NUMA 기반의 스케줄러 설계를 위한 고려사항 분석

김정훈, 민창우, 엄영익
성균관대학교 정보통신대학
e-mail:{myhuni20, multics69, yieom}@skku.edu

Analysis of the Design Factors in NUMA-aware Scheduler

Junghoon Kim, Changwoo Min, and Young Ik Eom
College of Information and Communication Engineering, Sungkyunkwan Univ.

요 약

하드웨어 플랫폼은 다수 코어 아키텍처의 메모리 대역폭을 만족시키기 위해 NUMA 구조로 설계되고 있다. 이러한 NUMA 구조에서 다른 노드의 메모리에 접근할 경우, 해당 노드의 메모리 접근에 비해 1.5~2배 지연이 발생한다. 따라서 이러한 특성을 고려하는 NUMA 시스템 기반 스케줄러가 필요하다. 본 논문에서는 NUMA 기반 스케줄러 설계를 위해 고려되어야 할 사항에 대해 분석해 본다. 분석 결과, 공유 자원 경쟁과 리모트 접근을 최소화하는 것이 NUMA 스케줄러 설계의 핵심이라는 것을 확인할 수 있었다. 뿐만 아니라 같은 노드에서 실행되는 워크로드의 조합 및 캐시 오염 태스크 관리, 그리고 노드별 남아있는 메모리 정보 또한 고려되어야 한다는 것을 확인할 수 있었다.

1. 서론

최근, 하드웨어 플랫폼은 다수 코어(many-core) 아키텍처의 메모리 대역폭을 유지하기 위해 NUMA(Non-Uniform Memory Access) 구조[1][2]로 설계되고 있다. 이러한 NUMA 아키텍처 위에서 동작하는 프로그램들의 성능을 높이기 위해서는, 주어진 하드웨어 캐시를 효율적으로 사용하여 메모리 접근 횟수를 줄이는 노력이 필요하다. 뿐만 아니라 NUMA 구조에서 다른 노드의 메모리에 접근할 경우, 해당 노드의 메모리 접근에 비해 1.5~2배 지연이 발생하므로 이 또한 고려해야 한다. 즉, NUMA 구조 기반의 스케줄러 설계는 데이터 지역성(data locality)과 캐싱 효율 두 측면의 종합적인 고려가 필요하다.

이전의 연구들[3][4] 또한 앞서 설명한 두 가지 측면을 고려하여 스케줄러를 설계하였다. 하지만 이러한 연구들은 태스크 이동(task migration)과 태스크가 사용하는 페이지 이동(page migration)으로 문제를 해결하거나, 혹은 태스크들이 특정 노드에 메모리를 할당한다는 가정을 두고 문제를 해결한다. 이는 런타임 오버헤드가 크고, 일반적인 스케줄러로 활용하기 어렵다는 단점이 있다.

본 논문에서는 NUMA 구조 기반의 스케줄러 설계를 위한 고려사항에 대해 분석해 본다. 이는 앞서 설명한 연구들이 가지는 단점들을 보완하여, NUMA 기반 효율적인 스케줄러를 설계하기 위한 지표로 활용된다. 논문의 구성은 다음과 같다. 2장에서는 다양한 실험을 통하여, 워크로드들의 특성 및 현재의 스케줄러의 특성을 분석한다. 3장에서는 결론 및 향후 연구 계획을 끝으로 본 논문을 마무리한다.

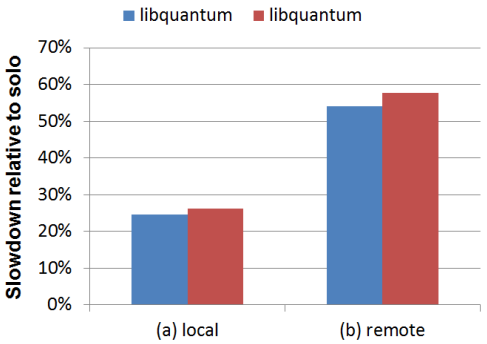
이 논문은 2012년도 정부(교육과학기술부)의 재원으로 한국연구재단-차세대정보컴퓨팅기술개발사업의 지원을 받아 수행된 연구임(No. 2012-0006423)

2. NUMA 기반 스케줄러 고려사항

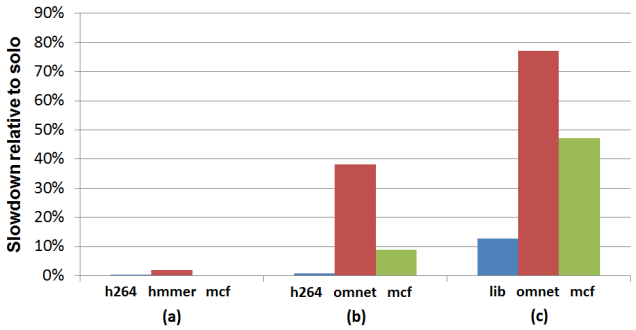
본 실험은 4개의 Intel Xeon 프로세서(프로세서 당 10 코어), 프로세서(노드) 당 8 GB 메모리를 가지는 환경에서 진행하였다. 그리고 워크로드는 SPEC CPU2006에 포함된 벤치마크를 사용하였다. 이는 CPU 중심, 메모리 중심, 그리고 캐시 오염 태스크(cache polluter)들을 포함하고 있다.

2.1 공유 자원 경쟁 및 리모트 접근 분석

그림 1은 공유 자원 경쟁 및 리모트 접근 오버헤드를 나타낸다. (a) local은 동일한 *libquantum* 프로그램 2개를



(그림 1) 공유 자원 경쟁 및 리모트 접근 오버헤드



(그림 2) 워크로드 조합 간 성능 차이 분석

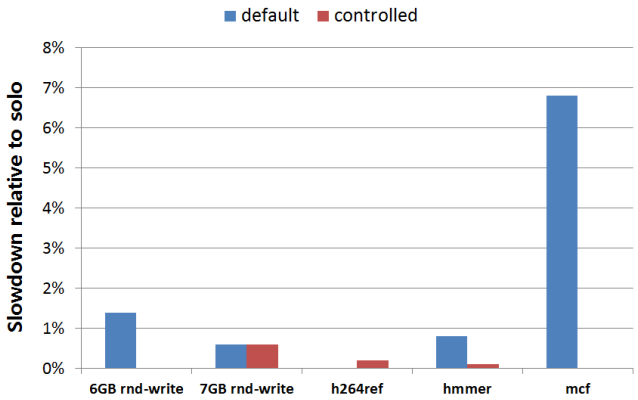
같은 로컬 노드에 메모리를 할당하여 실행한 결과이다. 그 결과, 솔로 실행 대비 약 25%의 성능 감소를 확인하였다. 이는 공유 자원(하드웨어 캐시 및 메모리 컨트롤러) 경쟁으로 인해 발생된다. (b) remote는 다른 노드에 메모리를 할당하여 실행한 결과이다. 그 결과, 솔로 실행 대비 약 55%의 성능 감소를 확인하였다. 이는 공유 자원 경쟁과 더불어 리모트 접근으로 인한 오버헤드까지 추가되었기 때문이다. 즉, 공유 자원 경쟁과 리모트 접근을 최소화하는 것이 NUMA 스케줄러 설계의 핵심이라 볼 수 있다.

2.2 워크로드 조합 간 성능 차이 분석

그림 2는 워크로드 조합 간의 성능 차이 분석 결과를 나타낸다. 여기서 워크로드들이 사용하는 메모리는 모두 로컬 노드에 할당하였다. 그리고 실험은 CPU 중심적인 워크로드(*h264ref, hmmer*)와 메모리 중심적인 워크로드(*mcf, omnetpp*), 그리고 캐시 오염의 원인이 되는 워크로드(*libquantum*) 조합으로 이루어졌다. (a) 실험은 하나의 메모리 중심적인 워크로드와 두 개의 CPU 중심적인 워크로드 조합 결과를 보여주는데, 솔로 실행 대비 성능 감소가 거의 없는 것을 확인할 수 있다. 반면 (b) 실험은 두 개의 메모리 중심적인 워크로드와 한 개의 CPU 중심적인 워크로드 조합 결과를 보여주는데, 메모리 중심적인 워크로드의 성능이 솔로 실행 대비 크게는 약 40% 가량 감소하는 것을 볼 수 있다. 마지막으로 (c) 실험은 캐시 오염 워크로드와 메모리 중심 워크로드 조합의 결과를 보여주는데, 메모리 중심 워크로드의 성능이 크게 약 80% 감소하였다. 여기서 우리는 메모리 중심적인 워크로드들을 노드 간 분산시키고, 캐시 오염 워크로드들을 관리하는 기법이 스케줄러 설계에 필요하다는 것을 확인하였다.

2.3 기존 OS 스케줄러 분석

그림 3은 기존 OS 스케줄러 분석 결과를 나타낸다. 실험은 6 GB, 7 GB 메모리 영역에 임의 쓰기를 하는 워크로드와 SPEC 워크로드 조합으로 이루어졌다. 그 결과, *mcf* 워크로드의 성능이 controlled 대비 크게 감소하는 것을 확인할 수 있다. 이는 기존 OS 스케줄러가 각 노드에 남아 있는 메모리 정보를 고려하지 않은 채, 태스크들을 배치하였기 때문이다. 여기서 controlled는 *mcf* 워크로드를 배치할 때, 노드별 남아있는 메모리 정보를 고려한 결과를



(그림 3) 기존 OS 스케줄러 분석

나타낸다. 따라서 노드별 남아있는 메모리 정보 또한 NUMA 스케줄러 설계에 고려되어야 한다는 것을 확인할 수 있었다.

3. 결론 및 향후 연구 계획

본 논문에서는 NUMA 기반 스케줄러 설계를 위해 고려되어야 할 점들에 대해 분석해 보았다. 그 결과, 공유 자원 경쟁과 리모트 접근을 최소화하는 것이 핵심이라는 것을 확인하였다. 또한 같은 노드에 배치되는 태스크들의 조합을 고려해야 하고, 캐시 오염 태스크들을 위한 기법이 필요하다는 것을 확인하였다. 마지막으로 노드별 남은 메모리 정보를 고려하여 스케줄링 할 필요가 있다는 것을 확인하였다. 향후에는 본 논문에서 분석한 결과를 바탕으로 NUMA 스케줄러를 설계하여, 다양한 워크로드들을 통해 실험을 진행할 계획이다.

참고문헌

[1] R. Maddox, G. Singh, and R. Safranek. The architecture of the intel quickpath interconnect. Technical report, Tech. Rep., Intel, 2009.

[2] E. Joseph, N. Kaumann, and C. Willard. The amd opteron processor: A new alternative for technical computing. *White Paper, IDC, November*, 2003.

[3] S. Blagodurov, S. Zhuravlev, A. Fedorova, and A. Kamali. A case for numa-aware contention management on multicore systems. In *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, pages 557 - 558. ACM, 2010.

[4] Z. Majo and T. Gross. Memory management in numa multicore systems: Trapped between cache contention and interconnect overhead. In *Proceedings of the international symposium on Memory management*, pages 11 - 20. ACM, 2011.