ETRI

2017
Electronics and
Telecommunications
Trends

매니코어 기반 고성능 컴퓨팅을 지원하는 경량커널 동향

Trends in Lightweight Kernel for Manycore Based High-Performance Computing

김진미 (J.M. Kim, jinmee@etri.re.kr) 차세대 OS 기초연구센터 책임연구원
차승준 (S.J. Cha, seungjunn@etri.re.kr) 차세대 OS 기초연구센터 선임연구원
전승협 (S.H. Jeon, shjeon00@etri.re.kr) 차세대 OS 기초연구센터 선임연구원
고광원 (K.W. Koh, kwangwon.koh@etri.re.kr) 차세대 OS 기초연구센터 선임연구원
정연정 (Y.J. Jeong, yjjeong@etri.re.kr) 차세대 OS 기초연구센터 책임연구원
김강호 (K.H. Kim, khk@etri.re.kr) 차세대 OS 기초연구센터 책임연구원
정성인 (S.I. Jung, sijung@etri.re.kr) 차세대 OS 기초연구센터 책임연구원/센터장

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

대규모 고성능 컴퓨팅 시스템에서 경량커널은 전통적으로 계산 노드에 탑재되어 특정 연산만을 수행한다. 특히 경량커널은 병렬 프로그램을 실행함에 있어 성능을 최대한 끌어올리기 위하여 자원 간의 간섭을 최소화할 수 있도록 개발되어 사용되고 있다. 최근에는 수천 개의 코어가 장착된 고성능 컴퓨팅 환경은 병렬프로그램뿐만 아니라 일반 응용 및 대규모 분산 응용에서도 필요하다. 고성능 컴퓨팅 환경에서는 매니코어와 메모리 자원이 늘어남에 따라 성능 확장성을 요구하는 현실적인 운영체제의 구조로서 경량커널과 리눅스를 같이 실행하는 멀티커널 구조를 선호하고 있다. 본고에서는 이러한 선행연구를 소개하고 매니코어 시스템에서 활용되는 최근 경량커널의 동향에 대해 살펴본다.

Ⅰ. 서론

최근 고성능 컴퓨팅 프로세서는 단일 코어당 클럭 속도 향상을 통한 성능 개선의 한계로 인해 코어의 수를 늘려 성능을 개선하는 매니코어 프로세서 시스템 환경으로 변화하고 있다. 메인 프로세서 기술은 멀티코어와 매니코어로 가속화되고 있으며, 주로 연산에 특화된 경량 코어를 사용한 매니코어는 기존 운영체제와 시스템 소프트웨어와는 차별화된 연구 방향을 요구한다.

인텔의 기술 로드맵에 의하면 이미 70여 이상의 코어를 가진 Xeon Phi 프로세서 제품군인 KNL(Knight Landing), KNM(Knight Mill)을 제시하고 있다. 또한, 이후에도 지속해서 코어를 확장한 프로세서, 코프로세서, FPGA(Field Programmable Gate Array), 네트워크 등 다양한 분야에서 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.

이러한 하드웨어의 발전과 빅데이터, 인공지능 분야 등 거대 산업에 필요한 고성능 컴퓨팅 요구에 힘입어 연구계에서는 수백~수천 코어 이상의 매니코어 시스템을 위한 연구를 진행하고 있다. 현재 매니코어 시스템에 적합한 공식적인 리눅스는 개발되지 않고 있으나 학계와 연구계에서는 코어 수가 많아짐에 따라 고성능 컴퓨팅 시스템에서 확장성 있는 성능을 얻기 위한 연구가 더욱 중요해지고 있다. 매니코어 시스템 환경에서 현재 범용 운영체제의 경우에는 Lock 구현, 캐시 미스, 캐시 일관성 기능 및 스케줄링 정책 등에서 확장성을 저해하는 여러 요인이 있음이 여러 연구 논문 등을 통해 밝혀져 있다. 이러한 문제점을 해결하기 위해 모노리틱 운영체제 뿐만 아니라 마이크로커널 구조의 운영체제, 혼합형 커널 구조를 가지는 운영체제, 멀티커널 운영체제 등 다각적인 운영체제 구조의 연구가 진행되고 있다[1]. 이렇듯 매니코어 운영체제의 구조를 결정함에 있어 커널의 특성 또한 고성능 컴퓨팅 시스템의 특장점을 결정하는데 중요한 역할을 한다.

본고에서는 앞서 언급한 매니코어 시스템 및 운영체제 구조에서 고성능 컴퓨팅에 활용되어 장착되는 경량 커널에 대해 알아보고자 한다. 먼저 경량커널에 관한 일반적인 특성을 조사하고 최근 시스템 발전에 따라 진화하는 여러 관련 연구를 살펴본다. 이러한 연구 내용을 분석하여 매니코어 기반의 고성능 컴퓨팅을 지원하는 경량커널에 관한 연구방향에 대해 소개하고 결론을 맺는다.

Ⅱ. 고성능 컴퓨팅의 경량커널 동향

1. 커널과 운영체제

운영체제는 사용자의 요구에 대해 컴퓨터가 동작할 수 있도록 항시 실행되는 소프트웨어 패키지이며, 그중 커널은 하드웨어 자원과 시스템 데이터를 직접 제어하는 운영체제에 포함된 핵심 부분이다. 인터페이스 측면에서 보면 운영체제는 사용자와 하드웨어 간 인터페이스라고 볼 수 있고, 커널은 소프트웨어와 하드웨어 간의 인터페이스라고 볼 수 있다.

운영체제의 엔진과도 같은 커널은 주로 하드웨어 자원을 추상화하여 관리하며 프로세스 관리, 메모리 관리, 장치 관리와 시스템 호출 부분 등을 포함한다. 커널은 개발 유형에 따라 전통적으로 모노리틱 커널과 마이크로커널로 구분한다. 모노리틱 커널은 커널이 있는 동일한 메모리 영역에서 커널의 주 스레드에 따라 운영체제의 서비스가 실행되는 형태이고, 마이크로커널은 운영체제의 서비스를 구현하기 위해 프리미티브 또는 시스템 호출을 사용하여 하드웨어에 대한 추상화를 제공하는 커널 구현 방법의 한 유형이다.

운영체제는 넓게는 사용자에게 제공하는 인터페이스이며 시스템의 특성과 목적성에 따라 단일시스템, 멀티

프로그래밍 시스템, 분산 운영체제 시스템, 실시간 운영체제 시스템 등으로 분류할 수 있다. 하지만 운영체제의 개념 역시 최근에는 시스템을 기준으로 분류하는 것을 넘어 웹 OS, 클라우드 OS 등 서비스에 특화된 운영체제의 형태로 광범위하게 불리고 있다. 본고에서는 운영체제 및 커널의 의미를 시스템에 기반을 두어 사용하였다.

2. 경량커널의 일반적인 특성

슈퍼컴퓨터와 같은 대규모 고성능 컴퓨팅 시스템은 성능을 최대한 끌어내기 위해 운영체제부터 응용에 이르기까지 최적화되어야 한다. 이러한 목적의 고성능 컴퓨팅 시스템은 일반적으로 여러 개의 계산 노드와 이를 관리하기 위한 서비스 노드로 구성한다. 계산 노드는 주로 특정 연산만을 수행하므로 경량커널로 불리는 LWK(Light Weight Kernel)을 활용한다. 이는 특히 병렬 응용 프로그램을 실행함에 있어 성능을 최대한 끌어올리기 위하여 자원 간의 간섭을 최소화할 수 있도록 개발된다. 서비스 노드는 계산 노드를 효율적으로 관리하도록 구성이 되어야 하며 범용으로 활용할 수 있도록 커널의 모든 기능을 가지는 FWK(Full Weight Kernel)을 사용한다.

고성능 시스템에 활용되는 전통적인 LWK는 여러 계산 노드에 각각 탑재되어 병렬화된 대규모 응용 프로그램에 대하여 성능 확장성을 가지며 실행시키는 것을 목표로 한다. 주로 다른 계산 노드와는 간섭 없이 동작하며 이를 병렬컴퓨팅에서는 'Embarrassingly parallel'이라고 한다. IBM의 Compute Node Kernel(CNK)이 이러한 목적성을 가진 대표적인 커널이다. LWK가 고려해야 할 기본 기능은 다음과 같다[2], [3].

- 하드웨어 추상화: 하드웨어 추상화 계층인 HAL (Hardware Abstraction Layer)에서 하드웨어를 제어하는 모든 기능을 고려할 필요는 없지만, 실행에 있어 필요한 하드웨어의 특성은 지원해야 함. Memory barriers, Atomic operations, Privileged instructions의 사용, 제어 레지스터 접근 등 특정 하드웨어를 다루는 기능이 필요함. LWK는 크기가 작고 대부분 복잡하지 않지만 이러한 하드웨어 특성으로 이식하기가 결코 쉽지만은 않음.
- 프로세스와 메모리 관리: LWK의 특성을 결정하는 주된 기능으로 대규모 응용 프로그램에 적합하게 페이지 크기를 크게 하고 커널 내의 버퍼링을 피하는 등 대부분 간단하고 효과적인 기능을 부여함. 오버헤드를 줄이고 성능을 높이는 중요도만큼 결정성 및 확장성 역시 중요한 요구사항이 됨.
- 실행 호환성: 리눅스 및 POSIX(Portable Operating System Interface) 호환을 고려. 실행 파일을 읽고 디코드하기 위해 ELF(Executable and Linkable Format) 인터프리터와 로더를 지원하고, 의사(Pseudo) 파일시스템을 지원하여 /proc, /meminfo 정보 등을 제공하며 로컬 입출력 기능을 위한 가상 파일시스템 계층을 제공.
- 부트 및 하드웨어 초기화: LWK에서 필요한 하드웨어를 부팅하고 초기화하는 기능이 요구.
- 디바이스 드라이버: LWK를 사용하고자 하는 시스템에서는 일반적으로 사용되는 여러 디바이스 장치를 사용하지 않으므로 선택적으로 필요함. 다만 차세대 메모리 등 시스템 환경에 따라 드라이버 제공 방안을 고려하는 것이 앞으로 시스템 구조를 결정하는 요구사항이 될 수 있음.
- 시스템 제어: 인터럽트 처리, 응용을 중단하거나 무시할 필요가 있을 때 흔히 커널에서 처리하는 시그널 처리와 시스템을 재설정하거나 시스템 오류가 발생할 때 제어하는 기능들을 필요에 따라 제공.

3. 고성능 컴퓨팅 시스템의 경량커널 사례

매니코어 기반의 차세대 고성능 컴퓨팅 시스템의 구

조를 결정하기 위해서는 하드웨어의 변화와 시스템의 특성에 따라 커널을 포함한 시스템의 운영체제 구조가 논의되어야 한다. 매니코어와 가속기 기술 등의 발전으로 인텔의 경우에는 빠른 호스트 프로세서인 Xeon과 느리지만 대량의 병렬프로세서(Massively-parallel co-processors)인 Xeon-Phi 들로 구성된 이기종 시스템으로 코어를 확장하여 고성능 컴퓨팅을 제공하고 있다. 이러한 구조에서는 호스트 프로세서와 코프로세서의 역할에 따라 커널과 운영체제의 기능들도 그 구조에 맞게 설계하고 사용하는 것이 바람직하다.

최근에는 코프로세서에 올라가는 운영체제도 멀티커널 구조를 선호하고 있는데 여기서 언급되는 멀티커널은 하나의 계산 노드에 경량커널과 리눅스를 동시에 실행하는 방법이다. 이는 한 노드에 코어의 수가 많아지고 다양한 특성의 응용 지원이 요구되기 때문이라고 볼 수 있다. 경량커널에서는 하드웨어의 특성을 충분히 살려 응용을 실행함에 있어 커널의 간섭 없이 실행할 수 있도록 하여 성능 확장성을 높이는 경우에 사용하며 동시에 실행되는 리눅스의 경우에는 일반적으로 디바이스 드라이버 지원 및 POSIX API를 지원하는 경우에 활용한다. 하지만 이 경우에도 두 커널 간의 시스템 자원을 활용하는 방안과 디바이스 드라이버의 실행 방법 등에 대해 효율적인 구조적 방안이 요구되고 있다.

멀티커널과 관련된 연구를 보면 Multikernel OS로 불리는 Barrelfish[4]에서 정의하는 운영체제의 개념과 McKernel[5] 및 mOS[6]에서 Multi-kernel로 정의하는 개념이 있으며 이 두 방법은 운영체제 구조에 있어 차이가 있다.

Barrelfish의 Multikernel OS에서는 이질적인 프로세서를 기반으로 하는 차세대 하드웨어 환경에서 새로운 개념의 매니코어 운영체제 구조를 제시하였다. 이질적 코어를 추상화하여 모듈화하고 각 코어 모듈에 대해 여러 개의 독립적인 OS 인스턴스를 정의하여 새로운 하드웨어에서도 코드 재사용을 높일 수 있도록 하였다. OS 인스턴스 간에는 메시지 기반의 통신을 할 수 있도록 운영체제를 구조화하였다. 즉, 코어 간 공유가 없다고 가정하고 마이크로 커널 방법으로 전통적인 운영체제 기능을 메시지 전달만으로 통신하는 분산 프로세스의 집합으로 정의하였다.

그와 달리 McKernel 과 mOS 에서는 매니코어 시스템에서 LWK 와 FWK를 역할에 맞게 배치하고 동시에 동작하게 하여 성능 확장성을 해결하고자 하는 시스템 구조를 제시하고 있다. 이러한 연구 방향들은 운영체제 구조에서는 차이가 있으나, 멀티커널에서 활용되는 경량커널의 개념에서는 공통으로 고려해야 할 기능이 있다. 본고에서 기술하는 멀티커널은 넓은 의미에서 하나의 서비스를 처리하는 경우에 다수의 커널을 활용하는 것을 의미한다.

가. ZeptoOS

미국의 아르곤 국립 연구소와 오레곤 대학교에서 공동으로 참가한 운영체제 프로젝트이다. IBM사의 Blue Gene/P 플랫폼을 기반으로 1만~100만 개의 CPU를 가진 환경에서 고성능 컴퓨팅을 위한 운영체제를 연구하고 있다. 운영체제의 경우에는 계산 노드와 입출력 노드를 구분하여 리눅스 기반의 경량커널을 개발하였다. 특징으로는 계산 노드에서 메모리에 접근할 때 성능을 높이기 위해 가상메모리의 개념을 대신하여 페이지 크기를 크게 한 빅메모리 구조를 제안하고 지원한다. 또한, 입출력 성능 개선을 위해 입출력의 경우 계산노드와 입출력 전용 노드 사이에 빠르게 입출력 포워딩을 수행하는 데몬을 두어 처리한다. 각 계산 노드들은 다른 계산 노드와 IPC로 통신하고 연동을 위해 리눅스 커널 위에 여러 단계의 레이어로 구성된 소프트웨어 스택이 존재한다.

DCMF(Deep Computing Message Framework)는 각

프로세스 사이에 메시지 전달속도를 최대화하여 빠르게 통신하기 위해 개발된 프레임워크이다. CCMI(Component collective messaging interface) 역시 다중 메시지 인터페이스로 같은 메시지 내용을 여러 프로세스에 효율적으로 전송할 때 사용한다. SPI(System programming interface)는 성능을 높이기 위해 시스템 Blue Gene/P에 특화된 인터페이스로 리눅스에서 수행할 수 있도록 코드를 번역한다. ZEPTO SPI로부터 시스템 호출의 형태로 하위 계층의 리눅스를 호출하고, 리눅스에서는 함수 호출의 형태로 바이너리 코드에 접근하게 하여 성능을 최대화한다[7].

나. IHK/McKernel

IHK/McKernel 은 일본의 과학기술연구소인 RIKEN에서 고성능 컴퓨팅을 목표로 멀티커널을 활용한 하이브리드 구조를 설계하고 새롭게 개발한 경량커널이다. IHK/McKernel은 성능에 민감한 시스템 호출 일부만을 구현하고 나머지는 시스템 호출 오프로딩 기능을 구현하여 리눅스에 위임하고 리눅스의 바이너리 호환 ABI를 유지하도록 설계되었다. 커널에는 자체 메모리 관리 기능과 간단한 라운드로빈 스케줄러로 프로세스와 멀티스레드를 지원하고 시그널링을 구현하였다. 또한, 프로세스 사이의 메모리 매핑을 허용하고 하드웨어 성능 카운터에 대한 인터페이스를 제공한다[5].

다. mOS

인텔의 엑사스케일 시스템을 연구하는 과제의 일환으로 개발한 멀티OS이며 McKernel을 개발하고 있는 일본의 RIKEN과 협업하고 있다. 리눅스 호환성보다 성능 및 확장성을 우선으로 새로운 하드웨어에도 설치가 어렵지 않고 런타임으로 하드웨어 특성을 고려하여 성능을 높일 수 있게 설계되었다. 매니코어 시스템의 다수 코어에는 리눅스를 설치하여 FWK 기능에 대해 호환성을 해결한다. 계산 코어에서 LWK 코드를 실행하고 리눅스가 나머지 코어를 관리하도록 함으로써 수행된다.

노드 부팅 때에 LWK에 지정할 CPU 및 메모리를 명시한다. 즉 mOS의 자원관리는 리눅스와 자원을 지정하여 부팅하는 것으로 시작한다. mOS에 실행할 프로그램은 자원을 예약하여 다른 프로그램과 중첩 실행을 하지 못하게 하고 실행 시 런타임으로 자원을 할당받아 사용하도록 한다[6].

Ⅲ. 매니코어에서 경량커널 현황

1. 고성능컴퓨팅을 위한 멀티커널 시스템

매니코어, 가속기 기술 등의 발전으로 인한 초고성능 컴퓨팅을 구현하기 위하여 운영체제는 크게 두 가지 연구방향을 따르고 있다. 리눅스를 개선하는 접근방식과 시스템 목적에 따라 새롭게 운영체제를 설계하고 개발하는 접근방식이다.

FWK인 리눅스를 기반으로 하는 접근 방식에서는 매니코어 및 다수 클러스터로 확장하는 시스템의 특성에 따라 확장성있는 운영체제의 연구가 진행되고 있다. 하지만 FWK의 경우 운영체제의 구조를 새롭게 하지 않으면 성능 확장성을 가지기에는 제한적일 것으로 예측할 수 있다.

반면에 LWK 접근 방식은 확장성 저해 요소들을 해결하고자 새로운 커널을 개발하고 호환성을 위해 필요한 API를 추가하는 방법을 따르고 있다. 각 접근 방법은 궁극적으로 성능을 목표로 하고 있으며 고성능 컴퓨팅을 필요로 하는 응용의 범위는 고전적인 거대과학용 계산 영역과 더불어 시각화 및 빅데이터 분석 등을 포함하여 확대되고 있으므로 리눅스 호환성의 필요성도 커지고 있다.

최근에는 노드에 코어 수를 증가하는 매니코어 프로세서의 발전으로, 하나의 계산 노드에 다수의 커널을 실행하는 실용적인 접근 방법이 연구되고 있다[5], [6]. 일

반적으로 리눅스와 LWK를 함께 실행하는 멀티커널 접근 방식으로 높은 성능과 확장성을 제공하는 동시에 리눅스 API를 유지할 수 있게 된다. 이 경우 두 커널 간에는 성능 격리와 응용 프로그램 구성에 중점을 두어 각 기능을 구성한다. 시스템 호출을 분류하여 LWK는 성능에 민감한 호출을 처리하고 리눅스는 호환성을 위해 나머지를 처리할 수 있도록 시스템 구조를 정한다. 다음은 실용적인 방안으로 멀티커널 접근법을 활용하고자 할 때 시스템 구조에서 고려해야 할 사항들을 기술한다[8].

가. 시스템 관리 측면에서 고려할 사항

멀티커널 시스템의 시스템 관리에서 운영체제를 관리하는 일반적인 절차에 추가로 다수의 커널을 운영하기 위하여 고려할 부분이 필요하다.

우선 LWK를 리눅스와 독립적인 바이너리로 관리할 것인가를 결정해야 한다. 앞서 기술한 경량커널 운영체제 McKernel의 경우에는 LWK를 독립적으로 운영한 사례이고, mOS의 경우에는 LWK 바이너리가 리눅스에 임베딩되어있는 경우이다. 이러한 구조들은 관리 측면과 아울러 시스템 호출 처리 과정을 결정하는 등의 운영체제 구조적 측면에도 영향을 미치게 된다. 부팅 가능한 LWK는 부팅 이미지를 만들고 유지 관리하기 위해 코드 기술, 전문 기술 및 세부 하드웨어 사양을 숙지하는 것이 필요하며 리눅스가 지원하지 않는 하드웨어를 빠르게 지원할 수도 있으나 이 또한 LWK에서 작성하고 유지하기가 쉽지는 않다.

시스템 부팅 절차에서도 한 노드를 대상으로 어떤 커널이 노드의 BIOS/펌웨어에 의해 먼저 부팅되는지 고려해야 한다. CPU 코어를 부팅하는 일조차 쉬운 작업이 아니므로 새로운 하드웨어를 지원하는 리눅스를 활용하고 LWK는 그 자체 역할만을 담당할 수 있도록 역할에 맞게 결정하는 것도 유용한 방안이 될 수 있다.

자원을 분할하는 측면을 보면 멀티커널 시스템에서 리눅스는 다른 커널과 자원을 공유하게 되어있지 않으므로 멀티커널 시스템에서 그 구조를 설계하고 또 다른 커널에 의해 자원을 나누고 관리해야 한다.

나. 응용 프로그램 측면에서 고려할 사항

멀티커널에서 응용 프로그램이 실행되는 경우, 현실적인 모델은 리눅스의 모든 기능 및 서비스를 받으면서도 LWK를 통해 성능 확장성을 보장받을 수 있는 것이다. 따라서 응용의 입장에서는 POSIX 호환성에 대해 어느 정도 지원을 할 것인지를 결정해야 한다. 대부분 현재 멀티커널 모델에서 LWK는 계산을 위해 사용하고 있고 리눅스를 통해 POSIX를 지원받는 방법을 선호하고 있다.

또한, LWK에서도 리눅스의 '/proc'과 같은 의사(Pseudo) 파일시스템을 지원할 것인가를 고려해야 한다. 의사 파일시스템의 경우 사용자 프로세스의 정보를 커널에 전달할 수 있어 유용할 수 있으며 리눅스 호환성을 제공할 수 있으나, 리눅스 인터페이스가 변경될 때마다 유지하기가 쉽지 않을 것이다.

무엇보다 호환성 문제를 어떻게 해결할 것인지 고려해야 한다. 성능을 중요시하는 일반적인 HPC 응용의 경우 시스템 호출은 거의 사용하지 않으나 POSIX 기반의 리눅스 인터페이스 사용이 적지 않다. McKernel에서는 리눅스 파티션에서 LWK 프로세스를 대신하여 시스템 호출을 하는 프록시 프로세스를 사용하는 반면, mOS는 호출하는 태스크를 리눅스 파티션으로 마이그레이션하고 시스템 호출을 실행하게 한다. 응용 프로그램이 LWK에서 실행되는 동안 동기화 문제로 다른 사용자는 리눅스에 접근할 수 없게 처리하기도 한다. 어떤 방법이든 시스템 호출의 경우 오버헤드를 가지게 된다.

LWK와 리눅스 프로세스 간 데이터 전달을 위해 공유메모리를 제공할 것인가를 고려해 본다. 리눅스로부터 LWK 프로세스에 데이터를 전달하기 위하여 요청하면 LWK 메모리에 직접 데이터를 저장하는 것이 가장 효율

적이므로 공유메모리를 제공하는 것은 유용하다.

NUMA 구조를 가지는 매니코어 시스템에서 LWK의 NUMA 특성 지원은 성능 확장성 측면에서 매우 중요하다. 응용 프로그램과 런타임 시스템에서 이러한 특성을 지원하도록 LWK에서는 시스템의 특성에 최적화된 기본적인 NUMA 처리와 필요한 정보를 제공하는 것이 필요하다. 커널에서 제공하는 시스템 특성을 인식하는 인터페이스를 사용하여 메모리 자원관리 등을 통해 성능을 향상할 수 있다. 그리고 무엇보다 리눅스가 LWK의 성능을 방해하지 않도록 성능 격리에 유념해야 한다.

다. FWK인 리눅스에서 고려할 사항

리눅스를 매니코어 시스템의 구성 요소로 설계할 경우 LWK의 프로세스를 ps와 top과 같은 표준 리눅스 도구에서 가시화될 수 있게 할 것인가를 고려해야 한다. HPC에서 리눅스를 고려하는 주된 이유 중 하나가 성능에 대해 고려해 볼 수 있는 도구들이므로 LWK와 함께 사용하기 위해서는 고려해야 할 요소이다. 리눅스와 별도로 LWK 에서 따로 제공할 수 있게 고려할 수도 있다. 이외에도 리눅스 커널 수정 여부와 리눅스가 업데이트 될 때마다 어떻게 지원할 것인지에 대한 전략을 가져가야 할 것이다.

라. 경량커널에서 고려할 사항

멀티커널의 LWK는 확장성이 뛰어난 고성능 환경을 제공하기 위한 것이다. 또한, 원하는 리눅스 기능을 제공해야 하므로 리눅스 호환성을 포함하여 LWK 설계를 어떻게 할 것인가를 생각해야 한다. 리눅스가 변경될 때마다 영향을 받게 되는 LWK의 변경 여부에 대해 고려해야 하고 개발 방안에 관해 결정해야 할 것이다.

LWK의 경우 코드 크기 및 복잡도도 중요한 고려 사항이다. 새로운 기능을 추가하고 새로운 하드웨어에 대한 이식성을 높이는 등 민첩성을 고려해야 하기 때문이다. 그리고 메모리 관리와 주소 관리 부분은 LWK의 기본 기능으로 물리적인 메모리를 관리하고 시스템의 메모리 계층 구조를 어떻게 효율적으로 관리할 것인가를 고려해야 한다. 또한, 커널이 사용할 가상 주소 범위를 결정해야 한다.

LWK의 스케줄링 정책을 결정하는 것은 작업이 실행될 때 간섭 없는 환경으로 성능을 높이는 중요한 구성 요소이다. 일반적으로 HPC를 위한 LWK에서는 시분할 정책과는 달리 비선점적인 공간분할 정책을 사용하는 것이 유용하다고 알려져 있다.

디바이스 드라이버를 제공하기 위해서는 리눅스를 활용하여 LWK에서 장치를 제어할 수 있도록 하는 구조 역시 유용할 수 있다.

2. 경량커널 전망 및 방향

매니코어 플랫폼에서 코어 수의 증가에 따른 운영체제 성능 확장성을 보장받지 못하는 것은 CPU와 메모리 자원을 효율적으로 사용하지 못함에 있다. 코어 수와 메모리가 늘어났음에도 현존하는 리눅스와 같은 범용 운영체제에서는 CPU의 활용률을 높이기 위해 시분할 스케줄링을 사용하고 이로 인해 캐시 미스율이 높아져 캐시 사용률에 영향을 미치게 되어 결과적으로 낭비되는 사이클이 발생한다. 또한, 공유메모리 모델에서 커널의 데이터 구조가 공유되기 때문에 스케줄링에 관한 정보를 공유하게 되고 코어 수의 증가로 다시 부하가 커져 성능 확장성을 기대할 수 없다. 즉, 여러 코어에서 동일한 메모리 위치로 동시에 접근하면 캐시 일관성을 유지하는 비용이 높아져 소프트웨어 성능에 직접적인 영향을 주기 때문이다.

이러한 이유로 매니코어 환경에서는 코어 수 증가에 따른 효율적인 성능 확장을 얻기 위하여 현존하는 운영체제를 그대로 사용하지는 못한다. 따라서 커널에서는 시스템의 자원 관리 방법을 다르게 하는 것을 고려해야

한다. 이 중 공간 분할 기반의 스케줄링으로 자원을 관리하는 방법은 매니코어 환경에서는 당연히 고려해야 할 기법으로 응용프로그램에서 사용할 자원을 할당하여 알아서 사용하도록 지원한다. 이를 통해 응용 프로그램은 자원 경쟁이나 인터럽트 같은 커널의 간섭없이 자원을 충분히 활용할 수 있으며, 파티셔닝을 통한 코어의 그룹 관리, 파티션 단위의 작업 할당 기술 등으로 매니코어 환경에서 성능의 확장성을 지원할 수 있다. 또한, NUMA를 인지한 메모리 할당 등 하드웨어 시스템 특성을 충분히 활용해야 한다.

그리고 최근에는 병렬프로그램뿐만 아니라 일반 응용 및 분산 응용에서도 고성능 컴퓨팅을 필요로 하고 있다. 데이터를 제어하는 응용이 많아지고 NVMe 와 같은 빠른 성능의 저장장치를 활용하고자 할 때 운영체제를 포함한 시스템 구조 연구도 필요하다.

다음으로는 가상화 기술을 적용하는 방안을 고려해 볼 수 있다. 경량커널은 성능을 위하여 기능의 제한을 감수하도록 설계된다. 경량커널이 갖는 서비스 지원의 문제를 해결하기 위해 경량 하이퍼바이저 기반의 가상화에 대한 연구 역시 병행되고 있다. Kitten의 경우 경량커널 내부에 Palacios라는 가상머신모니터를 삽입하여 이러한 단점을 극복할 수 있도록 설계되었다. 이외에도 가상화는 복잡한 이기종 하드웨어로 구성된 매니코어를 지원하기 위한 기술로 폭넓게 활용될 수 있다[8], [9].

Ⅳ. 결론

하드웨어 패러다임의 변화에 따라 수천 개의 코어가 장착된 고성능 컴퓨팅 시스템에 적합한 운영체제에 관한 연구는 최근 연구계에서 다시 주목받고 있다. 최근에는 고성능 컴퓨팅 응용에 특화되도록 하드웨어를 구축하는 것뿐만 아니라 각각의 시스템에서 적절한 운영체제를 함께 개발하고 있다. 매니코어 운영체제는 수천 개 이상의 프로세서, 분산된 메모리 구조와 결합형 네트워크를 가지는 고성능 컴퓨팅 시스템에서 성능 확장성을 제공해야 한다. 이러한 범위에서의 경량커널은 각각의 계산 노드가 지니고 있는 CPU, 메모리 등의 자원을 최대한 활용하면서 사용자의 응용 프로그램에게 예측 가능한 성능을 제공하는 것을 목표로 한다.

이에 따라, LWK는 점점 그 사용범위가 늘어남과 동시에 종류도 매우 다양해지고 있으며 고성능 시스템을 목표로 하는 시스템의 구축 목적과 이를 위한 운영체제의 역할에 따라 LWK의 특성 및 기능을 고려하여 구현해야 할 것이다.

용어해설

FWK(Full Weight Kernel) 범용으로 활용 가능하도록 하드웨어 추상화, 부트 및 하드웨어 초기화, 시스템 호출과 인터럽트 처리, 프로세스, 메모리, 입출력 등 자원 관리를 포함하는 모든 기능을 가진 커널로 윈도우 및 리눅스의 커널이 대표적임.

LWK(Light Weight Kernel) 커널의 기능을 최소화하여 슈퍼컴퓨터와 같은 대규모 고성능 컴퓨팅 시스템에서 계산 노드에 사용하는 커널로 많이 사용되고 있음. 최근 매니코어 프로세서의 발전으로 성능 확장성을 목적으로 시스템 및 응용에 특화된 코드 및 바이너리 크기가 적은 간단한 경량커널이 연구되고 있음.

약어 정리

API	Application Programming Interface
CCMI	Component Collective Message Interface
CNK	Compute Node Kernel
DCMF	Deep Computing Message Framework
ELF	Executable and Linkable Format
FPGA	Field Programmable Gate Array
FWK	Full Weight Kernel
HAL	Hardware Abstraction Layer
HPC	High Performance Computing
ISA	Instruction Set Architecture
KNL	Knight Landing
KNM	Knight Mill
LWK	Light Weight Kernel
NUMA	Non Uniform Memory Access
POSIX	Portable Operating System Interface
SPI	System Programming Interface

참고문헌

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

[2] R. Riesen et al., "Panel: What is a Lightweight Kernel?" *Int. Workshop Runtime Operating Syst. Supercomput.*, Portland, OR, USA, June 16, 2015, pp. 9:1-9:8.

[3] M. Giampapa et al., "Experiences with a Lightweight Supercomputer Kernel: Lessons Learned from Blue Gene's CNK," *Proc. ACM/IEEE Int Conf. High Performance Comput., Netw., Storage Anal.(SC)*, New Orleans, LA, USA, Nov. 13-19, 2010, pp. 1-10.

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

[5] RICKEN, McKernel, Accessed 2017. http://www-sys-aics.riken.jp/ResearchTopics/os/mckernel.html

[6] R.W. Wisniewski et al., "mOS: An Architecture for Extreme-Scale Operating Systems," *Proc. Int. Workshop Runtime Operating Syst. Suercomput.*, Munich, Germany, June 10, 2014, pp. 1-8.

[7] Argone National Laboratory, ZeptoOS, Accessed 2017. https://wiki.mcs.anl.gov/zeptoos/index.php/MPICH,_DCMF,_and_SPI

[8] B. Gerofi et al., "A Multi-kernel Survey for High-Performance Computing," *Proc. Int. Workshop Runtime Operating Syst. Supercomput.*, Kyoto, Japan, June 1, 2016.

[9] A. Vajda, *Programming Many-Core Chips*, NY, USA: Springer, 2011.