하여 필요한 토큰을 수집하면서 경쟁 관계에 있는 요청을 현재 노드에서 멈추게 하고, 특수 토큰을 획득한 노드가 필요한 만큼의 토큰을 수집하면 요청을 완료하고 경쟁 노드에게 토큰을 넘겨주는 방식으로, 모든 요청은 링을 한 바퀴만 돌면 되므로 거쳐 가는 노드의 개수는 N으로 일정하다. 그러나, 각 요청은 특수 토큰을 가진 노드에서 멈추게 되므로 지연시간은 더 커질 수 있으나, 모든 요청은 궁극적으로 링을 한 바퀴만 돌면 되므로 링의 대역폭 효율성은 높다. RING-ORDER는 토큰 일반성 기법에 기반을 두고 있어서, 메모리 블록에 대하여 고정된 개수의 토큰을 항상 유지 하여야 하는데, 만일 특정 노드에서 캐시 블록 교체 등에 의하여 메모리 블록이 쫓겨날 경우 이 메모리 블록에 대한 모든 토큰을 다른 노드로 전송해야 하는 오버헤드가 발생한다.

본 논문에서 제안하는 단 방향 링에서의 캐시 일관성 유지 기법 (RING-DATA ORDER)은 GREEDY-ORDER와 RING-ORDER의 장점을 결합한 것으로 GREEDY-ORDER의 단점인 재 요청을 방지하면서 RING-ORDER의 단점인 토큰 유지의 부담을 제거하여, GREEDY-ORDER기반의 링 구조로 RING-ORDER의 성능을 내도록 하였다. RING-DATA ORDER에서는 모든 요청은 링을 한 바퀴만 돌면 된다. 또, 캐시 리플레이스등에 의하여 메모리 블록이 쫓겨날 경우 RING-ORDER처럼 토큰을 유지하기 위한 추가 요청이 발생하지 않는 장점이 있다.

이와 같이 RING-DATA ORDER 기법에서는 링에 동시에 존재하는 일관성 유지 요청들의 순서를 노드의 순서와 데이타 전송여부에 따라 능동적으로 처리 할 수 있으며, 스누핑 프로토콜에서 사용되는 일관성 유지 장치 사용에 따른 요청의 집중 현상을 링에 골고루 분산할 수 있고, 캐시 태그를 2중으로 복제하지 않아도 되는 장점이 있다. 반면에 평균 지연시간이 긴 점이 단점이나, 링 연결구조의 대역폭 효율성이 더 높으므로 어느 정도 상쇄 될 수 있다. 또, 패킷 스위치나 라우터를 사용하지 않음으로써 보다 간단하게 구현 가능한 방법이다.

4. 실험 방법 및 결과

RING-ORDER와 RING-DATA ORDER 기법의 성능을 비교하기 위해 SimpleScalar 3.0d을 이용하여 SPEC CINT 2000 Alpha binaries를 수행할 때 생성되는 캐시 접근 상태를 각 벤치마크 별로 50억 개의 Instruction을 수행하여 프로파일로 저장하였으며, 링 연결구조 기반의 멀티 코어 시뮬레이터를 개발하여 RING-ORDER와 RING-DATA ORDER에 대하여 실험을 진행 하였다. ORDERING-POINT와 GREEDY-ORDER는 선행연구[10]에서 성능이 나쁨이 증명되었으므로 별도로 실험하지 않았다.

그림 4는 링의 노드 개수를 2개부터 32개까지 확장해 가면서 RING-ORDER 기법과 비교하여 RING-DATA ORDER의 평균 지연 시간을 비교한 것으로, 두 방식의 차이가 거의 없음을 알 수 있다.

그림 5는 RING-ORDER와 RING-DATA ORDER 방식에서 필요로 하는 L1캐시의 추가 용량을 나타낸 것으로 링의 노드 개수가 늘어날수록 RING-DATA ORDER가 더 효율적임을 알 수 있다.

5. 결 론

단 방향 링 연결구조 기반의 캐시 일관성 유지 방법 (RING-DATA ORDER)은 링에 동시에 존재하는 요청의 처리 순서를 각 노드의 위치와 데이타 동반 여부에 따라 처리함으로써 시스템의 캐시 일관성을 유지 하도록 하였다. 단 방향의 링 연결구조를 사용함으로써 각

표 3 프로세서 노드의 구성

항목	설명
L1 데이타 캐시	128 KB direct mapped cache 24-bit tag, 32 bytes per block 12-bit index, 2-bit offset, 64-bit data
주소	40-bit
IRT	1 issued request table entry
버퍼	Point to point incoming buffer Default 32 entries (configurable) Entry 0 used for local requests Circular queue
펜딩버퍼	Pending buffer for blocking coherent requests which contains data transfer Default 32 entries (configurable) Circular queue
프로토콜	MOESI protocol used

표 4 시스템의 구성

항목	설명
L2 캐시	8 MB direct mapped cache 24-bit tag, 32 bytes per block, 18-bit index Perfect L2 cache model L2 access hit always L2 provides the data immediately
P2P Links	Default 1 P2P link between cores Link size is configurable

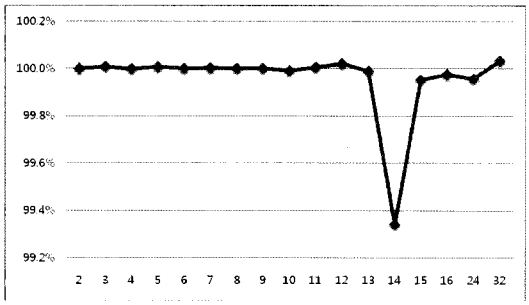

그림 4 평균 지연시간(Normalized by RING-ORDER)

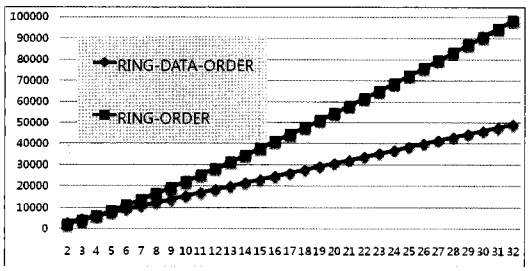

그림 5 L1캐시 추가 필요 용량(bytes)

노드는 점대점 연결을 하게 되어 고속의 동작이 가능하며, 펜딩 버퍼를 이용하여 경쟁 관계의 요청들에 대한 순서를 결정함으로써 RING-ORDER의 장점과 GREEDY-ORDER의 장점을 모두 가지게 된다. 또한, 패킷 스위치 등의 복잡한 장치 없이도 노드간의 평균 유효대역폭을 효과적으로 높일 수 있다.

참 고 문 헌

[1] ARM. ARM11 MPCore Processor Technical Reference Manual (r1p0), page 1-3, page 7-2, 2008.

[2] C. N. Keltcher, K. J. McGrath, A. Ahmed, and P. Conway. The AMD Opteron Processor for Multiprocessor Servers. IEEE Micro, 23(2):66-76, March-April 2003.

[3] L. A. Barroso and M. Dubois. Cache Coherence on a Slotted Ring. In Proceedings of the International Conference on Parallel Processing, pages 230-237, Aug. 1991.

[4] K. Gharachorloo, M. Sharma, S. Steely, and S. V. Doren. Architecture and Design of AlphaServer GS320. In Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems, pages 13-24, Nov. 2000.

[5] D. Gustavson. The Scalable Coherent Interface and related standards projects. IEEE Micro, 12(1): 10-22, Feb. 1992.

[6] L. A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese. Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing. In Proceedings of the 27th Annual International Symposium on Computer Architecture, pages 282-293, June 2000.

[7] S. Kunkel. IBM Future Processor Performance, Server Group. Personal Communication, 2006.

[8] B. Sinharoy, R. Kalla, J. Tendler, R. Eickemeyer, and J. Joyner. Power5 System Microarchitecture. IBM Journal of Research and Development, 49(4), 2005.

[9] S. W. Chung, S. T. Jhang, and C. S. Jhon. PANDA: ring-based multiprocessor system using new snooping protocol. In International Conference on Parallel and Distributed Systems, pages 10-17, 1998.

[10] Michael R. Marty and Mark D. Hill, Coherence Ordering for Ring-based Chip Multiprocessors, International Symposium on Microarchitecture, 2006.

[11] Jeffery A. Brown, Proximity-Aware Directory-based Coherence for Multi-core Processor Architectures, Symposium on Parallelism in Algorithms and Architecture (SPAA), 2007.

[12] Milo M.K. Martin, Token Coherence, PhD Thesis, University of Wisconsin-Madison, 2003.