다중코어 시스템의 메쉬구조 상호연결망이 성능에 미치는 영향

김한이, 김영환, 서태원
고려대학교 컴퓨터교육과
e-mail : hanyeemy@korea.ac.kr, tsohr@korea.ac.kr, suhtw@korea.ac.kr

The Effect of Mesh Interconnection Network on the Performance of Manycore System.

Young-Hwan Kim, Han-Yee Kim, and Taeweon Suh
Dept. of Computer Science Education, Korea University

요 약

다중코어(Many-Core) 시스템은 많은 코어들이 상호연결망을 통해서 연결되어있는 시스템으로, 단일코어나 멀티코어 시스템에 비해 보다 많은 병렬 컴퓨팅 자원을 지원한다. Amdahl 의 법칙에 의하면 병렬화되어 처리하는 부분은 이론적으로 프로세서의 개수에 비례하게 가속화 될 수 있지만, 상호연결망에서의 전송 지연을 비롯한 많은 요인에 의해서 성능의 가속화가 저해된다. 특히 캐시 일관성 규약(Cache Coherence Protocol)을 지원하는 대부분의 다중코어 시스템에서는 병렬화를 함에 있어서 캐시 미스로 인해 발생하는 데이터의 전송 지연이 성능에 많은 영향을 미칠 수 있다. 따라서 효과적인 병렬 프로그램을 위해서는 캐시 구조에 대한 이해를 바탕으로 상호연결망에 대한 연구가 필요하다. 본 논문에서는 메쉬(Mesh) 구조의 64 코어 다중코어 시스템인 TilePro64 를 이용하여 상호연결망의 데이터 전송 지연에 따른 프로그램 성능의 민감도를 측정하였다. 결과적으로 코어간 거리(Hop)가 늘어날수록 작업의 수행시간이 평균적으로 4.27%씩 선형적으로 증가하는 관계가 있는 것으로 나타났다.

1. 서론

컴퓨터 구조는 시대 별로 활발히 진행되었던 연구 분야가 확연히 구분된다. 1970 년대부터 1980 년대에는 컴파일러와 관련하여 ISA(Instruction Set Architecture)의 개발에 힘썼다면, 1990 년대에서는 파이프라인 구조에서 조건 분기 명령어의 실행에 따른 지연을 최소화하기 위한 아키텍처 개발에 힘썼다. 2000 년대에서부터 현재까지는 작업의 병렬화를 위한 멀티코어, 다중코어의 개발에 힘쓰고 있으며, 코어간 통신이 점차 중요해짐에 따라 보다 진보된 상호연결망을 개발하기 위해 많은 연구가 이루어지고 있다.

본 논문에서는 다중코어 시스템에서의 상호연결망 구조와 캐시 일관성 규약이 성능에 어느 정도 영향을 미치는지에 대해 TilePro64 프로세서를 대상으로 논의를 전개할 것이다. 이를 위해서 TilePro64 가 다중코어 프로세서 구조로서 가지는 특징과 어떠한 상호연결망을 적용했는지 알아보고, 직접 작성한 벤치마크 프로그램을 이용하여 상호연결망의 데이터 전송 지연이 프로그램 실행 성능에 어떻게 영향을 미치는지의 민감도에 대해 5 가지 지표를 선정하여 통계적으로 분석하였다.

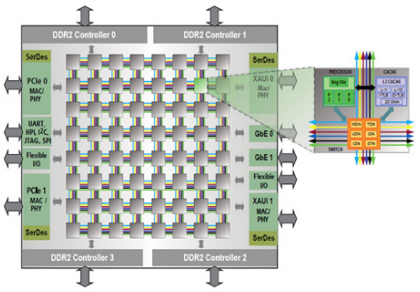
• 이 논문은 2011 년도 정부(교육과학기술부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행된 것임(2011-0003422)

2. 관련연구

다중코어 시스템을 연구하는 경우 상호연결망에 따른 성능 민감도는 성능실험의 일부로서 이미 많이 연구되고 있다. 참고문헌[1]에서 Godson-T 구조는 TilePro64 와 매우 유사한 상호연결망을 가진 다중코어 시스템을 설계했다. 이 시스템을 평가할 때 시뮬레이터를 통해서 다른 데이터 크기와 쓰레드 개수를 가지고 실험하는 FFT 벤치마크 코드를 사용하여 성능을 실험하였다. 참고문헌[2]역시 IBM Cyclops64 라는 새로운 구조의 설계를 평가하기 위한 실험의 일환으로 시뮬레이터를 사용하였다. 이외의 많은 논문들에서 주로 시뮬레이터를 이용하여 쓰레드 개수나 데이터 크기를 다양하게 하여 타깃 소스코드를 수행시켜보는 방식으로 진행하였다. 참고문헌[3] (SPLASH2)은 성능평가의 표준으로서 인정되는 논문으로서 공유 메모리 구조를 사용하는 멀티프로세서 시스템을 평가할 수 있도록 작성되었다. 이 실험에서 기준이 된 테스트 플랫폼은 Tango-Lite 라는 시뮬레이터인데, 이 시뮬레이터는 하드웨어의 구조적 변수를 조정하는 방식으로 캐시 크기나 구조 혹은 쓰레드 개수를 달리 하여 성능을 비교하는 연구를 진행하였다. 본 논문에서는 조사했던 위의 연구와는 달리 실제 존재하는 하드웨어 자원을 통해서 현실적인 성능 평가 자료를 작성하고, 미시적으로 코어간의 상호연결망에서의 전송 지연과 성능 민감도를 측정하고 분석한다.

3. TilePro64 의 구조 및 특징

TilePro64 프로세서는 Tilera 사에서 출시한 64 코어 시스템으로 32 bit 연산을 하며, 64 개의 일반목적 레지스터를 제공한다. 이 코어들은 가로와 세로가 8×8 인 격자 구조로 상호연결망을 구성하고 있다.



(그림 1) TilePro64 구조

(그림 1) 과 같이 각 코어는 스위치 엔진, 캐시 엔진, 그리고 연산을 하는 프로세서 엔진으로 구성된다. TilePro64 의 프로세서 엔진은 최대 3 개의 명령어를 동시에 처리할 수 있는 VLIW(Very Long Instruction Words) 구조를 가지며, 항상 순차적으로 처리한다.

3.1 TilePro64 의 캐시구조

TilePro64 의 각 코어가 사용하는 캐시 엔진의 주요 사양은 <표 1>과 같다.

캐시의 계층 및 종류	사양
L1 명령어 캐시	64B Line size 16KB Direct-mapped
L1 데이터 캐시	16B Line size 8KB 2-way Associative Write through 정책
L2 캐시	64B Line size 64KB 4-way Associative Write-back 정책

<표 1> 캐시 엔진의 주요 사양

3.2 TilePro64 의 상호연결망 과 캐시 일관성 규약

캐시 일관성 규약으로서 디렉토리 기반 프로토콜이 사용되는데 이 규약이 동작하기 위해서 코어와 코어 사이, 코어와 메모리 컨트롤러 사이의 상호연결망을 사용한다. 라우팅 방식은 데이터를 포함하는 근처의 코어로 접근하거나, 근처 코어에 데이터가 없을 경우 데이터가 저장된 메모리 컨트롤러 방향의 스위치 엔진으로 접근한다. 한 사이클에 한 칸씩 32 bit 패킷 단위로 인접한 스위치 엔진으로 전달하며 전송 중에는 3-credit 웜홀 라우팅[1] 방식을 사용하여 한 스위치 엔진에서는 동시에 최대 3 개의 서로 다른 패킷을 전달할 수 있다. 캐시 작업에 소요되는 사이클 수를 Tilera 사 측에서 문서로 정리해서 다음과 같이 <표 2>처럼 밝히고 있다.

[1] 웜홀 라우팅: 패킷의 전송 과정이 완전히 끝날 때까지 전송 경로를 배타적으로 점유하는 방식.

구분	소요 사이클
L1 hit	2
L1 miss, L2 hit	8
L1/L2 miss, *Adjacent DDC*[2] hit	35
L1/L2 miss, DDR2 page hit	69
L1/L2 miss, DDR2 page miss	88

<표 2> TilePro64 파이프라인 대기시간

<표 2>와 같이 L1 hit 이나 L2 hit 인 경우 코어 내의 캐시에서 데이터를 빠르게(2~8 Cycle 소요) 로드할 수 있지만, 캐시미스의 경우 데이터를 메인 메모리에서 가져오고, 가져온 후에도 인접한 코어의 스위치 엔진을 거치기 때문에 상당한 지연(69~88 Cycle 소요)이 발생한다. TilePro64 에서는 이러한 전송 지연을 개선하기 위해 인접 코어의 캐시를 사용하는 *Adjacent DDC* 라는 기술로서 디렉터리 기반 캐시 일관성 프로토콜을 구현했다.

4. TilePro64 에서 홉에 따른 민감도 실험 및 논의

원격 데이터 캐시 히트가 35 cycle 이라고 하지만 이 값은 평균적인 값으로서 실제로는 거리에 따른 전송 지연 시간의 편차가 존재한다. 따라서 코어간의 홉 차이가 다르다면, 같은 프로그램을 수행하더라도 원격 데이터 캐시작업의 대기시간은 다르게 나타날 것이다. 본 실험에서는 홉 차이에 따른 수행시간과 민감도를 성능 카운터를 통해서 분석해 보고 결과에 대해서 논의해 볼 것이다.

4.1 실험환경

호스트 시스템	
CPU 구조	인텔 Nehalem Core i7 950 (3.07GHz)
메모리	6GB (DDR3 2GB ×3 개)
메인보드	ASUS P6T SE
운영체제	Cent OS 5.6 (2.6.18-238.12.1.el5)
PCI-e 16x 로 연결된 다중코어 시스템	
CPU 구조	TilePro64, 700MHz × 64 코어
메모리	4GB (DDR2 1G × 4 개)
운영체제	Linux 2.6.36.4-MDE-3.0.1.125620

<표 3> 기본 실험 환경

기본적인 실험 환경은 위의 <표 3>과 같다. TilePro64 는 Host 시스템과 PCI 를 통해서 연결 되어 있다. 대상 하드웨어 자원의 조작을 위해 Tilera 사에서는 별도로 *Tilera Multicore Development Environment 3.0* ™ 을 제공한다. MDE 는 전체 실행 환경을 아우르는 말로써 하이퍼바이저, 슈퍼바이저, 운영체제의 소프트웨어 스택을 이룬다. 하지만 64 개의 코어를 모두 사용할 수 있는 것은 아니며, PCI 통신과 I/O 컨트롤

[2] DDC(Dynamic Distributed Cache): 다른 코어의 캐시 엔진에 데이터가 존재할 때, 해당 코어에서 데이터를 로드 하도록 하는 기능으로 L3 캐시와 유사.

러 제어 등을 위해서 7 개의 CPU 를 하이퍼바이저가 독점적으로 사용한다. 따라서 리눅스에서 사용할 수 있는 코어는 총 57 개로 한정된다. MDE 에서는 tile-gprof 라는 oprofile 기반 분석 툴을 제공하며 이 도구를 통해 성능 카운터를 측정할 수 있다.

4.2 실험 방법 및 주요 코드

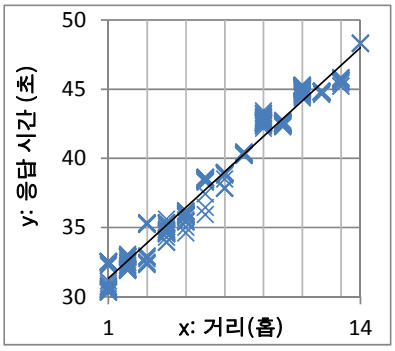
두 코어간의 홉 차이에 따른 전송 지연을 알아보기 위해서 먼저 거리를 기준으로 테스트 순서쌍(홈 쓰레드의 위치와 원격 쓰레드의 위치)을 수집한다. 홉은 1 부터 14 까지 존재한다. 또한 데이터를 외부로부터 읽고 써야 하므로 L2 캐시미스가 발생하여야 한다. 단, 상호연결망에서의 홉의 차이에 따른 민감도를 알아보는 실험이므로 메인 메모리에 접근하는 원격 데이터 캐시미스는 최소화한다. 이를 위해 L1 캐시의 크기인 8KB 이하 크기의 공유 데이터(카운터, 잠금, 차례 등)를 지정하고, 서로 다른 코어에서 번갈아 가면서 읽기/쓰기 작업을 하도록 한다. 문제의 복잡도를 줄이기 위해 별도로 제공된 메모리 할당 API 를 사용하여 공유변수의 소유자를 명확하게 지정한다. 이 API 는 TLB 의 속성값을 조작하여 특정 메모리 주소를 하나의 타일에서 관리(소유)할 수 있도록 한다. 실험 대상으로 서로 다른 코어에 있는 쓰레드가 번갈아 가면서 하나의 공유변수를 0 부터 정해진 값까지 1 씩 증가시키도록 하였다. 기능적 정확성을 유지하기 위해서 잠금을 사용하여 두 쓰레드의 쓰기 순서를 지정하였다. 시스템 함수로 제공되는 pthread_mutex 는 잠금에 얼마나 접근했는지 알 수 없기 때문에, 대신 test-and-set 명령을 사용하여 잠금을 구현하여 접근횟수를 측정하도록 하였다. 수행중의 비효율을 줄이고 하나의 명령 번들에 가능한 한 많은 명령을 포함시키기 위해 최대한의 최적화 정도(O3)를 사용하였다. 또한 거리당 측정할 수 있는 경우의 수를 줄이기 위해 거리당 최대 30 개의 테스트 데이터를 수집하도록 하였으며 목표 카운터는 100,000,000 으로 하였다.

4.3 측정 요소

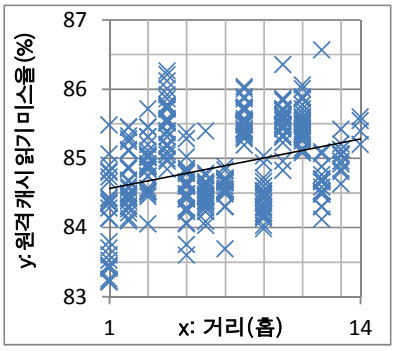
- **수행 시간**: 가장 기본적인 부하 측정의 지표로서 사용하고 있으며, 측정 시점은 두 쓰레드가 작업 준비를 마치고 특정 함수의 작업을 실행하기 직전과 직후를 측정한다.
- **원격 데이터 캐시 읽기 미스율**: 전송 과정에서 거리가 멀수록 스톨 시간이 늘어나므로 전송 시간 동안 수신자는 캐시 읽기 작업을 계속할 수 있으므로 원격 데이터 캐시 읽기 미스율이 낮아질 것으로 예상했다.
- **원격 데이터 캐시 쓰기 미스율**: 원격 데이터 쓰기 캐시 미스율(메인 메모리 쓰기 접근율)은 작성한 프로그램이 제대로 수행되었는지를 알아보기 위한 지표로 사용할 수 있다.
- **데이터 캐시 스톨 비율, 캐시 부하로 인한 스톨 비율**: 수행 명령 묶음을 기준으로 데이터 캐시 작업으로 인해 지체된 명령 묶음의 수의 비율과 캐시 부하로 인해 지체된 명령 묶음의 비율을 측정하였다. 거리가 가까우면 캐시에 접근하는 시간과 빈도가 늘어나 데이터 캐시에서 수행이 스톨되는 명령 묶음이 많이 관찰될 것으로 예상했다.

4.4 실험 결과

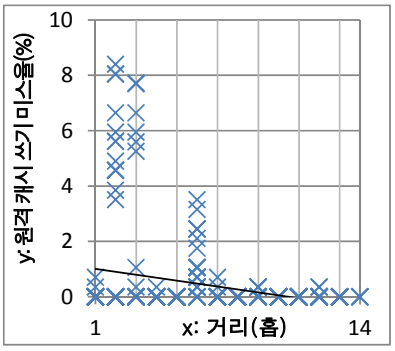
아래 5 개의 차트를 참조한다.



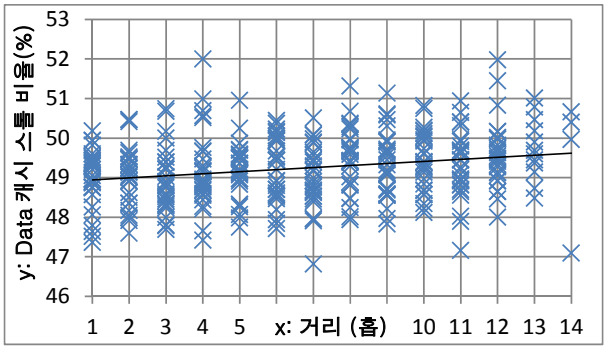
<표 4> 거리에 따른 응답 시간



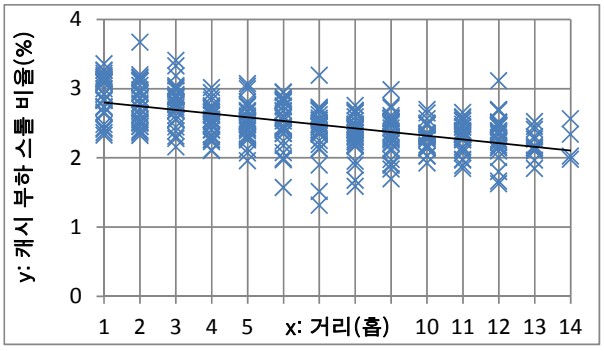
<표 5> 거리에 따른 원격 캐시 읽기 미스율



<표 6> 거리에 따른 원격 캐시 쓰기 미스율



<표 7> 거리에 따른 데이터 캐시 스톨 비율



<표 8> 거리에 따른 캐시 부하 스톨 비율

4.5 실험 결과 분석 및 논의

<표 4>에 따르면 홉 차이의 증가에 따라 작업의 완료 시간이 선형적으로 증가하는 관계가 있다. 단순 선형회귀 분석을 사용하였을 때 $y = ax + b$의 모델에서 기울기 $a = 1.2835$ 인 뚜렷한 선형적 관계가 도출되는 특징을 보인다. 이것을 비율로 환산하면, 평균한 홉이 증가할 때 마다 4.27%의 성능 저하를 예상할 수 있다.

실험결과 <표 6>에서 알 수 있는 것은, 몇몇 예외적인 경우가 존재하지만 대부분 0에 근접하는 것을 보아 실험의 의도대로 메모리 컨트롤러에 직접적으로 접근하지 않았음을 확인할 수 있다.

실험결과 <표 5>에서 보여준 원격 캐시 읽기 미스율은 약 83%~87% 정도로, 일반적인 응용프로그램의 캐시 읽기 미스율이 0~5% 정도인 데에 비하면 상당한 수준이다. 이것은 두 프로세스가 공유 데이터를 번갈아 가면서 수정하도록 작성했기 때문인데, 보다 구체적으로는 자신의 수행을 대기하는 동안에는 임계영역에 진입하기 위한 조건들을 확인하는 작업을 반복적으로 하므로 캐시히트가 일어나지만, 반면에 한쪽의 프로세서에서 임계 영역 내의 공유변수 수정 작업을 하면 다른 쪽의 프로세서에서는 캐시 무효화 신호를 받게 되면서 캐시미스가 발생한다. 타이밍이 아주 이상적인 경우 모든 load-word 와 test-and-set 명령에서 캐시미스가 일어나야 하지만, 실험 환경이 운영체제 위에서 동작하기 때문에 CPU 를 독점적으로 사용하지 못한다는 점과, 임계영역의 진입 조건에 소요되는 사이클과 임계영역의 실행에 소요되는 사이클 사이의 균형을 맞추어야 한다는 점에서 지표가 이상적으로 (100%의 원격 캐시 미스율) 나오기는 어렵다.

실험결과 <표 5>, <표 7>, <표 8>는 홉 차이 마다 결과값의 오차 폭이 추세선의 기울기를 크게 상회하여 홉 차이에 따른 민감도가 적다고 할 수 있다. 하지만 원격 캐시 작업(DDC)에서 두 코어간 거리가 멀경우 요청한 코어에서 소유자 코어까지 전달 지연 시간이 늘어난다는 관점에서 데이터 캐시 미스에 따른 지연(<표 7>)에서 추세선의 기울기가 약간의 양의 값($a = 0.0004$)을 나온 것은 어느 정도 긍정적인 실험결과라 할 수 있다. 이와 같은 맥락으로 <표 8> 역시 가까운 거리의 코어의 경우 단위시간당 데이터의 교환이 더욱 빈번하게 일어나기 때문에 거리에 따라 캐시부하가 줄어드는 관계($a = -0.0005$) 역시 긍정적인 실험결과라 할 수 있다.

5. 결론 및 향후 연구 방향

본 논문에서는 실제 존재하는 다중코어 시스템인 TilePro64 를 이용하여 코어간의 거리 차이에 따른 성능의 민감도를 측정하였다. 결과적으로 코어간 거리가 늘어날수록 작업의 수행시간이 평균적으로 4.27% 씩 선형적으로 증가하는 관계가 있는 것으로 나타났다. 하지만 이 실험은 Linux 환경에 기반했기 때문에 문맥교환으로 인해 두 가지 문제가 생기는데, 하나는 시간적 오차가 발생한다는 점이고, 다른 하나는 의도치 않게 다른 작업이 캐시를 사용하게 되므로 작업중인 캐시의 유효성을 확신할 수 없게 된다는 점이다. 따라서 운영체제가 개입하지 않는 환경에서 실험을 진행하면 보다 집약적이고 신뢰할만한 결과를 얻을 수 있을 것으로 기대된다. 또한 이 논문에서 직접 작성한 실험코드가 아닌 표준으로 인정받는 SPLASH2 (참고문헌[3])의 실험 코드의 결과와 대조해 보는 것도 좋은 연구가 될 것이다. 본 연구의 내용을 바탕으로 위의 문제를 개선하여 다양한 연구를 진행해 나갈 수 있을 것이라 생각되며, 특히 다중코어에서의 최적화를 위해서 고려해야 되는 전송 지연 및 기타 하드웨어에 자원들의 특성을 파악하면 타깃 하드웨어에 최적화된 병렬 프로그래밍 모델이나 통신 패턴을 개발하는데 도움이 될 것이라 생각된다.

참고문헌

[1] Xu Wang, Ge Gan, Joseph Manzano, Dongrui Fan and Shuxu Guo, "A Quantitative Study of the On-Chip Network and Memory Hierarchy Design for Many-Core Processor," 14th IEEE International Conference on Parallel and Distributed Systems, 2008

[2] Ying Ping Zhang, Taikyeong Jeong, Fei Chen, Haiping Wu, Ronny Nitzsche, Guang R. Gao, "A Study of the On-Chip Interconnection Network for the IBM Cyclops64 Multi-Core Architecture" IEEE 20th International parallel and Distributed processing Symposium (IPDPS06), Apr 25, 2006.

[3] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta, "The SPLASH-2 Programs: Characterization and Methodological Considerations," 22nd Annual International Symposium on Computer Architecture June 1995

[4] TILERA Corporation, "Programming the Tile Processor," Multi-Core Development Environment MDE 3.0 Document, UG205

[5] TILERA Corporation, "MDE User Guide," Multi Core Development Environment MDE3.0 Document, UG209

[6] TILERA Corporation, "Optimization Guide," Multi Core Development Environment MDE3.0 Document, UG105

[7] TILERA Corporation, "Application Libraries Reference Guide," Multi Core Development Environment MDE 3.0 Document, UG227

[8] TILERA Corporation, "Tile Processor User Architecture Reference," Multi Core Development Environment MDE 3.0 Document, UG101