ETRI

2012
Electronics and
Telecommunications
Trends

고성능 컴퓨팅을 실현하는 런타임 시스템 기술 동향

Technology Trends of Runtime Systems to Realize High Performance Computing

김진미 (J.M. Kim) 고성능컴퓨팅SW연구팀 책임연구원
이재진 (J.J. Lee) 서울대학교 컴퓨터공학부 교수
최완 (W. Choi) 클라우드컴퓨팅연구부 부장

최근 산업의 발전으로 대규모 문제 해결의 요구가 커지고 사용자가 원하는 서비스를 신속하게 받고자 고성능 컴퓨팅에 대한 요구가 계속해서 증가하고 있다. 이에 따라 멀티코어 및 매니코어와 이종 하드웨어의 혼용 등으로 지속해서 발전하는 새로운 고성능 컴퓨팅을 위한 시스템은 컴퓨팅 패러다임을 바꿀 시스템 소프트웨어의 혁신 요소로 등장하였다. 하드웨어를 활용하여 시스템의 성능을 높이기 위해서는 컴퓨팅 요소 간의 통신을 최소화하여 전력 소모를 줄이고, 메모리 계층 구조 및 지역성을 고려하여 성능을 높이는 것이 필요하다. 특히, 응용의 실행 시에 시스템 자원을 최고로 활용할 수 있게 하여 성능을 높이는 런타임 시스템은 하드웨어 및 운영체제를 변경하지 않고 시스템 자원을 최대한 활용하여 성능 최적화를 이룰 수 있는 기술이다. 따라서 본고에서는 런타임 시스템의 기능과 기술 방향을 파악하여 차세대 런타임 시스템에 필요한 기술 및 연구 분야를 전망하고자 한다.

Ⅰ. 서론

최근 산업의 발전으로 대규모 문제 해결의 요구가 커지고 사용자가 원하는 서비스를 신속하게 받고자 고성능 컴퓨팅에 대한 요구가 계속해서 증가하고 있다. 고성능 컴퓨팅 기술은 현재 수많은 업체와 연구계 및 학교에서 연구 및 개발되고 있으며 이미 십여 개 이상의 코어가 집적된 멀티코어가 시장에서 판매되고, 수십~수백 개의 코어가 집적된 프로세서들도 연구용으로 개발되거나 상용화를 위해 현재 개발 중이다. 또한, CPU와 더불어 GPGPU(General Purpose computing on Graphics Processing Unit)와 MIC(Many Integrated Cores)와 같은 이종의 코어를 사용하는 성능 가속형 컴퓨터 구조들도 활발히 연구되고 있고 최근에는 CPU와 GPU를 하나의 칩에 통합한 프로세서들도 개발되고 있다. 이러한 기술을 토대로 고성능 컴퓨팅 분야는 페타플롭스급 성능을 넘어 엑사플롭스급을 향해 진보하고 있다.

고성능 첨단 하드웨어를 기반으로 최고의 성능을 내기 위해서는 운영체제에서부터 계산 라이브러리까지 소프트웨어 지원이 필수이며 이러한 기술에 대한 중요성이 두드러지고 있으나 대부분 특정한 하드웨어에서만 사용할 수 있고 다양한 특성들에 관한 연구가 부족하다.

특히 복잡한 이종 병렬 하드웨어를 효과적으로 활용하기 위해서는 사용하기 쉬운 프로그래밍과 서비스를 신속하게 받기 위한 고속 병렬 처리 기술이 요구된다. 즉 고성능 컴퓨팅 시스템의 병렬 처리 핵심 기술로는 시스템에 적합하고 쉬운 병렬프로그래밍 모델과 이를 활용하여 실행 시에 시스템 자원을 최고로 활용할 수 있게 하여 응용의 성능을 높여주는 런타임 시스템이 필요하다[1],[2].

따라서 본고에서는 런타임 시스템의 전반적인 기술 현황과 발전 방향을 살펴보고자 한다. 먼저 Ⅱ장에서는 런타임 시스템의 역할 및 국내외 개발 현황을 살펴보고, Ⅲ장에서는 런타임 시스템의 기능과 기술 방향을 파악하여 이를 토대로 차세대 런타임 시스템에 필요한 기술 및 연구 분야를 전망하고자 한다.

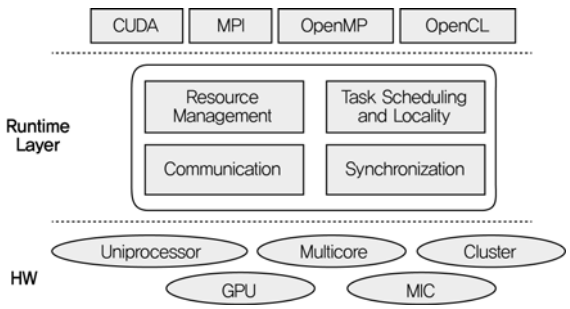
Ⅱ. 런타임 시스템 역할 및 동향

1. 런타임 시스템 역할

런타임 시스템은 사용자 응용 프로그램의 효율적인 실행을 지원하기 위하여 프로그램과 연결되어 사용자 공간에서 실행되는 소프트웨어이다. 시스템의 운영체제에서 제공해주지 못하는 특정 프로그래밍 모델에 특화된 기능을 제공하여 사용자가 응용을 실행할 때 시스템의 프로세서, 메모리, 네트워크 등의 여러 자원 상태를 고려하여 응용의 성능을 최고로 하는 것이다.

고성능 컴퓨팅 성능을 높이기 위한 런타임 시스템의 기능은 프로그램의 실행 기본 단위인 태스크의 병렬 제어, 병렬로 동작하는 태스크 간의 통신 및 동기화, 태스크 스케줄링, 메모리 관리, 사용자 제어 가능 정책 등 자원관리 기능을 포함한다. 이러한 기능은 시스템에 대한 추상화를 통해 사용자에게 제공되어 프로그램의 생산성, 성능, 이식성 및 확장성을 향상하고 사용자는 하드웨어에 대한 전문적인 지식 없이도 응용의 알고리즘만을 고려하여 프로그램을 작성하면 런타임 시스템이 자동으로 시스템을 최적화해 줄 수 있게 한다[3].

이러한 런타임 시스템은 응용 소프트웨어의 특성 및 시스템 상황에 따른 최적의 실행 환경을 제공하여 성능



(그림 1) 이종 병렬 시스템 런타임 구조

과 효율성을 극대화할 수 있게 하는 것으로, 응용 소프트웨어가 사용하는 프로그래밍 수준에서부터 시스템 특성을 활용하여 제공하는 하부구조에 이르기까지 광범위하게 제공될 수 있다. (그림 1)은 이종 병렬 시스템을 지원하는 런타임 구조이다.

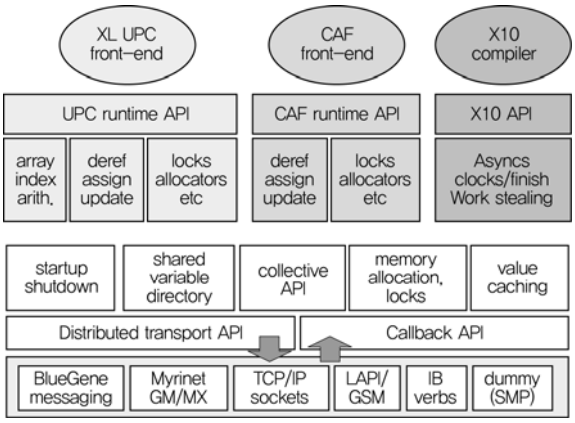
2. 런타임 시스템 현황

가. PGAS

PGAS(Partitioned Global Address Space)는 George Washington 대학, UC Berkeley 등에서 개발이 이루어졌으며 공유 메모리 모델과 메시지 전달 모델의 일부를 합쳐 놓은 형태의 프로그래밍 모델로 사용자가 전역 메모리 공간 사용 시 지역성 반영이 가능하도록 한 메모리 모델이다. 전역 메모리 주소 공간은 논리적으로 분할된 각 프로세스의 메모리 영역을 하나의 주소 공간으로 통합한 것이다. 기본적인 형태는 공유 메모리와 같으나 기존의 공유 메모리 모델이 가지고 있는 지역성을 높이기 위하여 사설 메모리(private memory) 개념을 사용한다[4].

PGAS 모델을 사용자가 편리하게 이용하기 위해 프로그래밍 모델을 제공하며 UPC(Unified Parallel C), CAF(Co-Array Fortran), Titanium, X10, Chapel 등이 사용된다. 이는 SPMD(Single Program Multiple Data) 프로그래밍 모델과 실행을 지원하고 UPC와 CAF는 기존의 언어를 확장한 방식이며 X10과 Chapel은 PGAS 모델을 구현한 언어들이다[5].

PGAS 실행은 GASNet(Global Address Space Networking) 지원과 통신 라이브러리를 제공하여 성능을 향상하고 효율적인 자원관리를 위하여 작업 관리 시스템을 제공한다. GASNet은 네트워크와 프로그래밍 언어에 독립적인 고성능 통신 인터페이스를 제공하며 그 인터페이스는 응용 프로그래머가 직접 알아야 할 필요 없이 프로그래밍 모델의 실행에 제공된다. (그림 2)는 PGAS 모델 언어의 런타임 구조이다.

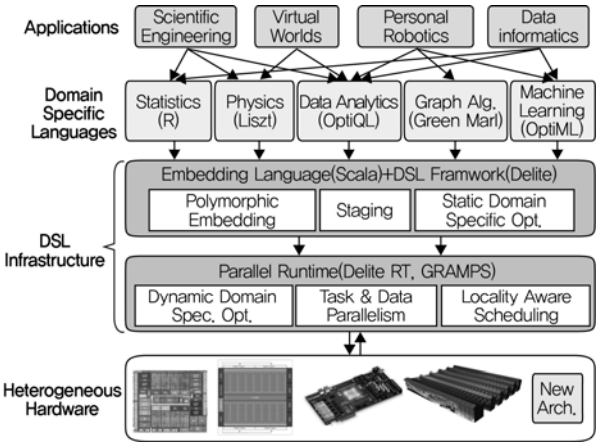


(그림 2) PGAS 런타임 구조[5]

그러나 PGAS는 SIMD(Simple Instruction Multiple Data), MIMD(Multiple Instruction Multiple Data) 등의 환경에 대한 동시 고려가 되어있지 않고 MIC와 같은 새로운 아키텍처에 대한 대비가 미비한 실정이다.

나. PPL DSL Infrastructure

Stanford 대학의 PPL(Pervasive Parallelism Laboratory)에서 개발하고 있으며 성능 향상을 목적으로 이종 시스템의 병렬성을 지원하기 위해 도메인에 특화된 소프트웨어를 제공한다. 도메인에 특화된 언어인 DSL(Domain Specific Language)을 사용하여 각 응용의 특성을 지원하고 이종 시스템을 활용하기 위한 병렬성을 제공한다. (그림 3)에서 병렬 실행으로 제공하는 Delite



(그림 3) PPL 런타임 구조[6]

는 컴파일러에 의한 런타임 시스템으로 병렬 코드를 위한 최적화, 도메인에 특화된 언어들을 실행하기 위한 이종 실행, 코드 생성 부분 등이 있다[6]. PPL의 런타임은 단일 노드 이종 프로세서만 고려되어 있고 다중 노드 환경에 대한 동시 고려가 되어있지 않으며, 새로운 하드웨어 아키텍처나 DSL에 바로 대응할 수 없는 문제가 있다.

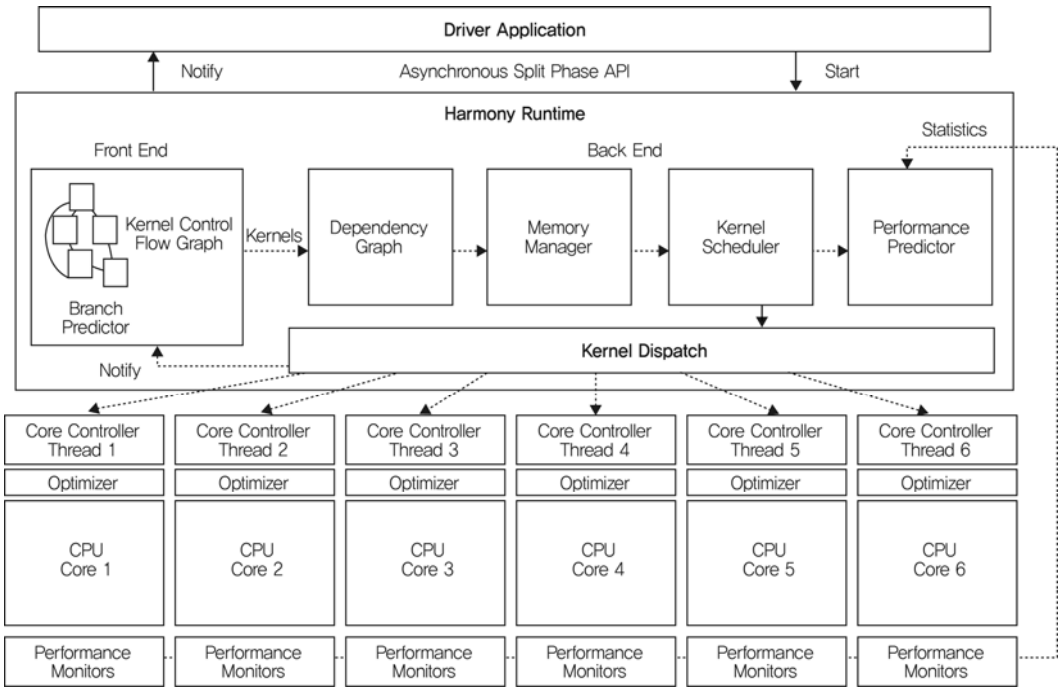
다. Harmony

Georgia Tech.에서 개발했으며 이종 매니코어 시스템의 실행 모델 및 실행을 제공한다. 이종의 멀티코어 하드웨어 구조를 효과적이고 효율적으로 이용함에 있어 복잡성을 제거하고 성능 손실을 최소화하면서 생산성 향상을 위하여 스케줄링, 파티셔닝(데이터/코드 분할), 병렬화 등을 (그림 4)의 하모니 런타임 시스템이 사용 가능한 정보를 통해 동적으로 지원한다. 이종의 프로세서 자원에 대하여 응용 커널들의 동적 스케줄링을 수행하며 이외에도 동적 실행 환경을 제공하는 Ocelot, 데이터 중심 컴퓨팅을 위해 GPU 성능을 향상시키는 Red Fox 프로젝트를 진행한다[7].

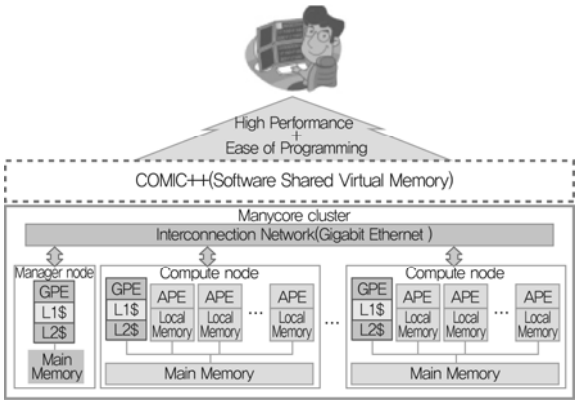
하모니를 사용할 경우 새로운 아키텍처가 생길 때마다 새로운 라이브러리를 제작해야 하므로 빠른 대응이 어렵고, 다중 노드 환경에 대한 동시 고려가 되어있지 않은 문제가 있다.

라. COMIC/COMIC++

서울대학교에서 개발한 COMIC(Coherent Shared Memory Interface for Cell BE)는 IBM의 Cell BE 프로세서를 위한 공유 가상 메모리 프로그래밍 모델로 사용자에게 매니코어 시스템의 복잡한 메모리 계층을 하나의 메모리 공간으로 보이도록 하여 프로그래밍을 쉽게 하면서 성능을 높인다. COMIC++는 클러스터 환경을 위한 프로그래밍 모델로 COMIC 프로그램을 별다른 수정 없이 클러스터 환경에서 수행시킬 수 있다. (그림 5)의 여러 매니코어 노드로 구성된 클러스터 시스템에서 동작하는 소프트웨어 공유 가상 메모리 지원 런타임 시스템으로 프로그래머에게 통합된 하나의 메모리 주소



(그림 4) Harmony 런타임 구조[7]



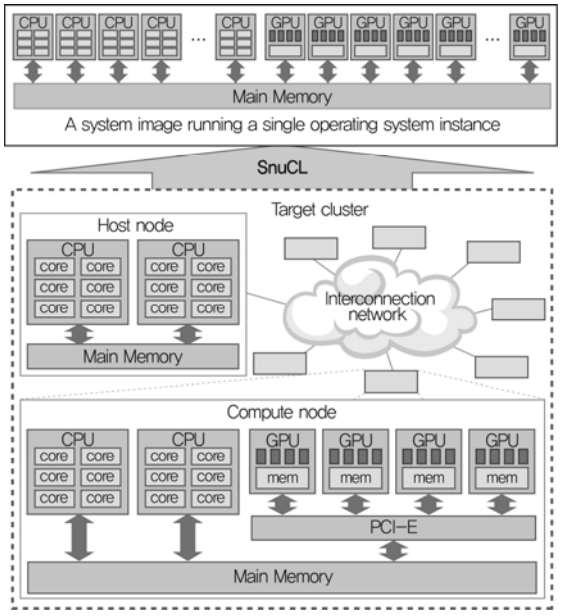
(그림 5) COMIC++ 런타임 구조[8]

체계를 제공함으로써 메모리 일관성 유지나 데이터 전달을 위한 어려움을 제거하였다[8].

Cell BE 프로세서는 현재 생산이 중단된 상태이나 같은 기법을 Intel의 SCC(Single-chip Cloud Computer)와 같은 하드웨어 캐시 일관성을 지원하지 않는 멀티코어에 적용할 수 있다[9].

마. SnuCL

서울대학교에서 개발한 SnuCL은 각 노드에 여러 개의 멀티코어 CPU와 GPU가 장착되어 있는 이종 클러스터에서 OpenCL(Open Computing Language)로 작성된 프로그램을 실행해 주는 런타임 시스템이다. OpenCL은 Khronos Group에서 제정한 이종 컴퓨터 시스템을 위한 표준 프로그래밍 모델이다. 하지만 OpenCL은 단일 노드 이종 프로세서만을 고려해 만들어졌으며, 다중 노드 환경에서는 통신을 위해 MPI(Message Passing Interface) 등의 프로그래밍 모델을 추가로 사용해야 한다. SnuCL은 (그림 6)과 같이 각 노드에 위치한 CPU나 GPU가 마치 한 개의 노드에 모여 있는 것처럼 추상화하여, 응용 프로그래머가 단일 노드를 위한 프로그램을 작성하듯이 클러스터를 위한 코드를 작성할 수 있도록 한다. 런타임 시스템이 자동으로 메모리 객체의 일관성을 유지하고 노드 간 동기화와 스케줄링을 관리한다. 따라서 프로그래머는 노드 간 통신을 신경 쓰지 않고 쉬운



(그림 6) SnuCL 런타임 구조[10]

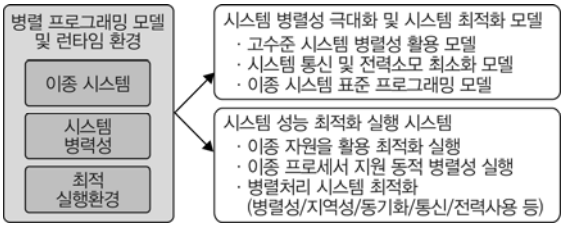
프로그래밍이 가능하다. 더불어 대부분의 이종 병렬 하드웨어 벤더들이 OpenCL을 프로그래밍 모델로 지원하고 있으므로 새로운 아키텍처에 대응하기 쉽다는 장점이 있다[10].

다만, SnuCL의 경우 하나의 호스트 노드에서 다른 모든 노드가 실행할 명령을 전송하고 자원을 관리하는 중앙 집중형 구조를 사용하고 있어, 128개 정도의 노드로 구성된 클러스터까지는 확장성이 보장되나 이보다 많은 노드를 가지는 클러스터는 확장성이 떨어져서 이를 보완하는 연구를 진행 중이다.

Ⅲ. 런타임 시스템 기술 전망

1. 런타임 시스템 기능

런타임 시스템은 응용의 실행을 효과적으로 지원하기 위하여 해당 응용과 연결되어 사용자 공간에서 실행되며 시스템 구조를 최대한 숨기고 추상화된 모습을 사용자에게 제공하는 것이 중요하며 이종 시스템을 지원하



(그림 7) 런타임 시스템 요구 기능

고 병렬성 및 최적 실행 환경을 제공하여야 한다((그림 7) 참조)[1],[2].

고성능 컴퓨터 시스템의 성능을 최대화하기 위하여 제공할 수 있는 런타임 시스템 기술은 태스크의 병렬 제어 기술, 병렬로 동작하는 태스크 간의 통신 및 동기화 기술, 태스크 스케줄링 기술, 그리고 메모리 관리 기술 등이다[3].

가. 병렬성

런타임 시스템에서 제공하는 병렬성은 실행에서 계층을 이루는 모습에 따라 플랫(flat), 내포(nested), 동적(dynamic) 병렬성으로 나누어진다.

플랫 병렬성은 병렬 태스크 간 계층 구조가 존재하지 않아 응용이 시작할 때 모든 병렬 태스크가 생성되며 종료할 때 모든 병렬 태스크도 종료된다. 보통 태스크 간 의존성이 없는 병렬 응용에 적합하며 대표적인 예가 MPI이다. 플랫 병렬성의 경우 시스템의 프로세서에 대한 로드 밸런싱이 어렵다.

내포 병렬성은 병렬 태스크 간 계층 구조가 존재하여 중첩 병렬 루프를 포함하는 응용이나 노드마다 멀티코어 CPU가 장착된 클러스터 시스템에 적합하다. Open-MP와 같은 프로그래밍 모델에 적합하다.

동적 병렬성은 응용 실행 시에 새로운 병렬성이 요구될 때마다 새로운 태스크를 생성하여 실행 도중 일을 처리하는 노드의 개수나 프로세서 개수를 변화시키는 확장성 있는 실행에 적합하다. 동적 병렬성을 제공하는 런타임 시스템으로는 워크 스틸링이나 태스크 이동 같은 기법을 통해 시스템 로드 밸런싱을 유지하며 Pthread, Chapel 같은 모델이 있다.

나. 통신

고성능 컴퓨팅 시스템은 분산된 시스템 환경에서 각 노드가 기가비트이더넷이나 인피니밴드 스위치로 서로 연결되어 있으며 해당 시스템에서 실행되는 병렬 응용은 네트워크 구조가 적용될 수 있도록 팻트리(fat tree)나 크로스바(crossbar)와 같은 구조를 가진다. 사용자 응용은 런타임 시스템이 제공하는 통신 기능을 이용하여 노드 간에 서로 데이터 및 코드를 주고받는다. 이때 사용자에게 시스템 구조를 노출해 사용자가 직접 각 노드 간 데이터를 주고받게 하는 통신을 저수준 통신이라고 하고 시스템 구조를 추상화하여 시스템 구조를 숨긴 채 사용자가 추상화된 객체를 통해 노드 간 데이터를 주고받게 하는 것을 고수준 통신이라고 한다.

저수준 통신은 그 사용이 복잡하고 개발자 실수로 말미암은 오류 발생이 빈번해져 프로그래밍 생산성이 낮아지고 시스템 네트워크가 변하면 기존 응용의 수정도 필요하게 된다. 이러한 응용의 이식 가능성 문제를 해결하기 위해 네트워크 구조를 저수준에서 추상화한 것이 MPI와 액티브 메시지(active message)이다. 이들 모델은 네트워크 구조를 저수준으로 추상화하여 사용자 응용의 최적화 가능성이 매우 높으나 프로그래밍 복잡도는 여전히 존재하게 된다.

고수준 통신은 네트워크 구조를 추상화하여 사용자 모르게 런타임 시스템이 자동으로 통신하게 되어 프로그래밍 복잡도를 해결하여 생산성이 높아질 수 있다. 하지만 자동으로 데이터의 이동이나 위치를 지정하기 때문에 사용자가 원하는 최적화를 이루기가 어렵다. HPF 등의 고수준 프로그래밍 모델에 해당된다.

응용의 성능에 가장 크게 영향을 끼치는 데이터 이동의 최적화는 매우 중요한 요소이다. 따라서 저수준 통신의 최적화와 고수준 통신의 프로그래밍 생산성을 동시에 만족하게 하는 노력이 계속되고 있다. 특히 미국방위고등연구계획국(DARPA)에서 수행된 고효율 컴퓨팅 시

스템(HPCS: High Productivity Computing Systems) 프로젝트에서 제안된 Chapel, Fortress, X10 같은 모델은 고수준 통신 기능을 제공하여 프로그래밍 생산성을 높이는 동시에, 저수준 통신의 특징인 데이터의 이동, 위치 지정을 할 수 있는 기능을 사용자에게 제공하여 높은 최적화 가능성을 달성하려고 시도하고 있다.

다. 동기화

시스템이 제공하는 병렬성을 활용하기 위해서 병렬 응용은 많은 수의 태스크를 동시에 실행시키며 올바른 동작을 위하여 실행 도중 서로 간의 동기화가 필요하다. 동기화가 실행되는 계층 구조에 따라 노드 내 동기화 및 노드 간 동기화로 나눌 수 있다.

노드 내 동기화는 주로 폴링이나 시그널/인터럽트 기법이 사용되며 폴링의 경우에는 한 프로세서에서 태스크 간 문맥교환이 필요하지는 않으나 주기적으로 조건을 검사하기 위해 프로세서 시간을 낭비할 수 있으므로 전체 시스템 성능이 낮아진다. 시그널/인터럽트는 태스크가 기다리는 동안 다른 일을 처리할 수 있어 프로세서 낭비가 없을 수 있으나 태스크 간 문맥교환과 인터럽트 처리 등의 추가적 작업으로 주로 폴링에 비해 느리다.

노드 간 동기화를 위해서는 주로 메시지 전달이 사용되는데 원격 함수 호출을 위한 원격 동기화를 위해서 주로 액티브 메시지를 사용한다. 액티브 메시지는 호출과 실행 완료의 두 가지 위상으로 나누는 위상 분할 모델 사용으로 원격 노드가 함수를 완료할 때까지 기다리지 않고 자기 일을 계속할 수 있어 프로세서의 시간을 낭비하지 않게 된다. 액티브 메시지가 사용하는 위상 분할 모델은 메시지 전달을 비동기식으로 실행하기 때문에 계산과 통신의 동시 실행을 가능하게 해준다. 이는 분산 병렬 시스템에서 가장 중요한 최적화 기법의 하나다.

라. 스케줄링

병렬 응용의 실행 시에 시스템 내 프로세서를 모두 활용하기 위해서는 올바른 태스크 스케줄링이 필요하다. 태스크 스케줄링은 크게 정적 태스크 스케줄링과 동적 태스크 스케줄링으로 나누어진다.

정적 태스크 스케줄링은 응용이 실행되기 전에 태스크를 각 프로세서로 분배하는 것으로 블록 스케줄링, 순환 스케줄링, 블록 순환 스케줄링이 있다.

블록 스케줄링은 전체 태스크를 전체 프로세서의 개수로 나누어 각각 나누어진 태스크의 집합이 하나의 블록이 되고 하나의 프로세서가 하나의 블록을 실행하는 것이다. 전체 개수가 프로세서의 개수로 나누어 떨어지지 않거나 각 블록의 실행 시간이 다르다면 프로세서의 대기 시간이 필요하게 된다. 구현과 실행은 매우 간단하나 로드 밸런싱에 어려움을 갖는데 주로 MPI 모델에서 가장 많이 사용하는 스케줄링 기법이다.

순환 스케줄링은 각 프로세서에 태스크를 라운드 로빈으로 돌아가며 배분한다. 블록 스케줄링보다 로드 밸런싱이 좀 더 효율적이나 하나의 프로세서가 실행하는 태스크들이 서로 떨어져 있어 데이터 지역성이 낮아지므로 캐시 미스가 높아질 수 있다. 블록 순환 스케줄링은 두 스케줄링의 혼합으로 블록으로 나누어 하나의 블록을 일정한 수의 태스크로 구성한 후 각 블록을 순환 스케줄링한다. 순환 스케줄링의 로드 밸런싱 장점과 블록 스케줄링의 지역성 향상을 동시에 만족할 수 있다.

동적 태스크 스케줄링은 실행 시 시스템의 로드 밸런싱을 고려해 태스크를 각 프로세서로 분배하는 것으로 셀프 스케줄링, 워크 스틸링, 태스크 이동 스케줄링이 있다. 셀프 스케줄링은 각각 프로세서가 전체 남아있는 태스크 중에서 일부를 가져오는 것으로 구현이 간단하나 가져오는 태스크의 양에 따라 로드 불균형이나 스케줄링 오버헤드가 커질 수 있다. 워크 스틸링은 일이 없는 프로세서가 일이 많은 프로세서에서 일을 뺏어오는 기법으로 각 프로세서에서는 큐를 두어 일을 관리한다. 태스크 이동 스케줄링은 런타임 시스템이 전체 시스템의 로드 밸런싱을 위해서 태스크들을 여러 프로세서 간

이동시키는 기법으로 이동에 필요한 비용이 고려되어야 하므로 보통 정책적으로 반드시 필요한 경우에만 수행하는 것이 바람직하다.

마. 메모리 관리

가장 단순한 형태의 메모리 관리 방법은 사용자가 직접 시스템 메모리를 할당하고 해제하는 기능을 제공하는 것이다. 하지만 이는 모든 메모리를 관리해야 하는 부담이 있으므로 대용량 고성능 병렬 시스템에서는 적합하지 않다. 따라서 런타임 시스템은 사용자의 메모리 관리 부담을 덜어주고 고성능을 실현하기 위해 메모리 가상화, 데이터 캐싱, 데이터 이동 등의 기능을 제공한다.

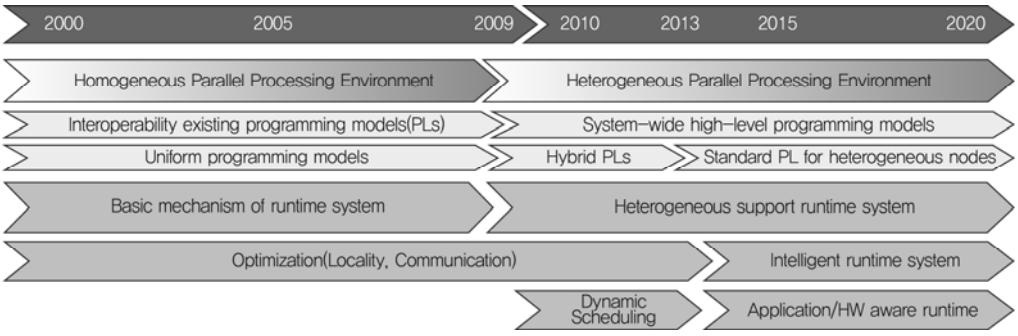
메모리 가상화는 시스템의 네트워크 위상과 그에 따른 메모리 계층의 정보를 사용자로부터 숨기고 가상화된 메모리 구조를 사용자에게 제공하는 것으로 프로그래밍 생산성을 높인다. 데이터 캐싱은 지역성을 활용하여 크기가 작지만 프로세서에 가까워 접근 시간이 짧은 메모리에 프로세서가 접근했던 데이터의 블록을 저장함으로써 시스템의 성능을 높인다. 데이터 이동은 데이터 접근 패턴을 파악하여 각 노드가 자주 사용하는 데이터를 타 노드에서 자신의 노드로 이동시키는 것이다.

2. 런타임 시스템 연구 방향

런타임 시스템은 응용의 특성에 따라 시스템의 성능을 최적화하기 위해서 사용] 가능한 시스템 자원을 자동으로 관리해 주는 것으로 사용자는 응용이 동작하는 하드웨어의 전문적인 지식 없이도 응용의 알고리즘만을 생각하고 작성하면 자동으로 시스템을 최적화할 수 있어야 한다.

기존 발표된 대부분의 런타임 시스템은 다양한 이종 프로세서에 대한 일관된 시스템이 없고 시스템의 확장에 따른 성능의 확장성에도 문제를 가진다. 최근에 이러한 문제를 해결하기 위하여 이종 컴퓨터 시스템을 위한 표준 프로그래밍 모델인 OpenCL을 제정하면서 여러 플랫폼을 아우르는 이식성을 제공하고 있다. 하지만, 이종 시스템의 성능을 최적화할 수 있는 성능 이식성 측면에서는 아직 많은 연구가 필요하다. 즉, 이종 프로세서 간의 성능 차를 고려한 작업 분배가 필요하고 병렬 실행 대상이 되는 태스크의 크기를 결정하는 문제도 있다. 그리고 특히 성능에 영향을 줄 수 있는 이종 프로세서 간의 통신 최적화도 필요하다.

앞으로 이러한 런타임 시스템은 다양한 응용의 특성, 여러 장치의 이종성, 그리고 메모리 계층에 따른 통신 지연시간 등을 고려하여 설계되어야 할 것이다. 그러기 위해서는 쉬운 프로그래밍을 가능하게 하는 추상화 제공, 응용 및 시스템 성능 향상을 위한 동적 병렬성 제공, 프로그래밍 생산성과 이식 가능성을 높이는 고성능 통신 수준의 지원, 프로세서 간 로드 밸런싱에 유리한 동적 태스크 스케줄링 지원 등이 필요하다((그림 8) 참조)[11].



(그림 8) 런타임 시스템 기술 방향

이러한 요구사항을 기반으로 변화하는 시스템 환경을 지원하는 런타임 시스템 연구 개발의 중점 방향에 대해 알아보도록 한다. 이러한 연구 방향은 서로 유기적으로 작용하여 사용 편이성과 시스템 성능을 높일 수 있을 것이다.

가. 이종 시스템 지원

앞으로 고성능 컴퓨팅 시스템은 여러 가지 다른 이종 프로세서로 구성되므로 차세대 런타임 시스템은 여러 다른 플랫폼상에서 동작 가능해야 한다. 이종성에 대한 런타임 시스템 연구는 응용의 동작 특성과 하드웨어의 특성을 분리함으로써 이종 프로세서에 대한 추상화된 이미지 제공으로 사용자의 개발 생산성을 높이고, 응용 자체의 이식 가능성을 높여야 한다.

나. 런타임 시스템 최적화

하드웨어 자원을 효율적으로 할당하여 시스템 자원 낭비를 최소화하는 것이 필요하다. 실행 시간과 전력 소비를 최소화하기 위해 여러 가지 기술을 사용할 수 있으며 주로 메모리 용량을 줄이고 프로세스를 관리하며 프로세스 간 동기화하는 오버헤드를 줄이는 등의 최적화가 필요하다. 이종 시스템에서는 각 프로세스의 처리량을 예측하여 자동으로 로드 밸런싱하는 시스템 성능 예측 방법을 사용할 수 있다. 고성능 컴퓨팅에서는 그 규모 면에서 여러 메모리 계층 구조를 가지므로 메모리 이동에 대한 비용이 매우 커질 수 있는데 이러한 메모리 이동의 비용을 줄이는 연구 방향이 필요하다.

다. 런타임 시스템 지능화

런타임 시스템은 응용이 실행될 때, 해당 응용에 할당된 시스템 자원에 대한 정보를 알 수 있고, 시스템 자원이 할당될 때 앞으로 얼마만큼의 시스템 성능 향상이 가능한지 예측할 수 있어야 한다. 앞으로는 기계 학습과 같은 좀 더 지능적인 접근으로 자동으로 응용과 하드웨어 구조에 따른 시스템을 최적화하는 연구 방향이 필요하다.

Ⅳ. 결론

시스템의 하드웨어 성능은 지속해서 발전하고 있으며 선진국은 고성능 컴퓨팅에 관한 연구 개발을 통하여 높은 수준의 기술을 보유하고 있다. 이러한 기술로 복잡한 이종 병렬 하드웨어를 효과적으로 활용하기 위해서는 쉬운 프로그래밍과 성능을 높여 원하는 서비스를 신속하게 받는 것이 필요하므로 병렬프로그래밍 모델과 이를 활용한 병렬 처리 기술은 고성능 컴퓨팅에 필수 불가결한 요소가 되었다.

병렬 처리 기술로 시스템의 성능을 높이기 위해서는 컴퓨팅 요소 간의 통신을 최소화하여 전력 소모를 줄이고, 메모리 계층 구조 및 지역성을 고려하여 성능을 높이는 것이 필요하다. 국내에서는 하드웨어 제조가 기업의 의지 없이는 매우 힘든 상황이므로 하드웨어를 최대한 활용할 수 있는 런타임 시스템 기술, 컴파일러의 성능 최적화 기술 등 병렬 처리 환경 기술을 중점으로 연구 개발 전략을 세울 수 있다.

특히, 응용의 성능을 실행 시에 시스템 자원을 최고로

용어해설

런타임 시스템 기술 사용자 응용의 효율적 실행을 지원하기 위하여 프로그램과 연결되어 사용자 공간에서 실행되는 소프트웨어로 주로 특정 프로그래밍 모델에 특화된 기능을 제공하여 응용을 실행할 때 시스템의 자원 상태를 고려하여 응용의 성능을 최고로 하는 기술

플롭스 컴퓨터의 성능을 수치로 나타낼 때 사용되는 단위로 페타플롭스는 초당 10^{15}(1초에 1,000조) 연산으로 2GHz PC 500,000대 규모이며, 엑사플롭스는 초당 10^{18}(1초에 100경) 연산

GPGPU 500여 개의 GPU 코어를 내장하여 1테라플롭스급 성능을 제공하는 엔비디아에서 개발한 시스템 반도체

MIC 50여 개의 x86급 CPU를 하나의 칩에 내장한 인텔에서 개발한 시스템 반도체

OpenCL Khronos Group에서 제정하여 CPU, GPU 및 이종 플랫폼에서 데이터 및 태스크 병렬성을 갖는 프로그램을 작성할 수 있게 해 주는 표준 병렬 컴퓨팅 프레임워크

활용할 수 있게 하는 런타임 시스템은 하드웨어 제조 없이도 시스템의 이해뿐만 아니라 기술과 경험 축적에 필요한 연구 분야로 국내 연구 개발로 추진하는 것이 바람직하다.

약어 정리

CAF	Co-Array FORTRAN
COMIC	Coherent Shared Memory Interface for Cell BE
DSL	Domain Specific Language
GASNet	Global Address Space Networking
GPGPU	General Purpose computing on Graphics Processing Unit
HPCS	High Productivity Computing Systems
MIC	Many Integrated Cores
MIMD	Multiple Instruction Multiple Data
MPI	Message Passing Interface
OpenCL	Open Computing Language
PGAS	Partitioned Global Address Space
PPL	Pervasive Parallelism Laboratory
SCC	Single-chip Cloud Computer
SIMD	Single Instruction Multiple Data
SPMD	Single Program Multiple Data
UPC	Unified Parallel C

참고문헌

[1] P. Kogge et al., "ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems," DARPA, Sept. 2008.

[2] S. Amarasinghe et al., "ExaScale Software Study: Software Challenges in Extreme Scale Systems," DARPA, Sept. 2009.

[3] L.V. (Sanjay) Kale, "Programming Models at Exascale: Adaptive Runtime Systems, Incomplete Simple Languages, and Interoperability," *Int. J. High Perform. Comput. Appl.*, vol. 23, no. 4, Nov. 2009.

[4] B. Carlson, "Introduction to the PGAS Model," IDA Center for Computing Sciences, 2003.

[5] Calin Cascaval, "HPC Challenge Class 2 UPC and X10 on Multiple Platforms," IBM, 2008, p. 5.

[6] K.J. Brown et al., "A Heterogeneous Parallel Framework for Domain-Specific Languages," *IEEE Int. Conf. Parallel Architectures Compilation Tech.*, 2011, pp. 89-100.

[7] G. Diamos and S. Yalamanchili, "Harmony: An Execution Model and Runtime for Heterogeneous Many Core Systems," *ACM/IEEE Int. Symp. High Perform. Distrib. Comput.*, Hot Topics Session, June, 2008.

[8] J. Lee et al., "COMIC++: A Software SVM System for Heterogeneous Multicore Accelerator Clusters," *IEEE Int. Symp. High Perform. Comput. Architecture*, Bangalore, India, Jan. 2010, pp. 329-340.

[9] T.G. Mattson et al., "The 48-core SCC Processor: The Programmer's View," *Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2010, pp. 1-11.

[10] J. Kim et al., "SnuCL: an OpenCL Framework for Heterogeneous CPU/GPU Clusters," *Proc. 26th Int. Conf. Supercomputing (ICS)*, San Servolo Island, Venice, Italy, June 2012.

[11] J. Dongarra et al., "The International Exascale Software Project Roadmap," *Int. J. High Perform. Comput. Appl.*, vol. 25, no. 1, 2011.