DOI: 10.3745/KIPSTA.2008.15-A.5.259

멀티-코어 서버의 성능 분석 및 특성화

이 명 호† · 강 준 석††

요 약

멀티-코어 프로세서는 최근 마이크로프로세서 시장의 주류 제품으로 자리 잡았다. 이러한 멀티-코어 프로세서를 기반으로 하는 서버들은 고성능 컴퓨팅 분야와 상용 응용 프로그램 분야에서 그 사용 범위를 넓혀가고 있다. 멀티-코어 프로세서는 높아진 병렬성으로 인하여 응용 프로그램의 성능도 한 단계 더 높여줄 것으로 기대된다. 하지만, 칩 내부의 여러 코어들이 공유 자원들을 사용하면서 발생하는 경쟁과 충돌이 성능에 병목으로 작용하기도 한다. 그러므로 멀티-코어 서버 상에서 높은 성능과 확장성을 얻기 위해서는 공유 자원의 사용을 최적화 하는 것이 필수적이다. 본 논문에서는 코어들 간의 공유 자원 사용에서 발생하는 긍정적/부정적인 효과들이 실제 응용 프로그램의 성능에 어떻게 반영되는지 실험을 통하여 분석해 본다. 또한 이러한 분석을 통하여 멀티-코어 서버의 성능을 특성화한다.

키워드 : 멀티-코어 마이크로프로세서, 고성능 컴퓨팅, 쓰레드 수준의 병렬성, 쓰레드 배치법

Performance Analysis and Characterization of Multi-Core Servers

Myungho Lee† · Jun Suk Kang††

ABSTRACT

Multi-Core processors have become main-stream microprocessors in recent years. Servers based on these multi-core processors are widely adopted in High Performance Computing (HPC) and commercial business applications as well. These servers provide increased level of parallelism, thus can potentially boost the performance for applications. However, the shared resources among multiple cores on the same chip can become hot spots and act as performance bottlenecks. Therefore it is essential to optimize the use of shared resources for high performance and scalability for the multi-core servers. In this paper, we conduct experimental studies to analyze the positive and negative effects of the resource sharing on the performance of HPC applications. Through the analyses we also characterize the performance of multi-core servers.

Keywords : Multi-Core Microprocessor, High Performance Computing, Thread-Level Parallelism, Thread Placement

1. 서 론

반도체 공정기술의 끊임없는 발전으로 트랜지스터 미세공정이 매우 세밀해 짐에 따라, 마이크로프로세서 칩 안에 집적시킬 수 있는 트랜지스터의 개수가 크게 증가되어 왔다. 이에 따라 상대적으로 넓어진 칩 영역의 사용을 어떻게 최적화 할 것인가에 관하여 마이크로프로세서 설계자들은 많은 디자인들을 고려해왔다. 이러한 관점에서 최근의 가장 두드러진 추세는 여러 개의 CPU 코어를 하나의 칩 안에 집적시키는 멀티-코어 마이크로프로세서라고 할 수 있다. 최초의 상용 멀티-코어 마이크로프로세서는 2004년 Sun Microsystems Inc.에서 출시된 UltraSPARC IV[15]로서 2002년에 개발된 UltraSPARC III-Cu CPU 코어 두 개를 하나의 칩 안에 집적시킨 Dual-코어 제품이었다. 그 이후 Intel[3], AMD[1], IBM[5], Fujitsu, 등 거의 모든 마이크로프로세서 제조업체들이 멀티-코어 제품들을 출시해왔다. Intel과 AMD는 Dual-코어를 넘어서 2007년에 Quad-코어 제품들을 상용화 시켰다. Sun의 경우, 2007년에 8개의 코어들이 내장된 UltraSPARC T2 마이크로프로세서를 출시한 바 있다[17]. Intel도 2008년 말까지 6개의 코어가 내장된 제품을 발표할 계획이다. 이러한 멀티-코어 프로세서의 개발은 앞으로도 가속화 되어 차세대 제품은 코어의 개수가 현재의 제품에 비해 획기적으로 높아진 Many-코어의 시대를 열 것으로 전망된다.

멀티-코어 마이크로프로세서는 기존의 Single-코어 마이크로프로세서들과 비교하면 칩 내부의 쓰레드 수준의 병렬성(Thread-Level Parallelism: TLP)이 크게 향상되어, 이론

※ 이 논문은 2006년도 정부(교육인적자원부)의 재원으로 한국학술 진흥재단의 지원을 받아 수행된 연구임(KRF-2006-311-D00785).
※ 본 연구의 실험에 사용된 Sun Fire E25K 서버는 독일 RWTH Aachen 대학의 Center for Computing and Communications의 지원을 받아 사용되었음.
† 정 회 원 : 명지대학교 컴퓨터소프트웨어학과 부교수
†† 정 회 원 : (주)다우기술 SM 사업본부 투자정보 시스템
논문접수 : 2008년 5월 6일
수 정 일 : 2008년 6월 17일
심사완료 : 2008년 7월 2일

적으로는, 높아진 TLP에 비례한 성능향상을 기대할 수 있다[10, 14, 18]. 하지만 멀티-코어 프로세서에서는 기능 유닛들과 캐시 메모리, 캐시 버스, 메모리 버스 등의 자원들이 여러 코어들 간에 공유되는데, 이러한 공유 자원들의 사용을 위한 코어들 간의 경쟁과 충돌이 전체 성능에 부정적인 결과를 가져오기도 한다[2, 7, 14]. 공유 자원들의 사용은, 제한적이기는 하지만, 성능에 긍정적으로 작용하기도 한다. 예로서, 한 코어에 의해 공유 캐시에 적재된 데이터를 여러 코어들이 함께 사용함으로써 협조적인 씨너지 효과를 가져다주는 경우를 들 수 있다. 그러므로 멀티-코어 마이크로프로세서 기반의 서버 상에서 높은 성능과 확장성을 얻기 위해서는 코어들 간의 공유 자원 사용을 최적화하는 것이 중요하다. 이러한 관점에서, 본 연구에서는 코어들 간의 공유 자원 사용에서 발생하는 긍정적/부정적인 효과들이 실제 응용 프로그램의 성능에 어떻게 반영되는지 실험을 통하여 분석해 본다. 특별히 Sun UltraSPARC IV 마이크로프로세서를 탑재한 공유 메모리 다중 프로세서(Shared-Memory Multi-Processor: SMP) 서버인 Sun Fire E25K[15]를 실험 플랫폼으로 하여 이러한 영향을 분석한다.

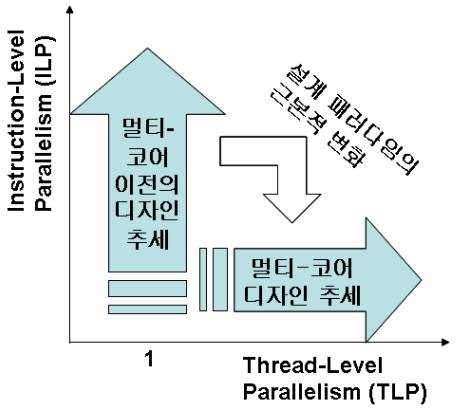
본 연구를 위해 일련의 성능 측정 실험을 수행한다. 먼저, 세 가지의 쓰레드 배치법*(NC, C, H)*을 사용하여 코어들 간에 공유자원에 대한 충돌이 전혀 발생하지 않는 경우*(NC)*, 모든 공유자원에 대한 충돌이 발생하는 경우*(C)*, 일부 공유자원에서만 발생하는 경우*(H)*등을 구성한다. 이러한 세 가지 경우들에서 응용 프로그램들의 성능측정 실험을 수행한다. 실험 결과를 토대로 코어들이 특정 자원을 공유함으로써 얻는 긍정적인 효과와 부정적인 효과들을 분석한다. 성능 분석 결과 실험 플랫폼인 Sun Fire E25K에서 공유 자원에 대한 충돌로 가장 성능 저하를 가져오는 곳은 메모리 버스인 것을 알 수 있는데, 메모리 버스 사용을 최소화시키는 skewed tiling 기법[8]을 적용하면 얼마나 성능을 향상시킬 수 있는지 그 효용성을 평가한다. 또한 이를 통하여 앞에서 얻은 성능 분석 결과를 검증한다. 성능 측정 실험에 사용되는 응용 프로그램은 SMP 시스템을 위한 대표적인 고성능 컴퓨팅(High Performance Computing: HPC)용 벤치마크인 SPEC OMPL suite[13]이다. Sun Studio 10 컴파일러[16]를 사용하여 SPEC OMPL의 각 벤치마크들에 대한 최적화된 실행 파일을 생성한 후, Sun Fire E25K 서버 상에서 실행시켜 성능측정 실험을 수행한다.

본 논문의 나머지 부분들은 다음과 같이 구성되었다. 2장에서는 배경 정보로서 멀티-코어 마이크로프로세서들의 발전 추세 및 특성에 관하여 설명한다. 3장에서는 본 연구의 실험을 위해 사용된 시스템과 사용된 벤치마크 프로그램에 대한 내용을 자세히 설명한다. 4장에서는 공유자원에 대한 충돌이 성능에 미치는 긍정적/부정적 영향을 관찰하기 위한 성능측정 방법 및 실험 결과를 자세한 분석과 함께 보인다. 5장에서는 메모리 대역폭 사용을 최소화 할 수 있는 skewed tiling 기법을 적용하여 그 효용성을 성능 측정 결과로서 검증하며, 4장의 분석 결과를 검증한다. 6장에서는 본 논문에서 얻은 결론과 향후 연구 방향을 제시한다.

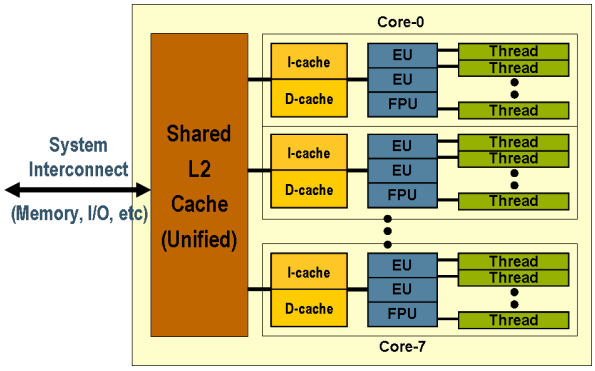
2. 멀티-코어 마이크로프로세서의 발전 추세 및 특성

반도체 공정기술의 끊임없는 발전으로 트랜지스터의 집적도가 크게 증가되어 왔다. 이는 상대적으로 마이크로프로세서 칩 영역이 확장되는 효과를 가져왔다. 확장된 칩 영역의 사용을 최적화하기 위하여 이전의 설계자들은 주로 명령어 수준의 병렬성(Instruction-Level Parallelism: ILP)을 최대한 활용하는 방법을 사용해왔다. 즉, 칩 내부에 여러 기능 유닛들을 추가하거나, 프로세서의 클럭을 높이기 위해 파이프라인을 복잡하게 설계하는 등의 디자인들을 사용하는 한편, DRAM과 프로세서간의 속도차이를 줄이기 위한 방법으로 칩 내부에 집적되는 캐시 메모리의 용량 및 단계(Hierarchy)를 증가시키는 등의 디자인이 사용되어 왔다. 그러나 이러한 디자인들은 프로세서를 설계하는데 드는 시간과 비용은 큰 폭으로 증가시킨 반면, 응용 프로그램을 설계된 마이크로프로세서 상에 실행 시 한계적인(marginal) 성능 향상만 보임으로써 제한적인 성공만을 거두게 되었다. 이는 새로운 프로세서 설계 방식의 필요성을 부각시켰다[2, 14].

최근 마이크로프로세서 설계의 가장 두드러진 추세는 여러 개의 CPU 코어를 같은 칩 안에 집적시키는 멀티-코어 프로세서라고 할 수 있다. 멀티-코어 마이크로프로세서에서는 이전의 디자인에서와는 달리 각각의 코어들을 복잡하게 설계하는 대신, 비교적 단순한 코어 여러 개를 칩 안에 집적시킴으로써 설계에 드는 시간과 비용을 단축할 수 있었다. 대신 이전의 디자인들이 ILP의 극대화를 통하여 칩 안에 집적된 한 코어의 성능을 극대화하는 것을 목표로 삼았던 반면, 멀티-코어 프로세서는 쓰레드 수준의 병렬성(Thread-Level Parallelism: TLP)을 활용하여 여러 코어들의 종합적 성능(aggregate performance) 또는 단위 시간당 전체 처리량(Throughput)을 극대화하는 것을 목표로 함으로써 근본적인 디자인 패러다임의 변화를 가져왔다((그림 1) 참조). 최초의 멀티-코어 마이크로프로세서인 Sun UltraSPARC IV는 2004년에 출시된 Dual-코어 제품이다. 2007년에 Intel, AMD 등



(그림 1) 마이크로프로세서 설계 패러다임의 전환



(그림 2) 고급 멀티-코어 마이크로프로세서(Sun UltraSPARC T2)의 구조

은 Dual-코어를 넘어서 Quad-코어 제품들을 출시했고, 공정을 더욱 미세화 하여 최근에는 45nm 공정의 Quad-코어 제품까지 상용화 시켰다. 이처럼 세밀화 된 공정을 이용한 멀티-코어 마이크로프로세서의 개발이 가속화되면서 멀티-코어 마이크로프로세서는 이제 시장의 주류로 자리 잡았다.

트랜지스터 집적도의 발전은 앞으로도 계속될 것으로 전망되며, 그에 따라 멀티-코어 마이크로프로세서들도 더욱 발전해 갈 것으로 예상된다. 위의 (그림 2)는 가장 최신의 멀티-코어 프로세서인 Sun UltraSPARC T2[17]의 구조를 보인다. UltraSPARC T2의 특징을 요약해 보면,

- 획기적으로 높아진 칩 수준의 병렬성 : 칩 안에 내장된 코어들의 개수가 8로 늘어나고, 각 코어는 8 개의 쓰레드들을 실행할 수 있게 되어 칩 전체의 TLP가 64로 높아졌다. 이를 현재의 Dual-코어나 Quad-코어에 비교하면 16~32배나 향상 된 것이다. 늘어난 TLP는 두 단계(코어 수준, 쓰레드 수준)로 이루어진다.
- 공유 자원의 사용 : 각 코어의 8개의 쓰레드들은 두 개의 명령어(Instruction) 파이프라인들(실행 유닛들)과 하나의 부동 소수점(Floating-Point: FP) 유닛을 공유한다. 여러 코어들은 칩에 내장된 4MB 크기의 level-2 (L2) 캐시를 공유한다.
- 다 단계 캐시 구조 : 코어 및 쓰레드의 개수가 늘어남에 따라, 캐시의 구조가 따라서 복잡해진다. 용량이 작은 level-1(L1) 명령어 캐시(16KB)와 데이터 캐시(8KB)들은 각 코어 전용으로 사용되고, L2 캐시는 모든 코어가 공유하도록 다 단계 구조를 갖는다.

UltraSPARC T2는 주로 TLP가 높은 웹(web)이나 데이터베이스 등과 같은 메모리와 디스크의 접근이 빈번한 응용 프로그램의 실행에 장점을 갖도록 설계되었다. 가까운 장래에 UltraSPARC T2와 유사한 구조를 갖고, 실행 유닛들과 FP 유닛 등이 T2보다 향상된 제품들이 출시되어 범용 멀티-코어 마이크로프로세서의 주류가 될 것으로 예상된다. 예를 들면 2009년 출시 예정인 Sun의 UltraSPARC VII를 들 수 있다. UltraSPARC VII은 칩 안에 8개의 코어가 집적되고, 코어 당 8개의 쓰레드를 지원하는 범용 마이크로프로세서이다. Intel 역시 2008년 말에 6개의 코어가 집적된 제품을 발표할 계획이며, 내부적으로 80개의 코어가 집적된 프로세서를 연구 개발 중이다. 이러한 추세가 계속된다면, 차세대 범용 멀티-코어 마이크로프로세서는 코어의 수가 획기적으로 증가되어 멀티-코어를 뛰어넘어 Many-코어의 시대를 열 것으로 예상된다.

멀티-코어 프로세서 기반의 서버 상에서 응용 프로그램의 높은 성능을 얻기 위해서는 이러한 멀티-코어 프로세서들의 특성을 잘 활용한 최적화 기법(컴파일러, 알고리즘, 등)을 적용하는 것이 중요하다. 멀티-코어 프로세서의 특징들 중 기존의 single-코어 프로세서와 비교하여 가장 큰 차이점은 여러 코어들과 코어 내부의 쓰레드들이 공유하는 자원들(예를 들어, 기능 유닛들, 캐시, 캐시 버스, 메모리 버스, 등)이라고 할 수 있다. 여러 쓰레드와 코어들은 공유 자원들의 사용을 위하여 경쟁하게 되는데,

(1) 이러한 경쟁은 공유 자원에 대한 충돌로 이어져 전체 성능에 부정적인 결과를 가져온다. 예를 들면, 같은 칩 안에 위치한 두 개의 코어들이 필요한 데이터의 접근을 위하여 공유 캐시와 메모리에 접근하는 경우, 캐시 버스와 메모리 버스의 사용을 위하여 충돌하거나, 캐시/메모리 뱅크의 사용을 위하여 충돌하는 경우들을 들 수 있다. 또한, 1번 코어가 사용하려고 캐시 블록에 적재시켜 놓은 데이터를 2번 코어가 나중에 다른 데이터를 같은 캐시 블록에 적재하여 대체(replace)시킴으로써 적재된 데이터를 파괴(overwrite)하는 등의 경우가 발생할 수 있다.

(2) 다른 한편으로, 이러한 경쟁은 제한적이나마 협조적인 씨너지 효과를 가져 올 수도 있다. 예를 들어, 1번 코어가 사용하려고 캐시 블록에 적재시켜 놓은 데이터를 2번 코어도 나중에 사용 할 경우 필요한 데이터가 미리 적재되는 프리페치(prefetch) 효과를 얻게 되어 성능에 긍정적인 영향을 미칠 수 있다.

따라서 전체 성능을 향상시키기 위하여 쓰레드들과 코어들 간의 공유 자원 사용을 최적화하는 것이 중요하다. 이러한 공유 자원 사용을 위한 경쟁과 충돌이 성능에 미치는 영향은 코어의 개수가 획기적으로 증가할 차세대 Many-코어 프로세서에서는 더 크게 증가할 것으로 예상된다.

3. 실험 환경

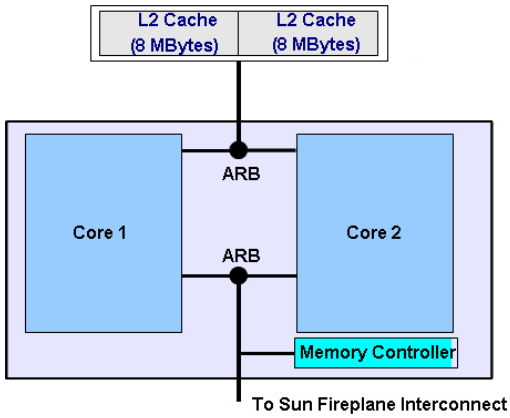
본 논문에서는 여러 가지 성능측정 실험을 수행하는데, 다음과 같은 실험 환경을 사용한다. 실험을 수행할 플랫폼으로 1세대 멀티-코어 마이크로프로세서 Sun UltraSPARC IV를 탑재한 최고 사양(high-end)의 SMP 서버 Sun Fire E25K[15]를 사용 한다. 성능 측정에 사용될 응용 프로그램은 SMP 시스템을 위한 대표적인 HPC용 벤치마크인 SPEC OMPL[13]이다. 아래의 절들에서는 Sun Fire E25K 서버와 SPEC OMPL에 대하여 자세히 설명한다.

3.1 Sun Fire E25K

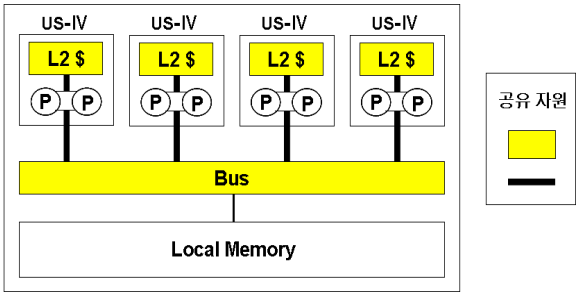
Sun Microsystems사의 Sun Fire E25K는 72개의 UltraSPARC IV 칩들을 기반으로 한다. 각 UltraSPARC IV에는 두 개의 UltraSPARC III Cu 프로세서(UltraSPARC IV 이전 세대의 프로세서로서 2002년에 출시) 코어가 내장된다((그림 3) 참조). 그 이외에도 각각의 UltraSPARC IV는 메모리 컨트롤러, L2 캐시를 위한 캐시 태그(tag)를 포함하고 있다. L2 캐시는 칩 외부에 위치하며 크기는 16MB (코어당 8MB)이다. 두 개의 코어 들은 L2 캐시 버스(bus)와 Fireplane System Interconnect(메모리 버스)를 공유한다. 72개의 UltraSPARC IV 칩들이 각각 2개의 쓰레드를 실행할 수 있으므로, 시스템 전체로는 144개의 쓰레드를 동시에 실행하도록 설계되었다.

Sun Fire E25K서버를 구성하는 기본 컴포넌트는 UniBoard이다. 각 UniBoard는 네 개의 UltraSPARC IV 칩들과 메모리로 구성되어져있다((그림 4) 참조). 같은 UltraSPARC IV 칩에 위치한 CPU 코어들이 L2 캐시, 캐시 버스, 메모리 버스 등을 공유하는 반면, 다른 UltraSPARC IV 칩에 위치한 CPU 코어들은 메모리 버스만을 공유한다. (이러한 공유 자원들은 (그림 4)에서 색칠된 부분 또는 굵은 선으로 표시되어 있다.) Sun Fire E25K 시스템 전체로는 18개의 UniBoard가 있다. E25K에서 캐시 일관성(cache coherency)을 유지하기 위해 같은 UniBoard상에서는 버스기반의 스누핑(snooping) 프로토콜을 사용하며, 다른 UniBoard간에는 디렉토리(directory) 기반의 프로토콜을 사용한다. E25K는 대표적인 Non-Uniform Memory Access (NUMA) SMP 서버이다. 메모리 접근 시간(latency)을 lmbench의 lat_mem_rd 루틴을 사용하여 측정한 경우, 동일한 UniBoard 상에 있는 메모리 접근에는 240nsec, 다른 UniBoard의 메모리에 접근하는 경우에는 455nsec가 걸린다.

Sun Fire E25K는 Solaris 10을 기본 OS로 사용하는데, Solaris 10은 Sun Fire E25K 서버를 위한 확장성과 고성능을 제공한다[12]. Solaris 10의 기능 중 MPO (Memory Placement Optimization: 메모리 배치 최적화 또는 NUMA 지원)와 MPSS (Multiple Page Size Support: 다중 페이지 지원)는 HPC 응용 프로그램의 고성능 확보에 매우 효율적



(그림 3) Sun UltraSPARC IV 마이크로프로세서의 구조



(그림 4) UniBoard의 구조

으로 사용될 수 있다 :

- MPO는 특정 프로세서가 자주 접근하는 페이지를 그 프로세서의 접근이 용이하도록 될 수 있는 대로 가까운 곳에 있는 메모리(예 : 같은 UniBoard 상에 있는 메모리)에 배치한다. 이렇게 함으로써 메모리 접근이 빈번한 응용 프로그램 실행 시, 그들 중 많은 접근들이 가까운 메모리로 향하게 함으로써 성능 향상에 크게 기여할 수 있다. 특히 E25K 서버와 같은 NUMA 시스템에서는 MPO 기능이 성능에 큰 장점으로 작용할 수 있다.
- MPSS는 응용 프로그램으로 하여금 가상 메모리의 상이한 지역에 각각 다른 크기의 페이지를 사용하도록 지원한다. UltraSPARC IV 프로세서에서는 네 가지 크기의 페이지들(8KB, 64KB, 512KB, 4MB)을 하드웨어적으로 지원하는데, MPSS를 사용하면 이들 네 가지 페이지들 중 하나를 특정 메모리 영역에 사용하도록 요청할 수 있다. 예를 들면, 스택과 힙 영역에는 4MB 크기의 페이지를 할당하고, 그 이외의 메모리 영역에는 기본 페이지 크기인 8KB를 사용할 수 있다. MPSS를 사용하면 대용량 메모리를 사용하는 응용 프로그램을 실행할 때 작은 크기의 페이지를 사용할 때 보다 필요한 페이지 개수가 크게 줄어든다. 그러므로 Translation Look-aside Buffer (TLB)에서의 접근 실패 횟수를 대폭 줄임으로써 응용 프로그램의 성능을 크게 향상시킬 수 있다.

3.2 SPEC OMPL 벤치마크

SPEC OMPL 벤치마크 suite은 Standard Performance Evaluation Corp (SPEC)의 고성능 컴퓨팅 그룹(High Performance Group: HPG)에서 2002년 6월에 발표한 벤치마크로서 SMP 서버의 성능 측정을 위한 대표적인 HPC 벤치마크로 인정받고 있다[13]. SPEC OMPL은 C 와 Fortran을 이용하여 제작된 아홉 가지의 응용 프로그램들로 구성되어 있고 OpenMP[11] 지시어를 사용하여 병렬화 되었다. 이들 벤치마크 들은 화학, 기계공학, 기상 모델링, 물리학 등의 분야를 대표하는 응용 프로그램 들이다(<표 1> 참조). 각 벤치마크는 한 개의 프로세서에서 실행 시 6.4GB 이상의 메모리를 필요로 한다. 그러므로 벤치마크들을 실행하기 위해서는 64-bit 주소공간을 사용하는 대용량 시스템이 필요하게 된다. 각각의 SPEC OMPL 벤치마크에 대한 최적화된 실행 파일을 생성하기 위하여 Sun Studio 10 컴파일러를 사용하

였다. Studio 10 컴파일러[16]는 OpenMP 버전 2.5를 지원하며, 실행 파일의 최적화를 위한 다양한 옵션들을 제공한다. -fast와 같은 고급 최적화 옵션들을 사용하여 각각의 벤치마크들을 컴파일 하였다.

〈표 1〉 SPEC OMPL 벤치마크들

벤치마크	응용 분야	프로그래밍 언어
311.wupwise_l	Quantum chromodynamics	Fortran
313.swim_l	Shallow water modeling	Fortran
315.mgrid_l	Multi-grid solver	Fortran
317.applu_l	Partial differential equations	Fortran
321.equake_l	Earthquake modeling	C
325.apsi_l	Air pollutants	Fortran
327.gafort_l	Genetic algorithm	Fortran
329.fma3d_l	Crash simulation	Fortran
331.art_l	Neural network simulation	C

4. 성능 특성화를 위한 실험 결과

이 장에서는 멀티-코어 프로세서의 서로 다른 코어들이 공유자원을 사용하면서 발생하는 경쟁과 충돌이 전체 성능에 미치는 영향을 실험을 통하여 관찰하고 그 원인을 분석해 본다. 4.1절에서는 성능측정 실험을 위해 사용된 세 가지 쓰레드 배치(Thread Placement) 방법에 대하여 설명한다. 4.2절에서는 이러한 배치법들을 사용하여 측정한 성능 결과 및 분석을 보인다.

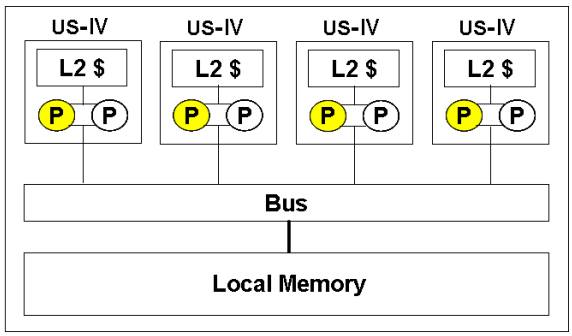
4.1 쓰레드 배치법

Sun Fire E25K 서버를 구성하는 기본 컴포넌트인 UniBoard는 네 개의 UltraSPARC IV 칩들과 메모리로 구성되며, 같은 UltraSPARC IV 칩에 위치한 CPU 코어들은 L2 캐시, 캐시 버스, 메모리 버스 등을 공유하고, 다른 UltraSPARC IV 칩에 위치한 CPU 코어들은 메모리 버스만을 공유한다. 이러한 공유자원들의 사용 유형을 반영하기 위하여 다음과 같은 세 가지 쓰레드 배치법을 고려한다.

1) No-Conflicts *(NC)*: 응용 프로그램을 Sun Fire E25K 서버에서 실행하기 위해 프로그램의 쓰레드들을 CPU 코어들에 배치할 때, 각 UltraSPARC IV에 있는 두개의 코어들 중 하나에만 쓰레드를 배치.
2) Conflicts *(C)*: UltraSPARC IV 상에 있는 두개의 코어들에 각각 쓰레드를 하나씩 배치.
3) Hybrid *(H)*: (1)과 (2)가 혼합된(Hybrid) 배치법.

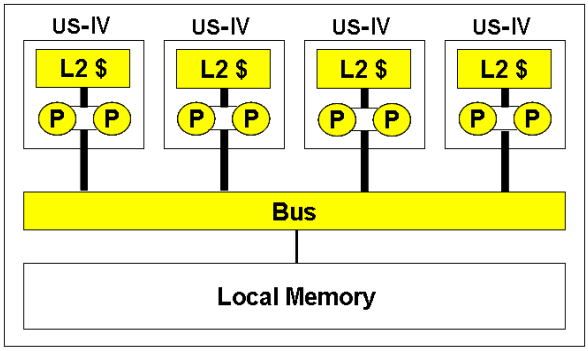
각 쓰레드 배치법에 대한 자세한 사항은 아래와 같다.

1) *(NC)*에서는 각 UltraSPARC IV에 있는 두 개의 코어들 중 하나에만 쓰레드를 배치하므로 하나의 코어가 L2 캐시와 캐시 버스를 단독으로 사용하여 경쟁과 충돌을 피하게 된다. L2 캐시를 코어들이 공유하지 않음으로써 코어들 간의 협조적인 씨너지 효과는 기대할 수 없다. 메인 메모리로 연결되는 메모리 버스를 다른 UltraSPARC IV 상에 있는 코어들과 공유하지만, (그림 5)에서 보듯이 각 UniBoard 상에 메모리 버스를 사용하는 코어의 개수가 4로 제한되어 메모리 버스에 걸리는 부하가 낮다.



• 칩 당 하나의 코어만 사용됨
• L2 캐시, 캐시 버스 등의 자원을 공유하지 않음
• 메모리 버스에 걸리는 부하가 낮음

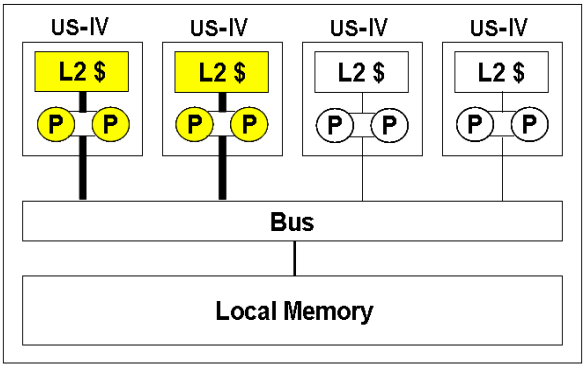
(그림 5) 코어 간에 자원을 공유하지 않는 경우(*NC*)



• 두개의 코어가 다 사용됨
• L2 캐시, 캐시 버스가 공유됨
• 메모리 버스에 걸리는 부하가 높아짐

(그림 6) 코어 간에 자원을 완전히 공유하는 경우(*C*)

2) *(C)*에서는 하나의 UltraSPARC IV 상에 있는 두 개의 코어들이 L2 캐시, 캐시 버스, 메모리 버스 등을 모두 공유함으로써 이들 공유자원에 동시에 접근하는 코어의 수가 *(NC)*의 경우에 비해 2배로 증가되어 경쟁과 충돌이 높아지게 된다((그림 6) 참조). 각 UniBoard에서 메모리 버스를 사용하는 코어의 개수가 8이 되므로, *(NC)*에 비하면 메모리 버스에 걸리는 부하도 두 배로 높아진다. 반면 L2 캐시의 공유로 인하여, L2 캐시 사용 시 코어들 간의 협조적인 프리페치와 같은 씨너지 효과를 기대할 수 있다.
3) *(H)*에서는 한 UniBoard 상에 있는 네 개의 UltraSPARC IV들 중 두 개의 UltraSPARC IV들에서는 두 개의 코어들에 모두 쓰레드들을 배치하고, 나머지 두 개의 UltraSPARC IV들에는 쓰레드를 배치하지 않는다. 쓰레드들이 배치된 UltraSPARC IV들에서는 *(C)*의 경우와 같이 L2 캐시, 캐시 버스들의 공유가 일어난다. 따라서 이들 자원들에 대한 경쟁과 충돌이 발생하고, L2 캐시에서의 협조적인 프리페치 효과도 발생한다. 각 UniBoard



• 두개의 US-IV에서만 두개의 코어가 다 사용됨
• 두개의 US-IV에서만 L2 캐시, 캐시 버스가 공유됨
• 메모리 버스에 걸리는 부하는 (NC)의 경우 수준으로 낮음

(그림 7) 코어 간에 자원을 일부 공유하는 경우(*H*)

상에 쓰레드가 배치된 코어들은 4이므로, 메모리 버스에 걸리는 전체 부하는 (NC)의 경우와 같다((그림 7) 참조).

아래의 <표 2>는 위의 세 가지 쓰레드 배치법들에서 발생하는 경쟁/충돌 및 씨너지 효과를 비교하여 보인다 :

- *(NC)*와 *(C)* : 모든 자원의 공유가 일어나는 경우*(C)*와 전혀 일어나지 않는 경우*(NC)*들의 비교이다. *(C)*에서는 공유하는 모든 자원에서 경쟁과 충돌을 일으키게 되지만, L2 캐시의 공유를 통한 씨너지 효과를 관찰할 수 있다.
- *(NC)*와 *(H)* : 메모리 버스에 걸리는 부하는 같고 L2 캐시, 캐시 버스를 공유하는 경우*(H)*와 공유하지 않은 경우*(NC)*들의 비교이다. L2 캐시, 캐시 버스를 공유함으로써 발생하는 충돌과 경쟁의 부정적 영향을 L2 캐시의 공유에서 발생하는 씨너지 효과와 비교할 수 있다.
- *(C)*와 *(H)* : L2 캐시 및 캐시 버스는 공유하는데, 메모리 버스에 걸리는 부하가 높은 경우*(C)*와 낮은 경우*(H)*들의 비교이다. 메모리 버스에 걸리는 부하의 영향을 관찰할 수 있다.

〈표 2〉 (*NC*), (*C*), (*H*) 비교

쓰레드 배치법	L2 캐시		캐시 버스 사용을 위한 충돌(-)	메모리 버스 사용을 위한 충돌이 증가(-)
	데이터를 코어들이 공유(+)	캐시 블록 및 bank에 대한 충돌(-)		
(NC)	X	X	X	X
(C)	O	O	O	O
(H)	O	O	O	X

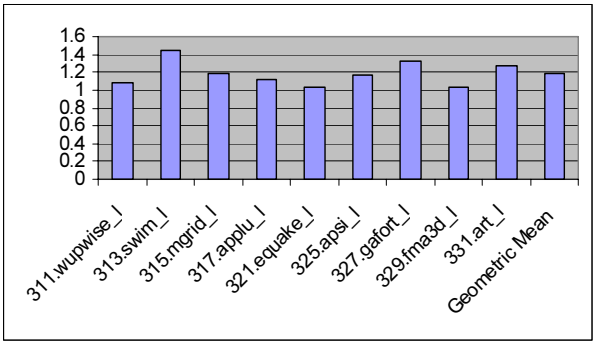
4.2 성능측정 결과 및 분석

SPEC OMPL의 9가지 벤치마크 프로그램들을 Sun Studio 10 컴파일러를 사용하여 실행 파일들을 생성하였다. 높은 수준의 최적화를 위하여 Studio 10 컴파일러에서 제공하는 옵션들 중 -openmp (OpenMP 지시어들을 처리), -fast (loop 변형, 명령어 스케쥴링, 등을 포함한 여러 가지 고급 최적화 작업들을 실행하는 옵션) 등을 사용하였다. 생성된 실행 파일들을 Sun Fire E25K 서버 상에 실행하기 위하여 64, 32 쓰레드를 사용한다. 아래의 4.2.1절에서는 먼저 *(NC)*와 *(C)*의 쓰레드 배치법을 사용하여 얻은 성능 결과를 비교한 후, 4.2.2절에서 *(NC)*와 *(H)*를 비교하고, 마지막으로 4.2.3절에서는 *(C)*와 *(H)*를 비교한다.

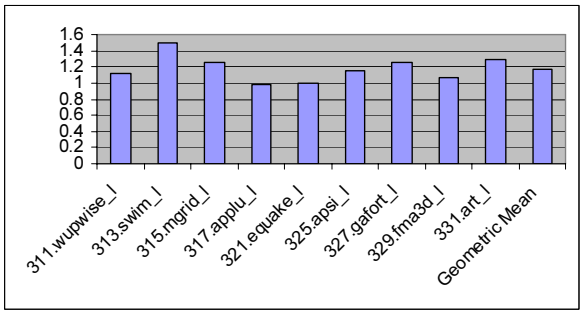
4.2.1 *(NC)*와 *(C)*의 성능 결과 비교

아래의 (그림 8)은 64개의 쓰레드를 사용했을 때, *(NC)*와 *(C)*의 성능측정 결과를 비교한 결과이다. 공유자원의 충돌이 없는 *(NC)*가 충돌이 있는 *(C)*보다 전체적으로 우수한데, 그림에서는 *(C)*의 실행시간을 *(NC)*의 실행 시간으로 나눈 스피드-업(Speed-Up) 결과를 그래프로 보인다. 각 벤치마크의 스피드-업 값들의 기하 평균 값을 구하면 *(NC)*가 전체적으로 18%가량 우수한 성능을 보인다. (그림 9)는 32개의 코어를 사용한 경우로, *(NC)*가 전체적으로 17%가량 우수한 성능을 보인다.

(그림 8)과 (그림 9)의 결과를 종합해보면, 코어들이 L2 캐시, 캐시 버스, 메모리 버스 등의 자원들을 공유하면서 발생하는 경쟁과 충돌로 인한 부정적인 효과가 L2 캐시 공유로 인하여 발생하는 긍정적 효과를 압도함을 알 수 있다. *(NC)*가 비교적 큰 스피드-업을 보이는 벤치마크들인 313.swim_l, 315.mgrid_l, 325.apsi_l, 327.gafort_l, 331.art_l 등의 경우, 높은 메모리 대역폭을 요구하거나 큰 용량의 메모리를 할당하여 사용한다는 특징이 있다 :



(그림 8) 64 쓰레드를 사용하여 (*NC*)와 (*C*)를 비교한 실험 결과



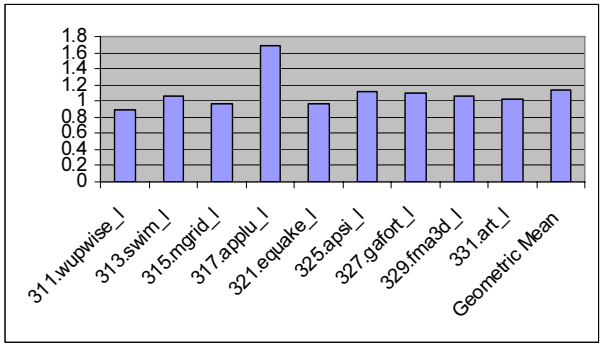
(그림 9) 32 쓰레드를 사용하여 (*NC*)와 (*C*)를 비교한 실험 결과

- 313.swim_l의 경우 메모리 대역폭 사용이 높은 벤치마크로서 코어 하나만 사용하는 *(NC)*가 캐시 버스 및 메모리 버스를 더 잘 활용할 수 있게 되어 1.45배나 되는 큰 성능 향상이 이루어진다.
- 315.mgrid_l의 경우도 313.swim_l과 같이 높은 메모리 대역폭을 요구하는 벤치마크로서 *(NC)*가 캐시 버스 및 메모리 버스를 더 잘 활용할 수 있게 되어 큰 성능 향상을 보인다.
- 325.apsi_l는 동적으로 큰 크기의 메모리를 할당하여 사용하는 특징이 있다. 두 개의 코어들을 모두 사용하면 한 UniBoard에서 8개의 쓰레드들이 동시에 여러 개의 4MB 크기의 페이지를 메모리에 배치하려 하기 때문에 메모리에 병목 현상이 발생하게 된다. 하나의 코어만 사용하면 이러한 병목 현상이 줄어들어 성능이 약 20% 가량 향상된다.
- 327.gafort_l의 경우 실행 시간이 가장 많이 소요되는 두 개의 함수에서 임계 영역(Critical Section) 안에 많은 메모리 연산들(Load와 Store)이 실행된다. 두 개의 코어들을 모두 사용하면 8개의 쓰레드가 한 UniBoard 상에서 실행되어, 임계 영역에 진입하기 위해 접근하는 lock 들이 같은 UniBoard의 메모리나 캐시에 있을 확률이 상대적으로 높아지기 때문에 lock에 대한 접근이 빨라지는 장점이 있다. 반면, 메모리와 코어들 간에 데이터 이동이 잦기 때문에 메모리 대역폭 사용을 위한 더 많은 경쟁과 충돌이 발생하게 되어 성능 향상에 장애가 된다. 후자의 부정적인 영향이 전자의 긍정적 영향을 압도한다.
- 331.art_l의 경우 각 쓰레드 당 여러 개의 큰 배열 들을 동적으로 할당받아 사용한다. 두 개의 코어들을 모두 사용하게 되면 한 UniBoard에서 8개의 쓰레드들이 동시에 많은 양의 메모리를 동적으로 할당 받아 사용하려 경쟁하기 때문에 메모리에 병목 현상이 발생한다. 하나의 코어만 사용하면 쓰레드간의 메모리 할당 및 사용을 위한 경쟁이 줄어들어 성능 향상에 도움이 된다.

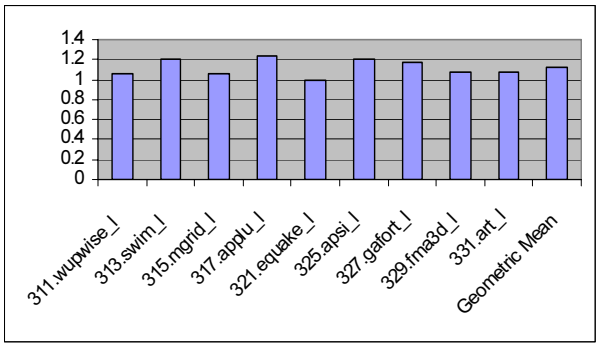
4.2.2 *(NC)*와 *(H)*의 성능 결과 비교

(그림 10)은 64개의 쓰레드를 사용했을 때, *(NC)*와 *(H)* 배치법을 사용하여 실행한 성능측정 비교 결과이다. *(H)*에서는 한 UniBoard 상에서 사용되는 두 개의 UltraSPARC IV들의 L2 캐시와 캐시 버스에서 충돌이 일어나고, 동시에 L2 캐시에서 프리페치 효과에 의한 데이터 공유 등의 씨너지 효과도 발생한다. *(H)*에서는 또 다른 공유 자원인 메모리 버스에 걸리는 부하가 공유자원 충돌이 전혀 없는 *(NC)*의 경우와 같게 된다. *(H)*가 *(NC)*보다 전체적으로 우수한 성능을 보인다. *(NC)*의 실행시간을 *(H)*의 실행 시간으로 나눈 스피드-업 결과를 그래프로 보여주는데, 각 벤치마크의 스피드-업 값들의 기하 평균 값을 구하면 *(H)*가 전체적으로 13%가량 우수한 성능을 보인다. (그림 11)은 32개의 쓰레드를 사용한 경우로, 여기서도 *(H)*가 *(NC)*보다 전체적으로 12%가량 우수한 성능을 보이고 있다.

- 위의 실험 결과를 분석해 보면 313.swim_l, 317.applu_l, 325.apsi_l, 327.gafort_l 벤치마크들의 경우 *(H)*가 특히 우수한 성능을 보이는데, *(H)*와 같이 메모리 버스에 걸리는 부하가 *(NC)*와 같은 수준으로 낮추어진 상태에서는 L2 캐시의 공유에서 오는 씨너지 효과가 L2 캐시 및 캐시 버스를 공유하면서 발생하는 부정적인 영향을 압도함을 볼 수 있다. 특히 이들 벤치마크들에서는 쓰레드들 간에 주변 데이터(Boundary Data)의 공유가 많다는 특징이 있는데, *(H)*를 사용하면 이러한 주변 데이터들이 L2 캐시에 적재되어 쓰레드들 간에 공유됨으로써 큰 성능 향상을 이룬다.
- 그러나 위의 벤치마크들 중 317.applu_l를 제외한 나머지 경우에서 64 스레드를 사용할 때보다 32 쓰레드를 사용할 경우 *(H)*의 장점이 더 크게 나타남을 볼 수 있다. 이러한 현상은 317.applu_l를 제외한 그 밖의 모든 벤치마크들에도 해당한다. 예로서 311.wupwise_l의 경우, 64개의 쓰레드 사용 시 *(NC)*가 *(H)*보다 10% 가량 높은 성능을 보이는 반면, 32 쓰레드 사용 시에는 *(H)*가 *(NC)*보다 4%가량 높은 성능을 보인다. 쓰레드의 개수가 32에서 64로 증가하면 쓰레드 당 working set의 크기가 줄어들어 L2 캐시에 적재해야 할 데이터의 요구량도 함께 준다. 따라서 쓰레드들 간에 함께 사용하는 주변 데이터를 공유 L2 캐시에 나누어 적재함으로써 얻는 이점도 줄어든다. *(H)*와 같은 씨너지 효과의 활용은 없지만 L2 캐시와 캐시 버스의 충돌이 없는 *(NC)* 배치법이 64 쓰레드 사용 시 더 높은 성능을 보인다. 즉 *(H)*는 캐시에 적재해야 할 데이터의 요구량이 높을 때 더 많은 성능 향상을 가져옴을 알 수 있다.
- 반대로 317.applu_l의 경우에는 32 스레드를 사용할 때보다 64 쓰레드를 사용할 경우 *(H)*의 장점이 더 크게 나타난다. 이 벤치마크에서는 동기화(synchronization)를 위한 lock/unlock 연산이 많이 실행되는데, L2 캐시를 공유하는 *(H)*가 lock에 대한 접근을 빠르게 할 수 있어 *(NC)*보다 좋은 성능을 보인다. 32 쓰레드의 경우보다 64 쓰레드의 경우 동기화를 위한 비용이 크기 때문에 *(H)*의 이점이 상대적으로 더 커진다.



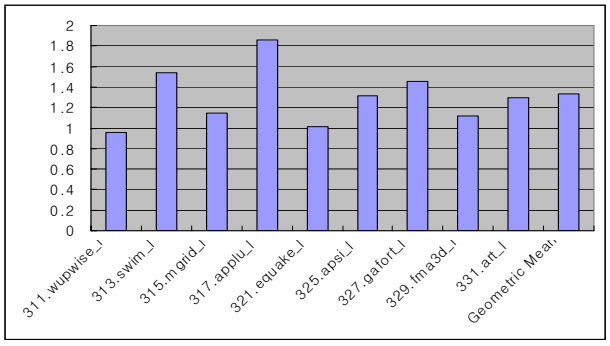
(그림 10) 64 쓰레드를 사용하여 *(NC)*와 *(H)*를 비교한 실험 결과



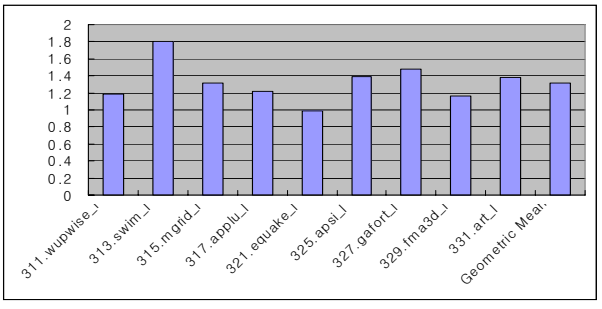
(그림 11) 32 쓰레드를 사용하여 (*NC*)와 (*H*)를 비교한 실험 결과

4.2.3 (*C*)와 (*H*)의 성능 결과 비교

아래의 (그림 12)와 (그림 13)은 각각 64, 32 쓰레드를 사용했을 때, *(C)*와 *(H)* 배치법을 사용하여 실행한 성능측정 비교 결과이다. *(C)*와 *(H)*를 사용하면 L2 캐시와 캐시 버스에서 충돌이 일어나고, 동시에 L2 캐시에서는 데이터 공유에 의한 프리페치 등의 씨너지 효과도 발생한다. *(C)*와 *(H)*의 다른 점은 메모리 버스에 걸리는 부하가 한 UniBoard 상에 있는 4개의 UltraSPARC IV를 모두 사용하는 *(C)*의 경우가 2배로 높은 것이다. *(H)*가 *(C)*보다 전체적으로 33%(64 쓰레드), 31%(32 쓰레드)나 높은 성능을 보인다. 이는 *(H)*의 *(NC)*에 대한 스피드-업 결과(64 쓰레드 : 1.13; 32 쓰레드 : 1.12)에 *(NC)*의 *(C)*에 대한 스피드-업 결과(64 쓰레드 : 1.18; 32 쓰레드 : 1.17)를 곱한 수준(64 쓰레드 : 1.33; 32 쓰레드 : 1.31)과 같다. 각각의 벤치마크 결과를 놓



(그림 12) 64 쓰레드를 사용하여 (*C*)와 (*H*)를 비교한 실험 결과



(그림 13) 32 쓰레드를 사용하여 (*C*)와 (*H*)를 비교한 실험 결과

고 보면 이와 같은 관찰이 잘 적용됨을 알 수 있다. 특히 313.swim_l, 317.applu_l, 327.gafort_l 등의 벤치마크들에서 큰 성능 차이를 보이는데, 이들 벤치마크들은 *(H)*의 장점인 주변 데이터의 공유와 *(NC)*의 장점인 메모리 버스에서의 부하 감소가 잘 활용되는 경우이다.

5. 메모리 사용 최적화 기법의 효용성

4장에서 얻은 실험 결과들 중 가장 특기할 점은 Sun Fire E25K에서 공유 자원들 중 가장 성능에 부정적인 영향을 미치는 부분은 메모리 버스라는 점이다. 따라서 메모리 버스의 사용을 최적화하는 것이 성능을 향상 시키는데 필수적이다. 이 장에서는 메모리 대역폭 사용을 최소화하는 기법인 skewed tiling[8] 기법을 적용하여, E25K 서버 상에서의 효용성을 실험을 통해 관찰한다. 또한 이러한 실험을 통하여 4장에서 얻은 결론을 검증한다.

5.1 Skewed Tiling 기법의 효용성

313.swim_l, 315.mgrid_l, 327.gafort_l 등의 벤치마크들은 메모리 대역폭 사용이 많은 프로그램들이다. 313.swim_l의 경우, 각각 453MB 크기의 2차원 배열 14개를 메모리로부터 스트리밍(streaming) 방식으로 적재(Load)하여 여러 부동소수점 연산을 수행하고 다시 메모리에 저장(Store)하는 특징이 있다. 배열들이 너무 크다보니 스트리밍 방식의 메모리 접근으로는 8MB 크기의 L2 캐시에 적재된 데이터들이 재사용될 확률이 거의 없으며, 캐시 미스 확률이 커진다. 또한 캐시 버스 및 메모리 버스 사용이 많아지게 된다. 공격적인 최적화 기법인 skewed tiling과 함수 인-라이닝(in-lining)을 적용하면 loop 내에 있는 함수 호출 때문에 tile 하기가 불가능한 loop들을 tile 하는 것이 가능해지는데,

- 먼저 피 호출 함수들을 loop 내부로 인-라이닝 시키고
- 피 호출 함수 내부에 있던 loop들을 tile 시킨 다음
- Tile 된 loop의 제어 loop를 피 호출 함수를 감싸고 있던 loop의 바깥으로 옮긴다.
- 그 이외에 인접한 loop들을 합병(fuse)하는 등의 최적화 작업이 수행된다.

위와 같은 방식으로 최적화시키면 메모리 접근 중 많은 부분들을 캐시 접근으로 바꿀 수 있다[8]. 이러한 최적화 기법의 멀티-코어 서버 상에서의 효용성을 검증하기 위하여 다음과 같은 실험을 수행 한다.

1) 먼저 아래의 두 가지 실행 파일을 생성한다.
   - (1-1) 실행 파일 : 4장의 실험에서 사용한 것과 같은 수준의 실행 파일을 Sun Studio 10 컴파일러를 사용하여 생성한다.
   - (1-2) 실행 파일 : Skewed tiling과 in-lining 최적화를 수행하는 Studio 10의 컴파일러 최적화 옵션을 적용하여 생성한다.

2) 실험 시나리오 :

- 실험 시간을 단축하기 위하여 각 입력 배열을 1/4 크기로(7702x7702 크기의 배열을 3802x3802 크기로) 줄여서 110MB 크기의 배열 14개를 사용하도록 한다. 따라서 전체 데이터 크기는 1,544MB가 된다.
- 쓰레드 개수 : 8개, 16개를 사용한다. 전체 데이터 크기 1,544MB를 쓰레드 수로 나누면 쓰레드 당 193MB 또는 96.5MB 크기의 데이터를 사용한다. 이들은 4장의 실험에서 사용된 쓰레드 당 데이터 크기와 거의 같다(각 배열의 크기 : 453MB; 배열 수 : 14; 쓰레드 수 : 32 또는 64; 쓰레드 당 배열 크기 : 198MB 또는 99MB). 193MB 또는 96.5MB는 각 코어 당 할당된 L2 캐시의 크기인 8MB를 크게 상회함으로써 최적화 되지 않은 실행 파일을 사용하면 한번 L2 캐시에 적재된 데이터가 재사용될 가능성은 거의 0에 가깝다.
- 쓰레드 배치법은 4장에서 설명한 여러 배치법들 중 메모리 버스 및 L2 캐시, 캐시 버스에 충돌과 경쟁을 발생시키는 *(C)*를 사용한다.

두 개의 실행 파일 (1-1)과 (1-2)를 위와 같은 실험 환경에서 실행하여 아래의 <표 3>과 같은 결과를 얻었다. Skewed tiling과 in-lining 최적화가 많은 메모리 접근들을 캐시 접근으로 바꿈으로써, 성능을 2.29배에서 2.49배까지 크게 향상시킴을 볼 수 있다.

〈표 3〉 Skewed tiling 최적화의 효능

쓰레드 개수	(1-1)의 실행 시간	(1-2)의 실행 시간	최적화로 인한 스피드-업
8	1431 sec	624 sec	2.29
16	1067 sec	428 sec	2.49

5.2 실험 결과 분석의 검증

이번 절에서는 쓰레드 배치법을 메모리 버스에서의 충돌과 경쟁이 없는 *(NC)*로 바꾼 후 5.1에서 얻은 (1-2)의 실행 파일들을 실행시켜 보았다. 이 실험은 메모리 버스에 걸리는 부하를 1/2로 줄임으로써, L2 캐시와 캐시 버스에서의 충돌이 성능에 어느 정도의 영향을 미치는 지 관찰하기 위함이다.

실험 결과 실행 시간이 620 sec (쓰레드 8개), 415 sec (쓰레드 16개)로서 *(C)*의 경우와 비교하여 1%~3%의 차이밖에 보이지 않는다. 즉, skewed tiling과 in-lining 최적화로 인하여 대부분의 메모리 접근들이 캐시 접근으로 바뀐 (1-2)의 실행 파일에서는 *(NC)*와 *(C)*의 쓰레드 배치법의 실행 시간들이 거의 차이가 없음을 보여주는 결과이다. L2 캐시와 캐시 버스에서 발생하는 충돌과 경쟁의 부정적 영향은 L2 캐시의 공유에서 오는 씨너지 효과로 거의 상쇄될 만큼 크지 않다. 바꾸어 말하면 UltraSPARC IV의 캐시 버스의 대역폭은 두 개의 쓰레드가 313.swim_l 벤치마크를 실행하며 동시에 사용하여도 성능에 별 다른 부정적 영향을 끼치지 않을 만큼 충분히 크며, Sun Fire E25K 멀티-코어 서버의 성능에 가장 심각한 영향을 끼치는 병목 부분은 메모리 버스라는 의미이다. 이는 4장의 실험에서 얻은 결론과 부합하는 것으로서 skewed tiling과 in-lining 최적화 기법을 적용하여 그러한 결론을 검증한 것이라고 할 수 있다.

6. 결 론

본 연구에서는 멀티-코어 마이크로프로세서 상에서 코어들 간의 공유 자원 사용에서 발생하는 긍정적/부정적인 효과들이 실제 응용 프로그램의 성능에 어떻게 반영되는지 실험을 통하여 분석해 보았다. 먼저 본 연구의 실험을 위해 사용된 시스템인 Sun Fire E25K 서버와 Sun UltraSPARC IV 마이크로프로세서에 대하여 설명하고, 실험에 사용된 벤치마크 프로그램 SPEC OMPL에 대하여 기술하였다. 실험에 사용된 세 가지의 *(NC), (C), (H)* 쓰레드 배치법에 대하여 설명한 후, 이러한 쓰레드 배치법들이 응용 프로그램의 성능에 반영된 실험 결과를 자세한 분석과 함께 보여주었다. 마지막으로 메모리 대역폭 사용을 최적화 할 수 있는 skewed tiling 기법을 적용하고, 그 효용성을 실험을 통하여 검증하였다. 또한 그러한 실험을 통하여 Sun Fire E25K 멀티-코어 서버의 성능에 가장 심각한 영향을 끼치는 병목 부분은 메모리 버스라는 결론을 검증하였다. E25K와 같은 멀티-코어 서버를 위하여 Skewed tiling과 같이 메모리 대역폭 사용 최적화 기법들을 향후 더 연구할 계획이다.

참 고 문 헌

[1] AMD Multi-Core: Introducing x86 Multi-Core Technology & Dual-Core Processors, http://multi-core.amd. com/2005

[2] Shailender Chaudhry, Paul Caprioli, Sherman Yip, and Marc Tremblay, High-Performance Throughput Computing, IEEE Micro, May-June, 2005.

[3] Intel Dual-Core Server Processor, http://www.intel.com/business/bss/products/server/dual-core.htm

[4] Intel Hyperthreading Technology, http://www.intel.com/technology/hyperthread/index.htm

[5] R. Kalla, B. Sinharoy, and J. Tendler, IBM POWER5 chip: a dual core multithreaded processor, IEEE Micro, March-April, 2004.

[6] Myungho Lee, Larry Meadows, Darryl Gove, Dominic Paulraj, Sanjay Goil, Brian Whitney, Nawal Copty, and Yonghong Song, Compiler Support and Performance Tuning of OpenMP Programs on SunFire Servers, European Workshop on OpenMP, Aachen, Germany,

September, 2003.
[7] Y. Li, D. Brooks, Z. Hu, K. Shadron, “Performance, Energy, and Thermal Considerations for SMT and CMP Architectures,” 11th International Symposium on High-Performance Computer Architecture, 2005.
[8] Zhiyuan Li, “Optimal Skewed Tiling for Cache Locality Enhancement,” International Parallel and Distributed Processing Symposium (IPDPS'03), 2003.
[9] Yuan Lin, Christian Terboven, Dieter an Mey, and Nawal Copty, Automatic Scoping of Variables in Parallel Regions of an OpenMP Program, 5th International Workshop on OpenMP Applications and Tools, Houston, Texas, May, 2004 (LNCS 3349).
[10] K. Olukotun et. al., The Case for a single Chip-Multiprocessor, International Conference on Architectural Support for Programming Languages and Operating Systems, 1996.
[11] OpenMP Architecture Review Board, http://www.o-penmp. org
[12] Solaris 10 Operating System, http://www.sun.com/ soft-ware/solaris
[13] The SPEC OMP benchmark suite, http://www.spec. org/omp
[14] L. Spracklen and S. Abraham, Chip MultiThreading: Opportunities and Challenges, 11th International Symposium on High-Performance Computer Architecture (HPCA-11), pp 248-252, 2005.
[15] Sun Fire E25K server, http://www.sun.com/serv-ers/highend/sunfire_e25k/index.xml
[16] Sun Studio 10 Compiler, http://www.sun.com/soft-ware/products/studio/index.html
[17] Sun UltraSPARC T2 microprocessor, http://www.sun. com/processors/UltraSPARC-T2
[18] D. Tullsen, S. Eggers, and H. Levy, Simultaneous MultiThreading: Maximizing On-Chip Parallelism, International Symposium on Computer Architecture, 1995.

이 명 호
e-mail : myunghol@mju.ac.kr
1986년 서울대학교 계산통계학과(학사)
1988년 미국 University of Southern California 컴퓨터과학과(석사)
1999년 미국 University of Southern California 컴퓨터공학과(박사)
1999년~2003년 미국 Sun Microsystems, Inc. Scalable Systems Group, 책임연구원
2003년~2005년 미국 Sun Microsystems, Inc. Scalable Systems Group, 수석연구원
2004년~2008년 명지대학교 컴퓨터소프트웨어학과 조교수
2008년~현 재 명지대학교 컴퓨터소프트웨어학과 부교수
관심분야 : 고성능 컴퓨팅, 병렬 알고리즘, 멀티-코어 마이크로프로세서, 컴파일러

강 준 석
e-mail : zepyr@naver.com
2006년 명지대학교 컴퓨터소프트웨어학과 (학사)
2008년 명지대학교 컴퓨터소프트웨어학과 (석사)
2008년~현 재 (주)다우기술 SM 사업본부 투자정보시스템
관심분야 : 고성능 컴퓨팅, 멀티-코어 마이크로프로세서, 성능 최적화