ETRI

2017
Electronics and
Telecommunications
Trends

매니코어 운영체제 연구현황 및 계획

Research Status and Plan for Manycore Operating System

정성인 (Sungin Jung, sijung@etri.re.kr) 차세대 OS 기초연구센터 책임연구원/센터장
김태수 (Taesoo Kim, taesoo@gatech.edu) 조지아공대 교수
민창우 (Changwoo Min, changwoo@vt.edu) 버지니아공대 교수
박성용 (Sungyong Park, parksy@sogang.ac.kr) 서강대학교 교수
변석우 (Sugwoo Byun, swbyun@ks.ac.kr) 경성대학교 교수
서의성 (Euiseong Seo, euiseong@skku.edu) 성균관대학교 교수
우 균 (Gyun Woo, woogyun@pusan.ac.kr) 부산대학교 교수
이경우 (Kyoungwoo Lee, kyoungwoo.lee@yonsei.ac.kr) 연세대학교 교수
이재욱 (Jaewook Lee, jaewlee@snu.ac.kr) 서울대학교 교수
임성수 (Sung-Soo Rim, sslim@kookmin.ac.kr) 국민대학교 교수
임은진 (Eun-Jin Im, ejim@kookmin.ac.kr) 국민대학교 교수
조희승 (Heeseung Jo, heeseung@jbnu.ac.kr) 전북대학교 교수
진현욱 (Hyun-Wook Jin, jinh@konkuk.ac.kr) 건국대학교 교수

The trend of manycore hardware has recently evolved more quickly than expected. However, an operating system, which is software used for managing computer resources, is still optimized for a multicore system. To handle this issue, we started a research project called 'Research on High Performance and Scalable Manycore Operating Systems' in 2014. This article briefly examines the technology trends of manycore hardware and operating systems, and introduces the research areas and outcomes during the first stage of the project(2014-2017). The core technologies improving the performance scalability of manycore systems are publicly available, and anyone can use the source code or apply the ideas of the core technique to other research activities. In addition, the research plans of the second stage of the project(2018-2021) are also included.

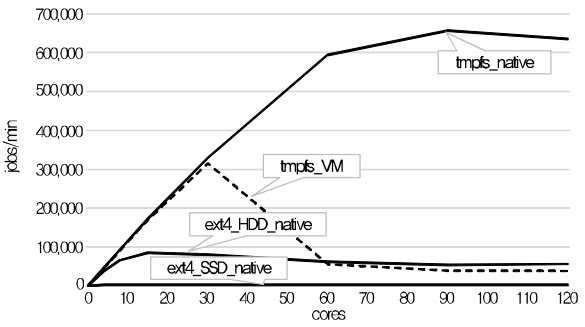
* DOI: 10.22648/ETRI.2017.J.320610

* 본 논문은 2017년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임[No. B0101-17-0644, 매니코어 기반 초고성능 스케일러블 OS 기초연구(차세대OS기초연구센터)].

Ⅰ. 머리말

매니코어 운영체제 연구는 대학 중심으로 진행되고 있는 초기 연구단계이다. 매니코어는 멀티코어보다 훨씬 많은 수백~수천 개의 프로세스 코어를 의미하며, 운영체제는 이러한 하드웨어 자원을 운영 관리하고 응용 소프트웨어 실행을 지원하는 시스템 소프트웨어이다. 운영체제 기술은 1970년대부터 개발되기 시작하여, 현재 거의 모든 기능이 성숙화된 분야이다. 하지만 매니코어 운영체제는 그렇지 못하다. 현재 리눅스 운영체제는 수십 코어에서 개발되고 있으며, 2016년 8월 리눅스 25주년 기념행사의 리누스 토발즈 인터뷰에서 SGI사는 매니코어를 지원하는 리눅스 커널 개발의 필요성을 언급한 바가 있다.

매니코어 시스템에서 리눅스의 스케일러빌러티 성능 실험 결과는 (그림 1)과 같다. 가상화 환경(점선), Native 환경의 여러 파일 시스템(실선) 등 모든 조건에서 리눅스가 스케일러블하지 않음을 알 수 있다. 따라서 차세대 OS기초연구센터 과제를 통해 이들 문제점을 분석하고 리눅스의 스케일러빌러티를 해결하는 요소기술들을 연구한다. 리눅스의 스케일러빌러티 연구뿐만 아니라 수천 개의 코어로 구성된 시스템에서 응용 서비스의 스케일러빌러티를 보장할 수 있는 새로운 운영체제를 연구하며, 과학계산 응용뿐만 아니라 빅데이터 및 딥러닝과 같은 데이터 집약적인 응용이 많은 코어 환경에서 효율적으로 동작되도록 한다.



(그림 1) 인텔 120코어에서 리눅스의 스케일러빌러티 성

Ⅱ. 기술 동향

1. 하드웨어 동향

프로세서는 2008년까지 회로의 집적도를 나타내는 트랜지스터의 수가 지속해서 증가함에 따라 동작 속도와 단일 쓰레드의 성능도 향상되었다. 그러나 2008년 이후, 발열 등의 이유로 단일 코어의 집적도를 더 높일 수 없는 물리적 한계에 다다랐다. 이에 코어의 수를 늘려 성능을 개선하는 멀티코어와 매니코어 프로세서 시스템 환경으로 변화하고 있다. 이러한 변화로 인해 Tilera, Kalray, 인텔 등 다양한 업체에서 매니코어 칩을 개발하고 있다.

Tilera사는 TileGx 칩을 개발하였는데 특히 TileGx72는 단일 칩에서 72코어를 지원한다. 개발된 프로세서는 EZChips에 인수되었고 지금은 멜라녹스에 인수되었다. Kalray사는 NoC(Network on Chip)로 연결된 클러스터 집합으로, 단일 칩에서 최대 256코어를 지원한다. ScaleMP사는 다수의 독립 서버를 인피니밴드(Infiniband) 인터커넥터로 연결하고 클러스터 시스템 소프트웨어로 통합한 vSMP 기술을 보유하고 있다. 이는 최대 128대의 표준 x86 시스템을 연결하여 3,276 코어와 256TB 공유 메모리를 지원하는 가상 시스템을 구성할 수 있다. NumaScale사는 2011년 멀티 프로세서를 가진 노드 4,096개를 연결하고 최대 256TB 메모리를 지원하는 NumaConnect SMP adapter를 출시하였다.

인텔은 올해 하반기에 70여 코어를 목표로 하는 Xeon Phi 프로세서 제품군인 KNL(Knights Landing), KNM(Knights Mill)을 제시하고 있고, 이후에도 지속적으로 코어를 확장한 제품을 출시하여 고성능 컴퓨팅을 지향하는 제품군의 로드맵을 제시하고 있다. 최근 ARM의 경우에도 96코어를 장착한 리눅스 기반의 서버를 발표했고, 오라클에서도 최대 64개의 SPARC 프로세서를 구축하여 1,024코어의 장착이 가능한 M10-4S 서버를 제공하고 있다.

2. 운영체제 동향

매니코어 시스템에서 코어 수에 비례하여 성능 확장성을 개선하는 운영체제 연구는 학계를 중심으로 진행중이다. 주로 매니코어 시스템의 구조적인 환경을 지원하기 위한 운영체제 구조 연구와 세부적으로는 성능 확장성을 보장하기 위한 락(Lock) 개선, 프로세서 캐쉬 일관성 기능의 성능 문제를 해결하고 코어의 캐쉬 효율 저하 등을 해결하기 위한 연구이다[1].

Corey[2] 커널은 기존 운영체제에서 공유가 필요 없는 내부 데이터까지 코어 간에 공유하여 시스템의 성능이 저하되는 문제점을 해결하고자 진행되었다. 이를 위해 응용이 직접 코어 간 공유를 제어하고 특정 코어를 임의의 OS 기능에 할당하기 위해 주소 공간, 커널 코어, 그리고 공유에 관련된 운영체제 추상화 계층을 제안하였다.

MIT의 FOS(Factored Operating System)[3]는 현재의 시분할 방식을 대체하는 공간분할 방식을 제시하였다. 기존 OS의 서비스를 여러 개로 분할한 후, 서비스 간에 메시지 전달을 하는 구조를 제안하였다. 마이크로 커널 기반의 메시지 전달로 공유 메커니즘을 삭제함으로써 성능 확장성을 향상시키는 구조이다.

IBM 왓슨의 Fused OS[4]는 이질적인 매니코어 환경에서 응용과 운영체제를 다른 코어에서 실행되도록 하여 응용의 성능이 운영체제에 의해 간섭받지 않도록 하였다. 운영체제 코어와 응용 코어의 분리로 운영체제와 응용이 함께 실행되는 환경보다 다른 코어에서 실행되는 응용의 성능이 우수하다는 것을 밝혔다. 하지만 응용에서 호출하는 시스템 호출은 원격의 운영체제 코어에서 처리되기 때문에 추가적인 호출시간이 필요하다.

멀티커널 운영체제인 Barrelfish[5]는 이종 하드웨어를 지원하고 각각의 코어마다 기본적인 커널이 탑재되어 사용자 공간에 OS 기능들이 존재하는 구조이다. Popcorn 리눅스[6]도 멀티커널 운영체제로, 이종 하드웨어 간에 응용이 이전(Migration)되면서 수행되도록 지원한다.

Ⅲ. 연구 현황

1. 모노리틱 커널 연구

모노리틱 커널 연구로는 인텔 120코어 시스템에서 리눅스 성능의 확장성을 보장하는 요소기술을 연구·개발하였다.

가. Lock-Free 알고리즘 연구

리눅스 커널은 업데이트 연산이 많이 발생하는 상황에서 확장성에 문제가 있다. 이러한 업데이트 비율이 높은 자료 구조를 위한 로그 기반의 동시적 업데이트 기법들이 그동안 연구되었다. 로그 기반 동시적 업데이트 기법 중 하나는 시스템에 전역 타임스탬프 카운터가 있다는 것을 가정하여 타임스탬프와 로그를 함께 저장하여 사용한 방법이 있다[7]. 이 방법은 업데이트 연산이 많을 경우 굉장히 높은 확장성을 가진다. 하지만 NUMA(Non Uniform Memory Acces) 구조를 가지는 매니코어 시스템에서는 아직 보장된 동기화된 전역 타임스탬프 카운터가 없는 것이 현실적인 문제이다. 아직 하드웨어적으로 지원되지 않는 동기화된 전역 타임스탬프 카운터 기법의 현실적인 문제점을 해결하기 위하여 새로운 동시적 업데이트를 위한 경량 로그 기반 지연 업데이트 방법인 LDU(Lightweight log-based Deferred Update)[8]가 개발되었다.

LDU는 타임스탬프 카운터가 필요한 연산을 하드웨어 동기화 기법을 사용하여 로그가 발생하는 순간 제거하고 불필요한 로그를 제거하는 방법이다. 이러한 LDU는 리눅스 커널 내부 자료 구조 중 높은 업데이트 비율 때문에 성능 확장성 문제를 야기하는 가상 메모리 시스템에 적용하였고, 이를 통해 확장성을 향상시켰다. 연구의

결과물을 리눅스 커널에 적용하였으며, 120코어 매니코어 시스템에서 확장성 벤치마크들을 대상으로 1.5배에서 2.7배까지 성능 향상을 이루었다. 하지만 이러한 LDU는 여전히 하드웨어 동기화 명령을 사용함으로 인한 문제가 있다.

향후 연구로는 소켓별로 동일한 클럭을 제공할 방법을 연구하여 동시적 업데이트 연산 때문에 발생하는 확장성 문제를 완전하게 제거하고자 한다.

나. 스케일러블 락 기법 연구

리눅스 커널에서 공유 데이터를 안전하게 접근하기 위해서 락 기법은 필수적인 기능이며 락 기법에는 스핀락(Spinlock)과 세마포락(Semaphore lock)이 있다. 이 기법으로 커널의 공유 데이터는 일관성 있게 관리가 되지만, 코어의 동작을 순서화하여 병목지점으로 인식되어 왔다. 이러한 문제로 적은 코어 환경에서 스케일러블 락 기법이 연구되었다. 현재 리눅스 커널은 초기에는 TAS lock를 사용하였고, 이후 티켓 스핀락(Ticket spinlock)과 큐 스핀락(Queue spinlock)으로 발전되었다. 스핀락 동작 중에 캐시 무효화 부담을 제거하는 방법으로 락 기법들이 발전되었다.

그렇지만 가상화 환경에서 리눅스 커널의 락 기법은 몇 가지 문제가 있었다. 대표적인 것이 하이퍼바이저와 게스트 운영체제 사이의 스케줄링의 불일치 문제이다. 예를 들면, 가상 코어에서 동작하는 락 소유자가 물리 코어를 반환하는 경우 또는 락을 얻을 수 있는 락 대기자가 물리 코어를 할당받지 못해 수행되지 않는 현상으로, 이러한 현상은 매니코어 시스템에서 더욱 자주 발생하여 심각한 문제가 되었다. 따라서 이러한 문제를 해결하는 기회 티켓 스핀락(Opportunistic ticket spinlock) 연구하였다[9], [10]. 과제 수행 중에 리눅스 커널의 락은 티켓 스핀락에서 큐 스핀락으로 변경되었고, 큐 스핀락에서도 같은 문제가 발생하여 락 기법을 개선하였다. 개선된 기법은 120 코어까지 성능의 스케일러빌러티를 보장하며, Gmake 벤치마크에서 기존 기법보다 3배의 성능 향상이 있었다.

가상화 환경에서 또 다른 문제는 오버 커밋 성능 문제였다. 2개 이상의 가상 머신들이 코어 자원을 나누어서 사용할 때, 성능이 자원을 나누어서 사용하는 것보다 더 많이 떨어졌다. 이 오버 커밋 문제도 티켓 스핀락과 큐 스핀락에서 해결하였다.

매니코어에서 스케일러블 락 기법 연구를 위해, 추가적으로 고려해야 할 사항은 NUMA effect이다. 많은 응용이 NUMA effect를 고려하여 소프트웨어를 만들지만, 운영체제는 그렇지 못하였다. 따라서 NUMA 구조를 인식하는 세마포락(CST-lock)을 연구 개발하였다[11]. CST-lock은 NUMA effect를 고려한 계층적인 구조이며, CPU 소켓마다 대기 자료구조를 두어 각각 대기지점에서 락을 대기하도록 한다. 임의의 소켓에서 락을 얻으면 해당 소켓의 모든 쓰레드가 락을 사용하고 다른 소켓으로 넘겨, 캐시 무효화 부담이 없어 스케일러블한 성능을 제공한다.

다. 스케줄링 연구

리눅스의 기본 스케줄러인 CFS(Completely Fair Scheduler) 스케줄러는 태스크 간의 공정성(Fairness)을 유지하기 위한 많은 특징을 가지고 있다. 그러나 매니코어 환경은 충분한 코어수로 인해 공정성을 유지하기 위한 작업들이 불필요하다. 특히 성능을 중요시하는 고성능 컴퓨팅 환경에서는 CFS 스케줄러의 무거운 동작이 성능 저하의 원인이 된다.

FLSCHED(Feather-Like Scheduler)[12]는 기존의 리눅스 라운드로빈 스케줄러 기반의 경량 스케줄러로, 공정성 유지에 필요한 연산을 최소화한 스케줄러이다. Xeon Phi 프로세서에 최적화하여 구현되었고, 병렬 프로그래밍 벤치마크 NAS Parallel Benchmark 결과로 최대 1.73배의 뛰어난 성능을 보였다.

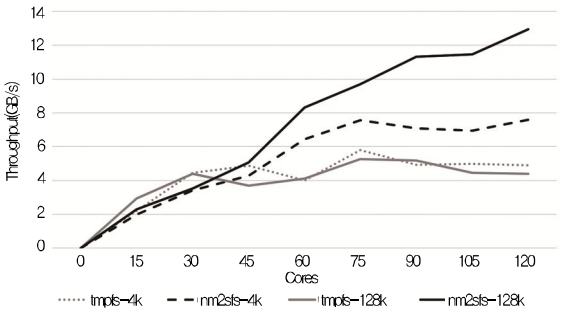
라. 스케일러블 파일시스템 연구

매니코어에서 파일시스템의 스케일러빌러티는 가장 심각한 문제였다. 먼저 SSD 캐싱 방법으로 이 문제를 해결하려고 하였다. 오픈 소스 EnhanceIO[13]에 캐시 히트율을 효과적으로 관리하는 기법의 알고리즘[14]를 연구하여 스케일러빌러티 실험을 진행하였다. 하지만 SSD 캐싱이 이 문제를 해결하지 못함을 확인하였다.

두 번째 방법으로 파일시스템 자체를 개선하는 것으로, 먼저 파일시스템의 문제점을 분석하였다. 하드디스크 드라이브, 솔리드스테이트(Solid-state)드라이브, DRAM 등 다양한 종류의 저장장치를 사용하여 tmpfs, ext4, XFS, btrfs, F2FS 파일시스템의 성능 분석으로, 기존 리눅스 파일시스템의 문제점을 발견하였다[15].

발견된 문제들은 다음과 같다. 저장 장치의 빠르고 느린 특성보다는 페이지 캐시 관리를 위해 사용하는 참조 카운트의 오버헤드로 인한 병목 문제, F2FS와 btrfs 같은 파일시스템에서 일관성 유지 메커니즘의 비효율성 문제, 파일 쓰기 과정에서 동시에 하나의 파일을 업데이트할 수 없는 구조의 락 사용 문제, 세부적이지 못한 락 사용 등이 파일시스템을 스케일러블하지 않게 하는 주요 원인임을 확인하였다[15].

스케일러블 파일시스템을 연구 개발하는데 먼저 메모리 저장 장치를 사용하는 방법을 선택하였다[16]. 메모리 저장 장치를 사용함으로써 페이지 캐시 및 관련 락 등 파일시스템의 많은 코드를 사용할 필요가 없게 되었다. 또한, NUMA effect와 저장 장치 위치를 고려한 파일 데이터 등을 관리하는 기법들로, 확장성 있는 고성능 파일시스템(nm2sfs: non-volatile memory storage to the scalable file system)을 개발하였다. 120 코어에서 실험한 경우, (그림 2)와 같이 리눅스 tmpfs 파일시스템보다 스케일러블하고 높은 성능 결과를 얻을 수 있음을 확인하였다.



(그림 2) 메모리 저장 장치기반 파일시스템 성능

마. 매니코어 하드웨어 특성 지원 연구

고속 네트워크의 대역폭은 급격히 증가하고 있으며, 현재 100Gbps에 이르고 있다. 반면, 프로세서는 발열 및 전력 문제로 인해서 처리속도의 증가는 더 이상 무어의 법칙을 따르지 않고, 코어의 개수를 증가시키는 방향으로 발전하고 있다. 따라서 매니코어 환경에서 급격히 증가하는 네트워크 대역폭을 효율적으로 활용하기 위한 방안이 필요하다.

본 연구에서는 프로세스, 시스템 호출, 인터럽트의 코어 친화도에 따른 네트워크 I/O의 성능 차이를 관찰하고, 네트워크 I/O 성능 향상을 위한 동적 코어 친화도 기법을 제안하였다[17]-[19]. 제안된 기법은 매니코어 시스템의 메모리 계층 구조와 현재 시스템 부하를 고려하여 코어 친화도를 동적으로 결정하며, 응용을 투명하게 적용 가능하다. 결과적으로 캐쉬 일관성을 위한 오버헤드와 캐쉬 오염을 낮추고 네트워크 연결의 지역성을 높일 수 있다.

제안된 기법은 HDFS(Hadoop Distributed File Systems), Apache 웹 서버, Memcached 등에 적용되어 응용 수준의 처리율 향상을 보였다. 예로서, Apache 웹 서버 성능을 최대 50% 향상시켰음을 확인하였다.

SPM(Scratchpad Memory)은 캐시와 같은 온 칩 메모리로서 소프트웨어적으로 제어되는 메모리이다. 두 개 이상의 코어를 가진 캐시 기반 시스템에서는 코어 간 캐시 내의 데이터를 동일하게 유지하도록 하는 캐시 일관성 기술이 사용되고 있다. 수 개의 코어를 가진 멀티코

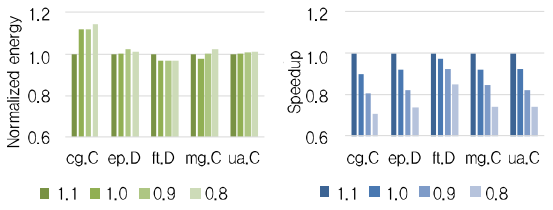
어 환경에서 이 기술은 적은 비용을 사용하여 자동으로 데이터 일관성을 유지하는 편의를 제공했지만, 매니코어 환경에서는 그 비용이 과도해짐에 따라 시스템의 확장성을 저해하는 요소로 지목되고 있다. 이에 따라 캐시 일관성 문제가 없는 SPM으로 캐시를 대체하는 방법이 다양하게 연구되고 있다.

캐시 대신 SPM을 사용할 경우, CPU가 사용하는 데이터를 명시적으로 메모리에서 SPM으로 불러와야 한다는 어려움이 존재한다. 이를 해결하기 위하여 컴파일러가 데이터 이동 코드를 자동적으로 생성하도록 하는 연구들이 진행되었다[20]. 또한, 관리 코드를 삽입할 때 특정 함수를 작은 함수들로 분할하는 최적화를 통해 성능을 향상시키는 등 소프트웨어적 관리 기법이 연구되었다[21].

바. 에너지 효율화 연구

매니코어 프로세서는 코어의 수를 늘리기 위해, 멀티코어 프로세서에 비해 낮은 주파수에서 동작하며, 조절 가능한 주파수 범위도 좁아졌다. 또한, 최신의 프로세서들은 에너지 효율성을 우선으로 고려하여 설계되었다.

이러한 매니코어 프로세서에서 응용을 실행할 때 (그림 3)과 같이 주파수를 낮춰도 줄어드는 에너지 소모보다 성능 저하가 더 심한 현상을 볼 수 있다. 이러한 변화는 응용의 자원 요구를 파악하고 적절한 성능을 선택하는 DVFS(Dynamic Voltage and Frequency Scaling)를 기반으로 하는 기존의 에너지 효율성 개선 기술을 무의미하게 만들었다.



(그림 3) 매니코어에서 주파수 변화에 의한 응용의 소모 에너지와 성능 비교 그래프

따라서 매니코어 운영체제는 에너지를 절감하기 위해 기존과 다른 방향에서의 접근이 필요하다. 최근 에너지 효율 개선 연구들은 프로세서의 처리량을 더욱 높이는 것에 우선순위로 두고, 그 이외 낭비되는 자원을 줄이기 위해 노력하는 경향을 보인다.

특히, 동기화 기법에서 사용되는 바쁜 대기의 에너지 낭비 문제는 매니코어 운영체제를 위해 반드시 해결되어야 할 문제이다. 바쁜 대기는 코어 수 증가로 얻을 수 있는 처리량의 증가보다 에너지 소모를 더 많이 증가시키는 원인이 된다. 리눅스 커널의 스핀락과 Select 시스템 호출 등에서 바쁜 대기로 인한 에너지 낭비 문제가 발견되었고, 이를 해결하기 위한 연구가 진행되었다. 또한, 매니코어 시스템은 열 집적도가 높아 냉각이 차지하는 에너지 비용을 줄이는 것이 중요하다[22]. 무 냉각(Free cooling) 데이터 센터와 같이 가혹한 환경에서도 온도를 적절하게 제한하면서 높은 우선순위의 작업에 적절한 성능을 제공할 수 있는 연구도 진행되었다[23].

사. 도구

연구개발 과정에서 부산물로 다음의 도구도 개발되었다. Juxta[24]는 파일 시스템의 성능 병목지점을 분석하는 과정에서 만들어진 도구로, 기존의 소스코드 구현물을 비교하고 대조하여 시맨틱 버그를 찾아내는 도구이다. APIsan[25]은 Juxta 도구와 유사한 개념의 도구이며, 올바른 API 사용 여부를 비교와 대조하여 버그를 찾는데 활용된다. FxMark[15]는 파일 시스템의 성능 확장성을 시험하기 위해서 만들어진 도구이다. Ivy Profiler[26]는 병렬 프레임워크 실험 과정에서 만들어진 도구로, 멀티/매니코어에서 병목 지점, 소비 전력, 캐시 미스 등 사용자가 선택한 프로파일링 정보를 제공한다.

아. 리눅스 커뮤니티 기여

리눅스 스케일러빌리티 연구 과정에서 만들어진

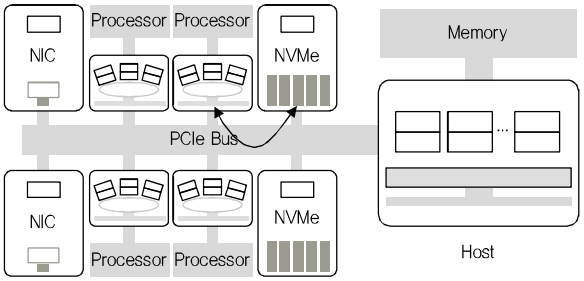
Juxta[24], APIsan[25] 도구로 리눅스 버그 패치 202개를 제출하였다. 또한, 가상화 환경에서 리눅스 티켓 스핀락의 스케일러빌러티 문제를 개선하는 기능 패치를 제출하였고[27], 이 기법은 리눅스 큐 스핀락 개발에 반영되었다. 오버커밋 워크로드에서 큐 스핀락의 성능 문제를 개선하는 기능 패치도 제출되었다.

2. 멀티커널 연구

매니코어 시스템은 수백에서 수천 개의 코어로 구성된 시스템이다. 모노리틱 커널은 모든 기능이 하나의 커널에 있고, 많은 수의 코어가 커널 기능을 수행하는 경우, 상호 간의 간섭으로 코어 수 대비 확장성을 갖지 못한다. 따라서, 기능에 따라 여러 개의 커널로 나누어 구성하는 멀티커널이 제시되었으며, CNK[28], McKernel[29], Popcorn Linux[6] 등 멀티커널에 대한 연구들이 진행되고 있다.

멀티커널 연구를 위해 먼저 매니코어 시스템 테스트베드 구축을 선행하였다. 본 연구에서는 Xeon과 여러 개의 Xeon Phi로 코어를 구성하고 빠른 입출력을 위해 NVMe(Non-Volatile Memory express)를 활용하여 1,000 코어 매니코어 테스트베드를 (그림 4)와 같이 구축하였다. 현재 Xeon Phi는 Knights Corner(55 코어 이상)를 사용하였으며 추후 Knights Landing(64 코어 이상), Knights Mill 등의 사용하여 더 많은 코어를 구성할 수 있다.

구성한 테스트베드에서 코어간 통신 지연시간, 캐시



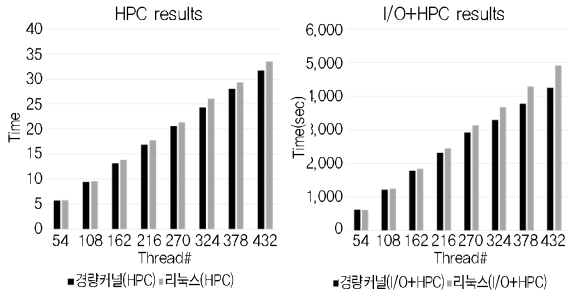
(그림 4) 매니코어 시스템 테스트베드

및 메모리 접근시간에 대한 실험 결과, Xeon Phi의 프로세서가 SMP(Symmetric Multi-Processing) UMA(Uniform Memory Access)로 동작하는 Xeon 프로세서와 유사하게 동작하는 것을 확인하였다. 또한, 멀티커널에서 Mosaic 그래픽 엔진[30] 응용을 실험한 결과, 코어 수 대비 확장성을 지원함을 확인하였다.

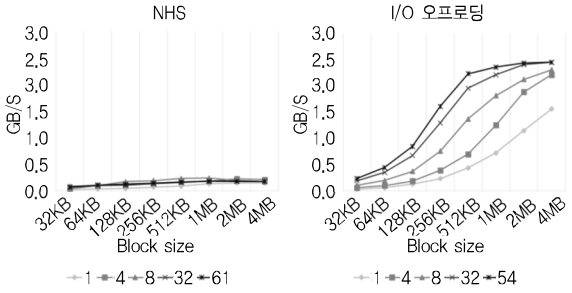
고성능 컴퓨팅 시스템에서 성능의 최대화를 위해 운영체제와 응용 프로그램의 최적화가 필요하다. 이를 위해 고성능 시스템은 연산을 주로 수행하는 계산 노드와 이를 지원하기 위한 서비스 노드로 구성된다. 계산 노드에는 경량커널 LWK(Light Weight Kernel)이 활용되며, 이는 Xeon Phi의 많은 코어에서 동작하며 운영체제의 간섭을 최소화하는 특징을 가진다. 서비스 노드에는 범용으로 사용이 가능한 FWK(Full Weight Kernel)이 활용되며, Xeon의 고성능 코어에서 동작한다.

경량커널은 기존 리눅스를 활용하는 방식과 새로운 경량커널을 만들어 적용하는 방식으로 연구가 진행되고 있다. 리눅스로 구성하는 경우, 리눅스의 풍부한 기능들을 활용하여 기존의 응용들까지도 쉽게 실행시킬 수 있다는 장점이 있지만, 커널의 간섭 때문에 성능이 제한되는 단점을 가진다. 반면에 새로운 경량커널로 구성하는 경우, 커널의 간섭 없이 응용프로그램들이 코어를 최대한으로 활용할 수 있는 장점이 있는 반면, 디바이스 드라이버 등 리눅스로 구성했을 때 가지는 풍부한 기능을 지원하지 못하는 단점을 가진다.

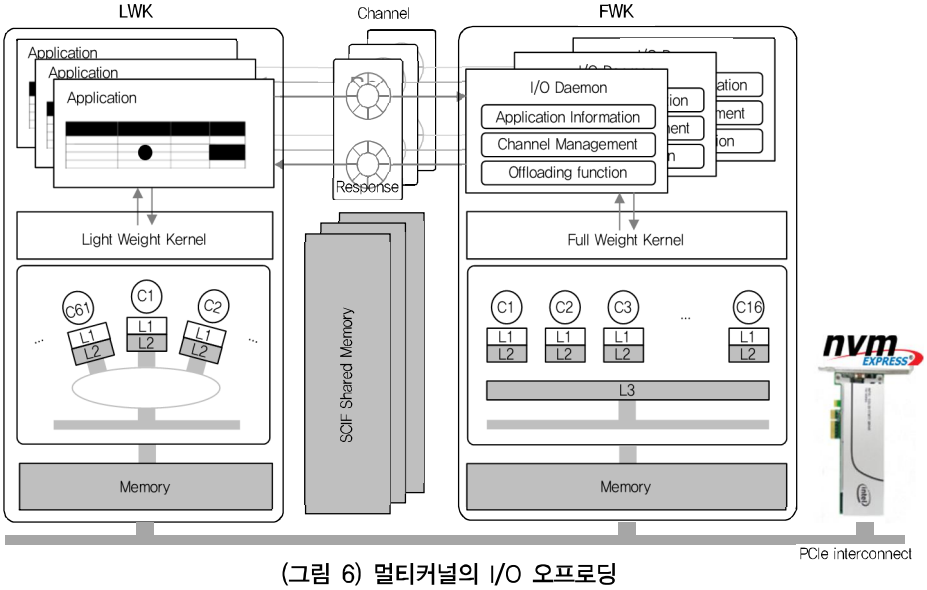
경량커널에서 코어의 사용률을 높이고 확장성을 보장하기 위해 모노리틱 커널에서 주로 사용된 시분할(Time sharing) 기법 대신 공간분할(Space sharing) 기법을 적용하였다. 공간분할 기법은 첫 번째로 응용 프로그램이 코어를 독점하여 사용할 수 있도록 코어가 관리된다. 이와 같이 관리되어 할당된 코어에서 동작하는 응용프로그램은 타이머 인터럽트 등 커널의 간섭 없이 종료할 때까지 수행이 보장된다. 두 번째로는 사용할 수 있는 특



(그림 5) 경량커널(공간분할)과 리눅스(시분할) 성능 비교



(그림 7) NFS와 I/O 오프로딩 성능 비교



(그림 6) 멀티커널의 I/O 오프로딩

정 메모리 영역을 지정함으로써, 메모리를 할당/반환하는 경우 커널의 페이지 테이블 연산에서 공유되는 자료구조가 없기 때문에 간섭이 없게 된다. 커널 간섭을 최소화함으로써 응용 프로그램은 코어를 충분히 활용할 수 있다. (그림 5)에서 HPC 응용 프로그램에 대해 코어가 많아질수록 간섭이 적은 공간분할 기법이 시분할 기법보다 좋은 성능을 나타냄을 확인하였다.

경량커널에서 응용의 동작 중에 발생되는 I/O 요청은 서비스 노드에 전달하여 처리하는 I/O 오프로딩(Offloading) 방식으로 처리된다. 멀티커널에서 I/O 오프로딩을 위한 경량커널과 FWK의 구조, 그리고 메시지 전달을 위한 채널은 (그림 6)과 같이 구성된다.

멀티커널의 I/O 오프로딩으로 서비스 노드는 NVMe에 저장된 데이터를 계산 노드의 응용에서 DMA(Direct Memory Access) 액세스[1]할 수 있도록 하여, 고속 입출력 기능을 제공한다. 이 기능은 대규모 데이터를 처리하는 응용에 적합하다. (그림 7)에서 리눅스의 NFS (Network File System) 기법은 데이터 크기와 코어 수 변화에 따른 성능이 낮게 나타나는 것에 반해, 멀티커널의 I/O 오프로딩은 데이터 크기가 증가하고 코어 수가 증가함에 따라 NVMe의 최대 성능까지 나타내고 있다.

□) "A Data-Centric Operating System Architecture for Heterogeneous computing," Eurosys에 논문을 제출함

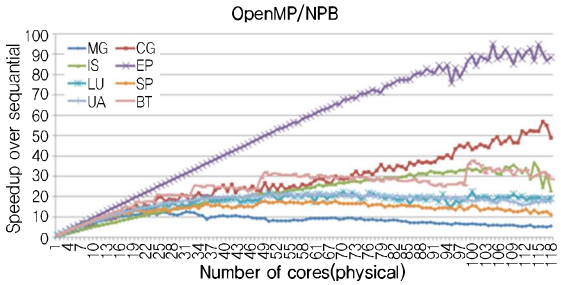
3. 병렬화 연구

가. 병렬 프로그래밍 프레임워크 연구

멀티코어 환경에서 병렬 프로그래밍을 위해 Pthreads, OpenMP, MPI, OpenCL등의 프레임워크가 널리 사용된다. 향후 수백 개 이상의 코어로 구성된 매니코어 환경에서 확장성 있는 성능을 달성하기 위해서는, 다양한 프레임워크에서의 성능 병목 분석과 이를 용이하게 하는 경량 성능 분석도구의 개발이 필수적이다.

(그림 8)은 인텔 Xeon E7-8870v2 8소켓 120 코어 플랫폼에서 OpenMP 기반 NAS Parallel Benchmark (OpenMP-NPB)의 성능 확장성을 평가한 결과이다. 이상적인 경우, 사용하는 코어 수의 증가에 따라 EP와 같은 선형적인 성능 증가가 나타나야 하지만, 다른 프로그램들은 비 확장적인(Non-scalable) 성능 특성을 보인다. 예를 들어, 일정 코어 수 이상에서 성능향상이 정체되거나(LU, UA) 감소하는 형태(IS, MG, SP), 또는 계단/톱니바퀴 모양의 성능 곡선(BG, CG) 등을 확인할 수 있다. 이는 알고리즘 자체의 병렬성이 제한적이거나, 메모리 또는 캐쉬 등 공유 자원의 충돌, 쓰레드 간의 로드 불균형, 비효율적인 I/O처리 등에 기인한다.

이러한 성능 확장성 병목 분석을 위해 최근 하드웨어에서 제공하는 성능 모니터링 카운터를 활용한 병목 분석 툴이 폭넓게 사용되고 있으며, 리눅스 perf, 인텔 VTune 등이 잘 알려져 있다. 본 연구에서는 함수 단위의 성능 결과를 효율적으로 시각화해 주는 경량 성능 분석 도구[26]를 개발하였으며, 성능 병목 지점 추출 및



(그림 8) OpenMP-NPB 워크로드의 성능 확장성 결과

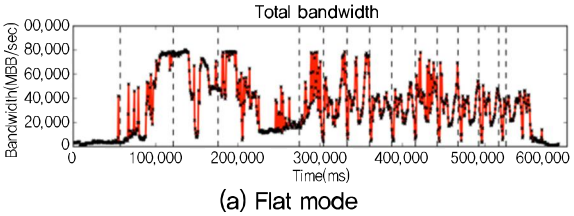
해소 방안에 대한 연구를 진행 중이다

나. 스케일러블 자바 가상 머신 연구

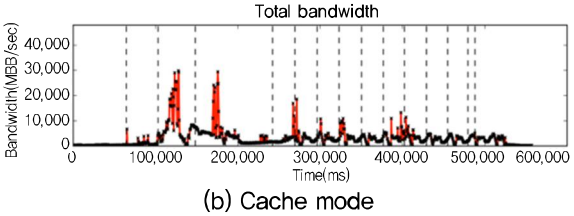
최근 높은 대역폭 성능을 가진 3D 층 DRAM의 채택이 활발히 이루어지고 있다. 예를 들어, 인텔 Knights Landing 매니코어 플랫폼은 CPU 패키지 안에 높은 대역폭을 갖는, 최대 16GB의 MCDRAM을 장착하고 있어, 기존의 DDR4 DRAM과 함께 Near-far 메모리를 구성하고 있다. Near 메모리는 하드웨어가 관리하는 대용량의 DRAM 캐쉬로 활용하거나(Cache 모드), 소프트웨어가 관리하는 물리 메모리의 일부로 사용(Flat 모드)할 수 있다. 그러나, 워크로드에 따른 최적 메모리 할당 및 활용 방법에 대한 연구는 여전히 진행 중이다.

최근 널리 채택되고 있는 Apache Spark 등의 상당수의 빅데이터 처리 엔진은 자바 가상 머신(JVM)위에서 동작하는데, Near-far 메모리 기반의 매니코어 플랫폼에서 JVM의 성능 확장성을 제공하는 것은 매우 중요한 문제이다. (그림 9)는 인텔 Knights Landing(Xeon Phi 7210) 플랫폼에서 Spark PageRank 어플리케이션의 모드 별 실행시간 및 메모리 처리량을 비교한 그래프이다.

Near 메모리를 사용하지 않는 Flat 모드의 경우, 특정한 Stage에서 대역폭 수치가 포화상태(DDR4, 약 80GB/sec)에 이르지만, Cache 모드의 경우 Near 메모



(a) Flat mode



(b) Cache mode

(그림 9) Spark PageRank 응용의 메모리 처리량 비교

리(MCDRAM, 최대 480GB/s 대역폭 성능)의 활용을 통해 짧은 시간 동안 높은 대역폭 수치를 활용하여 포화상태를 일부 해소하며, 이에 따른 성능 차이는 61.7%에 이른다.

그러나, Cache 모드의 사용을 통해서는 워크로드의 대역폭 사용특성을 반영하는 세밀한 할당이 어려우며, 또한 MCDRAM의 경우, 일반적으로 지연시간(Latency) 성능은 DDR4에 비해 나쁜 것으로 알려져 있어 워크로드의 저 대역폭 사용 구간에서는 오히려 성능 하락을 일으킨다. 따라서 Flat 모드상에서 워크로드 대역폭 사용특성을 반영하여, MCDRAM 영역을 효율적으로 활용하는 메모리 관리기법이 요구된다.

이를 위해, 본 연구에서는 매니코어 환경에서 빅데이터 프레임워크의 Near-far 메모리 환경에서의 성능 확장성 제공을 위해, JVM 힙 메모리 분할 관리 기술, 스케일러블한 Garbage Collection(GC), Spark 수행 단위(Stage)별 메모리 대역폭 예측 기법을 적용하는 스케일러블 JVM 연구를 진행하였다[31].

다. 이기종 매니코어 기반 그래프 프로세싱 엔진 연구

인터넷이 시작되면서 웹 그래프 또는 소셜 네트워크로 구성된 대규모 그래프가 보편화 되었다. 예를 들어 최근 최대 규모의 소셜 그래프는 페이스북으로, 14억 개의 정점과 1조 개의 에지 규모를 가진다. 일반적으로 이러한 대형 그래프 처리를 위해, 200대의 컴퓨터에 분산 그래프 엔진(Giraph[32])을 사용하지만, 단일 시스템에서 대형 그래프를 처리할 수 있는 그래프 엔진 Mosaic[30]를 개발하였다. 단일 시스템은 인텔 Xeon phi매니코어와 NVMe 스토리지로 구성된 저가 시스템이며, Ⅲ장 2절에서 소개한 멀티 커널 운영체제가 동작한다.

Mosaic은 저가의 시스템을 이용하지만, 다른 최첨단 코어에서 동작하는 분산 그래프 엔진보다 3.2배~58.6배 높은 성능을 발휘한다. 1조 개 에지 규모의 그래프에서 Mosaic은 다른 엔진보다 9.2배 빠르게 PageRank 알고리즘을 21분 만에 처리했다. 단일 시스템 엔진의 또 다른 장점은 장애허용(Fault tolerance) 접근 방식이 간단하다는 것이다. 분산 시스템 엔진은 다수의 노드 간 전역 상태의 일관성을 위해 동기화 프로토콜을 사용하는 것에 비해 단일 시스템은 중간 상태 데이터를 체크포인팅만 하면 된다.

성능 관점에서 Mosaic의 독창성은 대규모 그래프를 적절한 파티션 크기로 분할하고, 이 파티션을 캐시 지역성을 높이기 위해서 힐버트 순서로 탐색하여 서로 가까이 있는 정점들로 타일을 구성하는 것이다. 이 타일은 서브 그래프가 되며, 타일 크기는 LLC(Last Level Cache) 크기에 종속된다. 타일들은 NVMe로부터 빠르게 제공되며, Phi 코어에서 병렬 처리되어 로컬 정점 상태를 계산한다. 이렇게 병렬 처리되는 서브 그래프들의 결과는 Xeon 코어에서 글로벌 정점 상태로 계산된다.

Mosaic은 Gather-Apply-Scatter 모델과 유사한 Pull-Reduce-Apply 프로그래밍 모델을 제공한다. Pull은 타일로 나누어진 서브 그래프를 처리하는 API이며, Reduce는 같은 정점에 대해 두 개의 값으로 하나로 결합하는 글로벌 그래프 정점 상태 업데이트를 위한 API이다. Apply API는 각 정점에 대해 그래프 알고리즘을 수행하는 데 사용된다.

라. 분산 하스켈 연구

하스켈(Haskell)은 다양한 병렬 프로그래밍 모델을 제공하고 있다. 대표적인 하스켈의 병렬 프로그래밍 모델로 Eval 모나드(Monad)와 Cloud Haskell을 들 수 있다. 매니코어 환경에 적합한 병렬 프로그래밍 모델을 찾기 위해 두 모델을 적용하여 병렬 프로그램을 작성하고 실험해 본 결과, Cloud Haskell이 적합한 것으로 나타났다[33]. Eval 모나드는 개발 편의성은 높지만, 코어 수가 늘어남에 따라 확장성이 확보되지 못하였는데, 이는

GC로 인한 문제로 판단된다[33]. Cloud Haskell은 확장성은 뛰어나지만 Eval 모나드에 비해 개발이 어려운 문제점이 있다.

매니코어 환경에서 Cloud Haskell의 낮은 개발 편의성은 프로그래머에게 진입 장벽이 될 수 있다. 이를 해결하기 위해서는 Cloud Haskell의 개발 편의성을 높여야 하는데, 메타 프로그래밍이 이에 대한 해결책이 될 수 있다. 구체적으로 Template Haskell을 이용하여 라이브러리를 제공함으로써 개발 편의성을 개선할 수 있다. 현재까지의 연구에 따르면 Template Haskell 라이브러리를 통해 병렬 함수 코드 크기를 약 66%가량 줄일 수 있음을 확인할 수 있었다[34].

마. 매니코어 기반 딥러닝 병렬화 연구

기계 학습을 활용한 인공 지능 연구는 2010년 이후 매니코어 프로세서가 제공하는 병렬성을 활용하여 본격적으로 실용화가 시작되었다. 모든 기계 학습 알고리즘의 중심에는 다차원 행렬 간의 반복적인 연산이 있으므로 이 연산을 효율적으로 수행하는데 SIMD(Single Instruction Multiple Data) 연산 기능과 조건에 따른 분기가 가능한 SPMD(Single Program Multiple Data) 매니코어 연산장치가 중요한 역할을 한다. NVIDIA GPU와 인텔 Xeon Phi, 구글 TPU와 같은 매니코어 프로세서들이 텐서 연산의 가속을 위해 사용되고 있으며 이들을 활용성을 높이기 위한 소프트웨어 개발이 또한 필수적이다. 동시에, 개발되는 하드웨어/소프트웨어 도구들을 연구자/개발자들이 쉽게 적용해 볼 수 있는 프레임워크의 역할도 중요하다. 2015년 구글이 발표한 TensorFlow가 대표적인 예이다.

본 과제에서는 연구 개발 단계의 병렬화된 딥러닝 알고리즘을 적용하고 성능을 비교할 수 있는 Ktune 프레임워크를 개발하여 매니코어를 이용한 딥러닝 연산의 병렬화 성능 실험[35]을 수행하고 있다.

Ⅳ. 향후 계획

과제 2단계(2018-2021)는 다음과 같은 연구 활동을 계획하고 있다. 첫째, 락(Lock) 기법을 사용하지 않고 공유 데이터를 동기화할 수 있는 새로운 기법을 연구하고자 한다. Lock-free 기법 등 기존 기술과의 차이점은 여러 자료구조에 적용하는 점이다. 커널에서 락 기법으로 여러 자료구조를 동시에 접근하는 사례가 많다. 또한, 이 연구의 필요성은, 과제 1단계에서 스케일러블 락 기법을 연구하였지만, 120 코어 이상의 더 많은 코어 환경에서 락 기법을 사용하지 않고 새로운 동기화 기법이 스케일러빌러티에 더욱 좋을 것으로 기대하기 때문이다.

둘째, 스케일러블 파일 시스템 개발이다. 기존의 리눅스 파일 시스템에서 병목 요인인 저널링, 락 방식 등을 해결하고자 한다. 또한 메모리 기반 스토리지 디바이스용 파일 시스템도 계속 연구한다.

셋째는 스케일러블 네트워크 스택 연구로, 100Gbps 네트워크 디바이스를 지원하는 연구이다. 패킷 수신부터 응용까지 커널 병목요인을 해결하여 저 지연, 높은 대역폭의 네트워크 서비스를 제공하는 요소기술을 연구한다. 이들 연구는 최대 192 코어 시스템에서 진행될 예정이다.

넷째, 멀티커널 운영체제 연구이다. 인텔 Knight Landing 하드웨어를 사용하여 1,000 코어 이상의 시스템에서 진행될 예정이다. 모노리틱 커널 대비, 데이터 집약적인 응용이 높은 성능을 낼 수 있도록 하며, 딥러닝 응용에 집중할 계획이다. 이러한 1,000 코어 이상 시스템의 에너지 제어 기법도 연구된다.

Ⅴ. 맺음말

매니코어 하드웨어 추세는 최근 들어 더욱 뚜렷한 양상을 보이고 있다. 이에 반해 운영체제는 아직까지 멀티코어 지원 수준에 머물러 있어 코어 수에 따른 성능 확장성에 문제를 보이며, 이로 인해 응용은 하드웨어의 성

능을 충분히 활용하지 못하고 있다.

본고에서는 이러한 문제를 다루고 있고 '매니코어 기반 초고성능 스케일러블 OS 기초 연구' 과제의 연구 활동을 소개하였다. 과제 1단계(2014-2017)에서 연구된 요소기술들은 https://github.com/oslab-swrc에 공개되어 있다. 공개된 기술은 연구용 등 다양한 목적으로 소스코드를 활용할 수 있다. 소스코드 활용뿐만 아니라, 매니코어 운영체제 연구 과정을 통해 발견된 새로운 기법 등 아이디어들을 다양한 분야에서 활용되길 기대해 본다.

용어해설

매니코어 빠른 동작속도보다 다수의 단순하고 독립적인 프로세서코어 (예: 10, 100 또는 1,000)를 포함하는 높은 수준의 병렬처리를 위해 설계된 프로세서

스케일러빌리티(확장성) 프로세서의 코어 수 증가에 따라 소프트웨어 성능이 비례적으로 증가하는 현상

커널 컴퓨터 하드웨어 자원을 운영 관리하며, 응용 소프트웨어를 동작시키는 시스템 소프트웨어. 예로 리눅스, 윈도우 등이 있으며, 구조에 따라 모노리틱 커널, 마이크로 커널, 멀티 커널로 구분됨

약어 정리

CFS	Completely Fair Scheduler
DMA	Direct Memory Access
DVFS	Dynamic Voltage and Frequency Scaling
FLSCHED	Feather-Like scheduler
FOS	Factored Operating System
FWK	Full Weight Kernel
GC	Garbage Collection
HDFS	Hadoop Distributed File Systems
KNL	Knights Landing
KNM	Knights Mill
LDU	Lightweight log-based Deferred Update
LLC	Last Level Cache
LWK	Light Weight Kernel
NFS	Network File System
NoC	Network on Chip
NUMA	Non Uniform Memory Access
NVMe	Non-Volatile Memory express
SIMD	Single Instruction Multiple Data
SMP	Symmetric Multi-Processing
SPM	Scratchpad Memory
SPMD	Single Program Multiple Data
UMA	Uniform Memory Access

참고문헌

[1] 정진환, 김강호, 김진미, 정성인, "Manycore 운영체제 동향," 전자통신동향분석, 제29권 제5호, 2014. 10. 1, pp. 176-185.

[2] B.-W. Silas et al., "Corey: An Operating System for Many Cores," *Symp. Operat. Syst. Des. Implement.*, San Diego, CA, USA, Dec. 8-10, 2008, pp. 43-57.

[3] D. Wentzlaff and A. Agarwal "Factored operating systems (fos): The Case for a Scalable Operating System for Multicores," *ACM SIGOPS Operating Syst. Rev.*, vol. 43, no. 2, Apr. 2009, pp. 76-85.

[4] Y.H. Park et al., "FusedOS: Fusing LWK Performance with FWK Functionality in a Heterogeneous Environment," *IEEE Int. Symp. Comput. Archit. High Perform. Comput.*, New York, USA, Oct. 2012, pp. 211-218.

[5] A. Baumann et al., " The Multikernel: a New OS Architecture for Scalable Multicore Systems," *Symp. Operating Syst. Principles*, Big Sky, MT, USA, Oct. 11-14, 2009, pp. 29-44

[6] A. Barbalce, B.Ravindran and D. Katz, " Popcorn: a Replicated-Kernel OS Based on Linux," *Proc. Linux Symp.*, Ottawa, Canada, July 14-16, 2014, pp. 123-138

[7] Silas Boyd-Wickizer et al., "OpLog: a Library for Scaling Update-Heavy Data Structures," Technical Report MIT-CSAIL-TR2014-019, 2014.

[8] J. Kyong and S.-S. Lim, "LDU: A Lightweight Concurrent Update Method with Deferred Processing for Linux Kernel Scalability," In *Proc. IASTED Int. Conf., Parallel Distribut. Comput. Netw.*, Innsbruck, Austra, Feb. 15-16, 2016.

[9] S. Kashyap, C. Min, and T. Kim, "Opportunistic Spinlocks: Achieving Virtual Machine Scalability in the Clouds," *ACM SIGOPS Operat. Syst. Rev.*, vol. 50, no. 1, Jan. 2016, pp. 9-16.

[10] S. Kashyap, C. Min, and T. Kim, "Scalability in the Clouds!: a Myth or Reality?" *Proc. Asia-Pacific Workshop Syst.*, Tokyo, Japan, July 27-28, 2015, pp. 1-7.

[11] S. Kashyap, C. Min, and T. Kim, "Scalable NUMA-Aware Blocking Synchronization Primitives," In *Proc. USENIX*

Annu. Tech. Conf., Santa Clara, CA, USA, July 12–14, 2017, pp. 603–615.

[12] H. Jo et al., "A Lockless and Lightweight Approach to OS Scheduler for Xeon Phi," *Proc. Asia-Pacific Worshop Syst.*, Munbai, India, Sept. 2, 2017.

[13] STEC, EnhanceIO SSD Caching Software, Accessed 2017. https://github.com/stec-inc/EnhanceIO

[14] 허상복, 조희승, "리눅스 SSD caching mechanism의 성능 비교 및 분석," *Smart Media J.*, vol. 4, no. 2, 2015, pp. 62–67

[15] C. Min et al., "Understanding Manycore Scalability of File Systems," *Proc. USENIX Conf. Annu. Tech. Conf.*, Denver, CO, USA, June 22–24, 2016, pp. 71–85.

[16] J. Xu and S. Swanson, "NOVA: a Log-Structured File System for Hybrid Volatile/Non-volatile Main Memories," In *Proc. USENIX Conf. File Storage Technol.*, Santa Clara, CA, USA, Feb. 22–25, 2016, pp. 323–338.

[17] J.Y. Cho et al., "Dynamic Core Affinity for High-Performance File Upload on Hadoop Distributed File System," *Parallel Comput.*, vol. 40, no. 10, Dec. 2014, pp. 722–737.

[18] 조중연 외, "다중 큐를 지원하는 고속 I/O 장치를 위한 동적 코어 친화도," 정보과학회 논문지, 제43권 제7호, 2016. 7, pp. 736–743.

[19] 엄준용, 조중연, 진현욱, "네트워크 성능향상을 위한 시스템 호출 수준 코어 친화도," 정보과학회: 컴퓨팅의 실제 논문지, 제23권 제1호, 2017. 1, pp. 80–84.

[20] B. Ke et al., "CMSM: an Efficient and Effective Code Management for Software Managed Multicores," *Int. Conf. Hardw./Softw. Codes. Syst. Synthesis*, Montreal, Canada, Sept. 29–Oct. 4, 2013, pp. 1–9.

[21] Y. Kim et al., "Splitting Functions in Code Management on Scratchpad Memories," *Int. Conf. Comput.-Aided Des.,* Austin, TX, USA, Nov. 7, 2016.

[22] 서의성, 우영주, 반도체 장치를 위한 실시간 온도 예측 장치 및 방법, 등록번호: 10-1621655, 2016. 5. 10.

[23] N. Badano, et al., "A Thermal Margin Preservation Scheme for Interactive Multimedia Consumer Electronics," *IEEE Trans. Consumer Electron.*, vol. 62, no. 1, Feb. 2016, pp. 53–61.

[24] C. Min et al., "Cross-Checking Semantic Correctness: The Case of Finding File System Bugs," In *Proc. ACM Symp. Operating Syst. Principles*, Monterey, CA, USA, Oct. 4–7, 2015, pp. 361–377.

[25] I. Yun et al., "Apisan: Sanitizing API Usages Through Semantic Cross-Checking," *USENIX Security Symp.*, Austin, TX, USA, Aug. 10–12, 2016, pp. 363–378.

[26] Y. Park et al., "Ivy Profiler: A Lightweight Performance Analysis Tool for Multicore Systems," *Int. Techn. Conf. Circuits Syst., Comput. Commun.*, vol. 2015, no. 1, June 2015, pp.855–856.

[27] Linux Weekly News 2015, Accessed 2017. https://lwn.net/Articles/650776/

[28] B. Knudson et al., "IBM System Blue Gene Solution: Compute Node Linux," IBM Redpaper, 2009

[29] B. Gerofi et al., "On the Scalability, Performance Isolation and Device Driver Transparency of the IHK/McKernel Hybrid Lightweight Kernel," *IEEE Int. Parallel Distrib. Proc. Symp.*, Chicago, IL, USA, May 23–27, 2016, pp. 1041–1050

[30] S. Maass et al., "Mosaic: Processing a Trillion-Edge Graph on a Single Machine," *Proc. Eur. Conf. Comput. Syst.*, Belgrade, Serbia, Apr. 23–26, 2017, pp. 527–543.

[31] S. Ha et al., "Performance Analysis of an In-Memory Big Data Framework in a Near-Far Memory Platform," *Int. Tech. Conf. Circuits Syst., Comput. Commun.*, vol. 2017, June 2017, pp. 1–13.

[32] Ching, S. Edunov et al., "One Trillion Edges: Graph Processing at Facebook-Scale," *Proc. VLDB*, vol. 8, no. 12, Aug. 2015, pp. 1804–1815.

[33] 김연어 외, "병렬 프로그래밍 모델에 따른 Haskell 병렬 프로그램의 성능 비교," 한국정보과학회 학술발표논문집, pp. 1381–1383, 2016.

[34] 안형준, "Template Haskell을 이용한 Haskell 병렬 프로그램 이 방법 개선," 석사학위논문, 부산대학교, 2017.

[35] 채한울, 임은진, "Intel Xeon Phi 1,2 세대 가속기에서의 SGD의 병렬화 성능 연구," 2016년 한국정보과학회 동계학술발표회 논문집, pp. 1881—1883.