특집 08

에너지 효율 제고를 위한 퍼스널 슈퍼 컴퓨터와 그 프로그래밍 수단

목 차

최동훈 · 홍정우 · 이홍석 · 윤상윤
(한국과학기술정보연구원)

1. 서 론

슈퍼 컴퓨터에 대한 가장 중요한 이슈 중의 하나가 슈퍼 컴퓨터에서 프로그램을 실행하는 데 필요한 전력 소비 문제이다. 슈퍼 컴퓨터 500[1]에 의하면, 현재 전세계에서 가장 빠른 로스 알라모스 국립 연구소의 Road Runner 슈퍼 컴퓨터의 경우 소비 전력은 2.5 MW에 달하며, 그 다음으로 빠른 오크리지 국립 연구소의 Jaguar 슈퍼 컴퓨터는 7MW의 전력을 소비한다[2]. 톱 10 랭킹을 들여다 보면 슈퍼 컴퓨터의 성능이 높다고 해서 소비 전력이 높은 것이 아님을 알 수 있다. 이러한 결과는 단일 코어의 CPU보다 계산 성능과 에너지 효율이 뛰어난 멀티 코어 CPU와 엑셀러레이터의 발전에 힘입은 바가 크다고 볼 수 있다.

대표적인 멀티 코어 칩으로는 Intel의 Xeon과 AMD의 Opteron 등이 있으며, 엑셀러레이터로 IBM의 Cell 프로세서, NVIDIA나 Intel의 GPU, FPGA(field programmable gate array) 등이 있다. Road Runner는 AMD의 Opteron이라는 멀티코어 칩과 IBM의 Cell 프로세서를 장착하고 있고, Jaguar는 Opteron으로 구축되었다. 이들 멀티코어 칩과 엑셀러레이터는 일반 CPU보다 계산 성능이 월등할 뿐만 아니라, 에너지 효율이 매우 높고 가격이 비교적 낮아 대규모의 슈퍼 컴퓨터에 이미 장착되어 널리 사용되고 있다.

2007년부터 멀티 코어 칩을 사용하여 베오울프(Beowulf) 클러스터를 여행 가방 크기로 축소한 마이크로울프(Microwulf)[3]가 퍼스널 슈퍼 컴퓨터로 개발되었다. 이후, SGI, Supermicro, NVIDIA 등은 엑셀러레이터와 멀티 코어를 기반으로 데스크 탑이나 타워형의 퍼스널 슈퍼 컴퓨터 제품군을 출시하고 있다. 이들은 매우 높은 계산 성능과 에너지 효율로 이미 과학 기술자들에게 많이 알려져 있으며, 블룸버그는 증권 정보의 신속한 제공[4]을 위해 이러한 시스템을 사용하고 있다.

엑셀러레이터와 멀티코어 CPU로 구성된 퍼스널 슈퍼 컴퓨터는 프로그래밍이 매우 복잡하다. 이들의 아키텍처가 서로 다르기 때문에 인스트럭션 집합이 서로 다르다. 이것은 사용자가 엑셀러레이터와 CPU의 성능을 극대화하기 위해 각각 다른 프로그램을 작성해야 한다는 것을 의미

한다. 물론 사용자는 고급 언어로 프로그램을 작성할 수도 있으나, 엑셀러레이터를 위한 컴파일러의 지원이 제한적이거나 정교하지 못한 경우도 있다. 이러한 이유로, 퍼스널 슈퍼 컴퓨터를 위한 어플리케이션의 개발은 여전히 하드웨어에 특화된 인터페이스에 의존적이라서, 개발 생산성도 매우 낮을 수 밖에 없다.

뿐만 아니라, 이들 사용자는 코어수의 증가에 따라 자신의 어플리케이션을 병렬화하는 문제를 해결해야 한다. 그러나 일반적인 사용자는 순차적 프로그래밍에 익숙해 있기 때문에 병렬 프로그래밍에 어려움을 겪을 수 밖에 없다. 이러한 문제를 해결하기 위해 기술력이 부족한 사용자도 사용하기 쉬운 병렬 프로그래밍 모형과 실행 시스템을 사용자에게 제공하여 기존의 레거시 코드의 사용을 돕거나, 사용자 친화적인 병렬 프로그램 개발 환경[5]을 제공하는 것이 중요하다. 현재 사용자는 각 하드웨어 벤더가 제공하는 병렬 프로그래밍 수단으로 Open MP[6]를 비롯하여 Intel의 TBB(task building block)[7], NVIDIA의 CUDA[8] 등을 사용하고 있다. 이 밖에 대학교나 연구소에서 개발한 프로토타입으로 Mars[9], Merge[10] 등을 통해 레거시 코드의 병렬 처리를 수행할 수 있다.

이 글에서는 저전력 소비형 멀티코어 칩과 엑셀러레이터의 발전에 따라 사용자에게 보급 증가 추세에 있는 에너지 효율이 높은 퍼스널 슈퍼컴퓨터에 대해 살펴 보고자 한다. 퍼스널 슈퍼컴퓨터의 구조, 성능과 에너지 효율, 퍼스널 슈퍼컴퓨터의 활용 극대화에 가장 큰 장벽인 병렬 프로그래밍 문제, 이의 해결을 위한 사용자 편의적인 프로그래밍 수단에 대해 소개고자 한다. 2장에서 멀티코어 칩에 대한 소비 전력의 효율성과 이를 성취하기 위한 하드웨어 벤더의 시도를 서술하고, 3장에서 특화된 멀티 코어 칩을 장착한 상용 퍼스널 슈퍼 컴퓨터와 그의 에너지 효율에 대해 서술한다. 4장에서 이러한 퍼스널 슈퍼 컴퓨터에서 어플리케이션의 성능을 극대화하기 위해 사용자가 해결해야 할 병렬 프로그래밍에 대해 서술한다. 5장에서 이 문제의 해결 수단으로 과학 계산 문제를 위한 병렬 프로그래밍 모형을 설명하고, 6장에서 끝맺는다.

2. 멀티 코어 칩: 계산 성능과 에너지 효율 제고의 열쇠

지금까지 컴퓨터 아키텍트는 성능을 개선하기 위해, 하나의 칩에 코어 수를 증가시키는 것과 클락 속도를 증가시키는 것을 동시에 연구해 왔다. 이들 두 가지 요소의 조합은 새로운 칩이 출시될 때마다 상당한 성능 발전을 보여 주었다. 현재 칩당 코어의 수는 1.5년에서 2년마다 2배로 증가하고 있으며 이러한 현상은 지속될 것으로 보인다. 반면, 클락 속도는 더 이상 증가하지 않고 거의 유사하게 유지되고 있다[11].

(그림 1)에서 실선은 칩의 코어 수를 나타내고 있으며 앞으로도 지수적으로 증가할 것임을 보여준다. 점선은 클락 속도를 나타내며, 몇 년 전까지 지수적인 증가를 보여왔으나 개발 비용 대비 성능 향상 효과가 비교적 약하기 때문에 작년부터 현상 유지가 되거나 개선의 가능성이 낮다. 사실 소비 전력 대비 성능의 개선이라는 측면에서 보면 클락 속도는 약간의 감소를 예상할 수 있다.

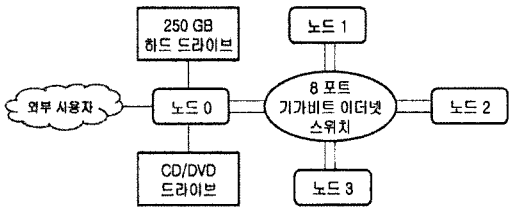
(그림 1) 클락 속도와 코어수의 증감 추세

이제 왜 칩의 성능과 에너지 효율을 높이기 위해 멀티 코어 칩을 개발하는지 그 이유를 살펴보자. 프로세서의 소비 전력은 클락 속도의 3제곱에 비례한다고 알려져 있다[2]. 아키텍트가 클락 속도와 코어 수 증가를 통해 프로세서의 성능을 50% 올리고자 노력한다고 하자. 클락 속도를 50% 증가시킨다면, 칩의 소비 전력은 1.5의 세제곱, 즉, 3.375배로 증가한다. 반면, 코어 수를 하나 더한다면, 클락 속도를 75%로 줄이면서 성능을 50% 증가시킬 수 있다. 이 경우, 클락 속도가 75%로 줄었기 때문에, 소비 전력은 2 * 0.75의 세제곱, 즉 0.845배로 떨어진다. 이와 같이 멀티코어 칩을 사용하는 것이 클락 속도를 높이는 것보다 에너지 효율 측면에서 대단히 유리하다는 것을 알 수 있다.

멀티 코어 프로세서는 전통적인 CPU 형태의 동일한 코어를 여러 개 집적한 것과 엑셀러레이터와 CPU를 섞어서 구성한 것이 있다. 후자를 이질적 멀티 코어라고 부르는데, 이것은 동일한 소비 전력에 대해 많은 계산을 실행할 수 있기 때문에 계산 성능과 에너지 효율을 동시에 올리는 효과를 나타낸다. IBM Cell 프로세서는 게임을 위한 특수 목적으로 개발된 이후 빠른 계산을 요구하는 어플리케이션을 위한 프로세서로 IBM이 제작하는 대규모 슈퍼 컴퓨터에서 많이 사용되어 왔다. 최근 들어, 그래픽 어플리케이션을 위해 NIVIDIA가 개발한 GPU는 탁월한 수학 계산 능력을 강점으로 슈퍼 컴퓨터에 널리 장착되고 있는 추세에 있다. 특히, NVIDIA의 Tesla는 퍼스널 슈퍼 컴퓨터의 계산 전용 프로세서로 활용되고 있으며 4 개까지 장착이 가능하다. Intel, AMD 등도 가능한 한 칩에 많은 코어를 집적시키고 있으며, 가까운 미래에 매니 코어(many-core) 코어 프로세서를 출시하기 위해 안간힘을 쓰고 있다. 이 중에서 Intel이 개발 중인 라라비(Larrabee)는 전통적인 CPU 성격의 코어와 GPU 성격의 코어를 하나의 칩에 장착한 매니 코어 칩으로, 두 가지 유형의 어플리케이션 모두에 강점을 보일 것으로 예상된다. 동일한 코어로 구성된 멀티 코어 칩의 속도는 GPU, Cell, FPGA를 포함하는 이질적인 멀티 코어 프로세서에 비해 성능이 극적으로 증가하지는 않지만, 엑셀러레이터에 대한 계산을 별도로 고려하지 않아도 되기 때문에 병렬 프로그래밍의 용이함에서 큰 장점을 갖는다.

3. 퍼스널 슈퍼 컴퓨터의 성능과 에너지 효율

퍼스널 슈퍼 컴퓨터는 마이크로울프부터 시작되었다. 마이크로울프는 베오울프형 클러스터를 여행 가방의 크기(28cm*30cm*43cm)와 무게(14kg)로 축소한 데서 유래한다. 이렇게 크기는 축소되고 가격 대비 성능은 올라가게 되었던 것은 AMD의 멀티 코어 칩과 이더넷 기가비트 스위치의 속도 덕분이라고 할 수 있다. (그림 2)는 마이크로울프의 구조를 보여준다.



(그림 2) 마이크로울프의 구조

2007년 1월에 구축된 마이크로울프의 성능은 26Gflops에 달하는 반면, 구축 비용은 2,500불도 되지 않고, 가격 대비 성능은 96불/Gflops였다. 같은 해 2007년 8월에 구축된 두 번째 마이크로울프는 성능은 그대로지만, 구축 비용이 1,256불로 떨어져, 가격 대비 성능은 48불/Gflops였다. 여기서 flops는 슈퍼 컴퓨터의 성능을 표시하는 단위로 초당 부동점 소수의 계산 수를 의미한다. 슈퍼 컴퓨터의 성능은 표준 벤치마크를 실행하여 측정하는데, 가장 많이 쓰이는 것이 Linpack이다. 마이크로울프의 성능은 Linpack을 실행한

결과에 의한 것이다.

이제 마이크로울프의 에너지 효율을 살펴 보자. 일반적으로 슈퍼 컴퓨터의 계산 능력에 관한 에너지 효율은 소비 전력에 따른 단위 시간당 부동점 소수 계산 비율을 나타내는 W/Gflops (1Gflops 계산에 드는 소비 전력)로 표시된다. 마이크로울프는 계산 중에 450W를 소비하며, 계산 성능이 26Gflops이므로, 에너지 효율은 17.3W/Gflops이다. 이것을 2002년도에 전력 소비를 최소화하기 위해 로스 알라모스 국립 연구소에서 구축한 대규모 슈퍼 컴퓨터인 그린 데스티니(Green Destiny)[13]와 비교해 보자. 그린 데스티니는 3.2kW를 소비하며, 101Gflops의 계산 성능을 나타내므로, 에너지 효율은 32W/Gflops정도에 불과하다. 여기서 우리는 그린 데스티니보다 마이크로울프가 에너지 효율이 높다는 것을 알 수 있다. 물론, 이것은 모든 어플리케이션을 퍼스널 슈퍼 컴퓨터로 계산하는 것이 좋다는 것을 의미하지는 않는다. 사용자는 자신이 풀고자 하는 문제의 규모가 크면, 계산 성능이 대단히 높은 대규모 슈퍼 컴퓨터를 쓰는 것이 적합하다.

NVDIA의 테슬라(Tesla) 퍼스널 슈퍼 컴퓨터의 성능은 소비 전력 600W에 170.24Gflops로, 에너지 효율은 3.5 W/Gflops이다. 이 시스템에 GPU 2개를 추가하면 시스템 전력이 880W로 증가되지만 성능은 1,996Gflops로 증가하므로, 에너지 효율은 0.44W/Gflops로 엄청나게 좋아진다. 이것은 GPU 테슬라 엑셀러레이터가 슈퍼 컴퓨터의 에너지 효율에 얼마나 크게 기여하는지를 보여주는 단적인 예다.

마이크로울프는 리눅스 운영체제를 사용하고 있으며, 병렬 프로그래밍을 위한 수단으로 베오울프와 마찬가지로 클러스터 컴퓨팅에서 널리 사용되는 Open MPI와 PVM(parallel virtual machine)을 라이브러리로 제공한다. 아울러 이들 라이브러리를 사용하여 과학 문제의 해결을 위해 작성된 어플리케이션도 제공한다. 이들 어플리케이션은 다양한 CFD(computational fluid dynamic) 코드, DNA 서열 분석을 위한 계통수(phylogenetic tree) 계산을 위한 DPMA, 유한요소 해석을 위한 Adventure와 ParaFEM, 프리에 변환을 위한 FFTW, 분자 동력학 시뮬레이션을 위한 GROMACS와 NAMD, 유전자 서열 비교를 위한 mpiBLAST 등이 있다.

4. 퍼스널 슈퍼 컴퓨팅의 장벽: 병렬 프로그래밍 문제

멀티코어 칩의 발전은 하드웨어 신제품의 출시에 따른 소프트웨어의 변화에 막대한 영향을 주고 있다. 즉, 신제품으로 출시된 칩의 클락 속도는 구제품과 거의 같겠지만, 칩에 장착된 코어의 수는 당연히 증가하게 된다. 따라서 사용자는 기존의 코어 수만을 사용하여 자신의 어플리케이션을 실행할 경우 성능 개선을 기대할 수 없고, 추가된 코어까지 활용해야만 어플리케이션의 성능 개선을 기대할 수 있다. 이러한 이유로 마이크로소프트를 비롯한 소프트웨어 회사는 기존의 소프트웨어 제품을 병렬화하는 데에 따르는 엄청난 부담을 안고 있다. 만일 기존 소프트웨어 제품이 코어 수의 증가를 인식하지 못하면, 사용자는 하드웨어 신제품에 부적합한 소프트웨어를 구입하지 않을 것이기 때문이다. 역으로, 사용자는 자신의 어플리케이션의 성능 향상을 기대할 수 없다면 하드웨어 신제품을 사지 않을 것이다. 결국 에너지 효율이 높은 멀티 코어 칩에 의한 컴퓨터 아키텍처의 발전이 소프트웨어 시장 전체에 영향을 미치는 중요한 요소가 된 것이다.

멀티 코어 프로세서 상에서 소프트웨어를 효율적으로 실행하려면 여러 코어를 동시에 이용할 수 있도록, 사용자는 소프트웨어를 병렬 프로그램으로 개발해야 한다. 이러한 병렬 프로그램은 병렬 프로그래밍 언어로 작성될 수도 있으나, 일반 사용자는 순차적 프로그래밍에 익숙하기

때문에 병렬 언어에 어려움을 느낀다. 전통적인 병렬 프로그래밍 기법은 순차적 프로그래밍에 익숙한 일반 프로그래머에겐 매우 귀찮은 일을 포함하고 있다. 프로그래머는 동시성의 명시적 관리, 데이터 국지성의 수동 관리, 어플리케이션의 포팅에 따른 코드의 재튜닝 등을 신경 써야 한다. 동시성의 명시적 관리를 위해, 프로그래머는 쓰레드의 생성, 메시지 및 lock을 통한 쓰레드 간의 동기화 등을 프로그램에 표현해야 한다. 데이터 국지성의 수동 관리로 인해, 알고리듬이 조금만 어려워 져도 올바르고 확장성 있는(scalable) 병렬 코드를 작성하는 어려움은 훨씬 증가한다. 어플리케이션을 다른 시스템이나 대규모 시스템에 포팅할 때 프로그래머는 코드를 재튜닝해야 하는 경우가 자주 발생한다.

이와 같이 귀찮은 일을 수반하는 병렬 코딩을 단순화하기 위해 MPI나 Open MP와 같은 방법이 사용되어 왔으나, 이것도 그리 만만한 일은 아니다. 순차적 프로그래밍에 익숙한 일반 개발자를 위한 병렬 프로그래밍 모형과 이를 효율적으로 수행할 수 있는 실행 시스템을 개발할 필요가 있다. 병렬 프로그래밍 모형은 동시성과 국지성을 고급 언어로 쉽게 명세화할 수 있어야 하고, 실행 시스템은 시스템의 특성이나 규모에 상관없이 컴파일, 쓰레드 생성 및 동기화, 스케줄링, 자원 관리, 부하 균형, 결함 허용 등의 문제를 자동으로 해결할 수 있어야 한다. 이를 위한 방법으로 구글에서 제안한 MapReduce를 멀티코어에 적합하도록 개발하기도 한다. 그러나 MapReduce는 데이터 병렬 처리에 근거하기 때문에 타스크 병렬 처리를 성공적으로 수행하지 못하는 문제가 있다. NVIDIA는 GPU 상에서 병렬 프로그래밍을 위해 CUDA를 통한 C 언어와 디버거를 제공하며, Intel은 TBB(Thread Building Block)이라는 라이브러리 방식의 병렬 프로그래밍 모형과 C를 확장한 병렬 프로그래밍 언어 Cilk[14]를 제공한다.

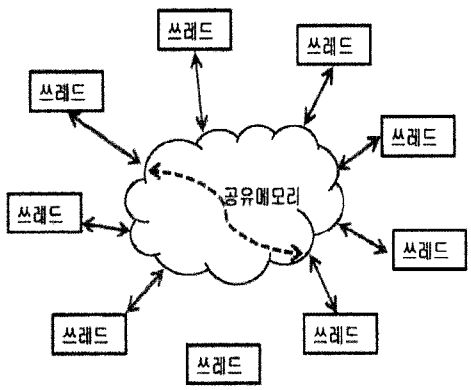
5. 과학 계산을 위한 병렬 프로그래밍 모형

이 장에서는 멀티코어 프로세서를 위한 병렬 프로그래밍 모형을 소개한다. 과학 어플리케이션에서 주로 사용되고 있는 Open MP와 요즘 많은 사람들의 관심을 끌고 있는 GPU를 위한 NVIDIA의 CUDA를 소개한다. 아울러, Cell 프로세서를 위한 MapReduce와 엑셀러레이터를 장착한 프로세서에서 과학 어플리케이션에 일반적으로 사용될 수 있는 Merge를 소개한다.

5.1 Open MP

OpenMP는 공유메모리 환경에서 다중 쓰레드 병렬프로그램 작성을 위한 응용프로그램 인터페이스 (API)이다. OpenMP의 역사를 보면, 1990년대 초에 고성능 공유메모리 시스템의 비약적 발전으로 1996년 openmp.org를 설립하여 1997년에 OpenMP API 1.0을 발표하였다. 이후 몇 차례 버전이 업그레이드 되었으며, 2009년 현재 OpenMP 3.0 버전으로 발전을 하였다.

OpenMP 병렬프로그램의 핵심은 (그림 3)에서 보듯이 각각의 다중 쓰레드들이 공유메모리 환경에서 같은 주소의 데이터를 공유한다는 것이다.



(그림 3) 공유메모리와 OpenMP 프로그램 개념도

OpenMP를 이용한 병렬화 기법은 순차코드에 컴파일러 지시어를(예를 들면, !$ OMP PARALLEL DO) 삽입하면 컴파일 시 다중 쓰레드 코드를 생성하여 병렬화 작업이 수행된다. 공유메모리 환경에서 설정해야 하는 환경변수로는 시스템 아키텍처에 의한 최대 사용 가능한 쓰레드 개수(Nth)이며, 다음과 같이 지정한다.

export OMP_NUM_THREADS=Nth.

〈표 1〉 칩당 최대 쓰레드 수

제조회사/ 시스템	AMD/ Barcelona	Intel/ Nehalem	IBM/ POWER5
쓰레드 개수 (Nth)	16	8	64/노드

〈표 1〉은 KISTI 슈퍼컴퓨팅본부에서 서비스하고 있는 최신 슈퍼컴퓨터와 각 시스템에서 사용 가능한 최대 쓰레드 개수(Nth)를 보여준 것으로, IBM POWER6 p595+의 경우 OpenMP로 병렬화 하면 최대 64배의 성능을 쉽게 얻을 수 있다.

5.2 Cell 프로세서를 위한 MapReduce

Cell 프로세서는 분산 메모리 구조로 되어 있기 때문에 Cell 프로세서를 위한 MapReduce는 구글의 MapReduce와 상위 수준의 설계가 유사하다. 그러나 멀티코어 상에서 MapReduce를 수행하기 때문에 작업의 크기를 볼 때 Cell 프로세서를 위한 MapReduce는 Phoenix[15]와 유사하다. 구글의 MapReduce에서 데이터 분할의 Sorting과 Reduce는 동일한 노드에서 수행되지만, Cell 프로세서를 위한 MapReduce는 부하 균등을 실현하기 위해 Sorting과 Reduce를 다른 코어에서 수행한다. 이와 같은 기법은 Cell 프로세서의 작은 로컬 메모리, 빠른 계산, 충분한 대역폭을 감안할 때 매우 합리적인 방식이다. Phoenix에서 MapReduce 작업은 다른 코어 간에 분산되고 하드웨어 캐싱은 국지성을 이용한다. 키에 근거한 그루핑(grouping)을 위해, 출력 키에 해싱을 적용하고 키와 값에 대한 포인터를 이진 탐색 트리에 삽입하므로, Sorting과 메모리 복사가 필요 없다. 그러나, 소프트웨어에 의한 메모리 관리를 이용하여 그래프 구조를 처리하는 것은 매우 복잡하기 때문에 Phoenix의 이 기법을 Cell 프로세서에 적용할 수는 없다. 따라서 Cell 아키텍처의 강점을 이용하여, MapReduce 실행 시스템의 구현에 필요한 새로운 기법으로 PPE를 통한 전역적 통제, DMA 전송과 계산의 병렬화 등을 개발하였다.

Cell 프로세서의 MapReduce 실행 시스템은 멀티 쓰레드 방식으로 주로 PPE(power processing element)에서 동작하고, Map과 Reduce 함수는 여러 SPE(synergistic processing element)로 매핑되어 동작한다. PPE 실행 시스템은 서로 다른 SPE에 대한 작업을 스케줄하며, 데이터 구조의 관리와 메모리 관리를 수행한다. SPE의 주요 임무는 사용자 코드를 실행하고 그루핑 단계를 일부 실행한다. 그루핑은 분할과 정렬의 2 단계 과정을 거쳐 달성되는데, 분할은 PPE가, 정렬은 SPE와 PPE가 수행한다.

Cell 프로세서를 위한 MapReduce의 C 프로그램을 위한 API는 다음과 같다.

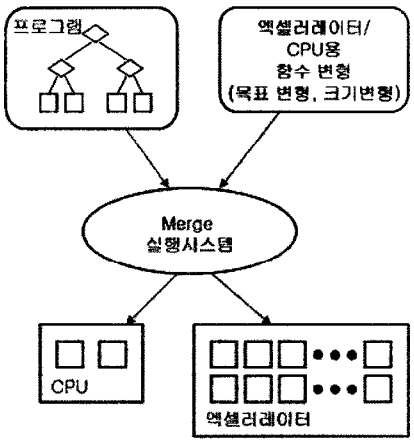
```
void MapReduce_exec (MapReduce_Specification specification);
                        // MapReduce 실행 시스템의 초기화 //
void MapReduce_emitIntermediate (void **key, void **value);
                        // 사용자가 정의한 Map 함수 호출 //
void MapReduce_emit (void **value);
                        // 사용자가 정의한 Reduce 함수 호출 //
```

5.3 Merge

Merge는 이질적 멀티코어 프로세서를 위한 고급 병렬 프로그래밍 모형으로, 컴파일러와 실행 시스템을 제공한다. Merge는 어플리케이션을 기본 함수의 집합으로 매핑한다. 이 함수 집합은 사용자에 의해 확장 가능하며 EXOCHI

와 도구에 의해 생성된 기본 함수의 형태를 갖는다. Merge는 EXOCHI를 통해 엑셀러레이터 상에서 범용적 계산을 지원할 뿐만 아니라, 사용 가능한 모든 코어를 이용하기 위해 기본 함수를 동적으로 선택하고 분배하여 성능 향상을 이룩함으로써 이질적 멀티코어 시스템의 활용을 극대화한다.

(그림 4)에서 볼 수 있듯이, Merge 어플리케이션은 원래의 계산을 더욱 작은 연산으로 쪼개서 생성된 함수의 계층적 집합으로 표현되며, 계층적 집합은 이들 함수에 대한 병렬 처리의 표현과 데이터 분할의 서술을 포함한다.



(그림 4) Merge 실행 시스템의 개념도

예를 들어 K-means 클러스터링 알고리듬을 Merge 모형을 사용하여 실행한다고 하자. K-means 알고리듬은 데이터의 집합과 초기 클러스터의 중심이 주어졌을 때, 각 데이터와 초기 클러스터의 중심 간의 거리를 계산하여 최소 거리에 있는 클러스터에 소속시킨다. 다음 반복 수행 과정을 위해, 주어진 클러스터에 속한 모든 데이터를 평균하여 각각의 클러스터에 대한 새로운 중심을 계산한다. 이후 두 과정을 반복 수행하면, 최소 거리에 근거한 클러스터를 찾아 내게 된다.

이 과정에서 데이터와 클러스터 중심 간의 거리를 계산 할 때, 여러 크기의 데이터 분할이 가능하고 이를 위한 함수의 정의가 가능하다. Merge 어플리케이션에서 병렬 계산의 크기와 양은 어떤 크기의 데이터 분할을 선택하느냐에 달려 있다. 이렇게 다양한 크기의 분할에 대한 함수 정의를 크기 변형(granularity variants)라고 한다. 계층 구조의 어느 수준에서든 다양한 크기의 변형을 정의하고 실행할 수 있기 때문에, 어플리케이션은 이질적인 여러 엑셀러레이터에 대해 이미 다양하게 구현된 기존의 라이브러리 API를 이용하여 실행하거나, 엑셀러레이터에 특정된 고유 함수로 확장되어야 할 새로운 API를 정의한다. 따라서 함수의 호출은 분할을 계속할 것인가 또는 알고리듬의 병렬 계산을 특정 엑셀러레이터에 매핑하여 해당 고유 함수를 실행할 것인가 간의 선택이다. 여기서 특정 엑셀러레이터를 위해 구현된 각 함수를 목표 변형(target variant)라고 부른다.

다양한 크기 변형과 목표 변형은 함수에 대한 풍부한 구현 사례를 제공하기 때문에, Merge 컴파일러와 실행 시스템은 어플리케이션 성능과 효율을 최적화할 수 있도록 구현된 함수를 선택할 수 있다. 최적 함수 변형의 선택은 다음과 같이 이루어 진다. 첫째, 주어진 시스템에서 사용 가능한 코어에 적합한 목표 변형으로 제한하고, 둘째, 사용 가능한 코어 간에 작업을 분배하는데 적합한 크기 변형을 선택한다. 끝으로 주어진 함수와 아키텍처에 대한 최적의 모수를 선택하여 성능을 최적화 한다.

Merge 프레임워크의 API는 다음과 같다.

```
bundle dp0 : public sse { void dp(…); };
                    // 함수 변형 dp0를 정의하여 sse
                       그룹의 요소로 소속시킴 //
mapreduce (int i=0; i < n; i++) {…}
                    // {…} 속의 함수를 실행하고,
                       reduce를 실행할 수도 있음 //
```

5.4 CUDA

CUDA는 C/C++같은 고급언어를 통해 프로그래밍을 가능하게 하며, GPU 전용 컴파일러가 해당 부분의 코드를 위해 따로 호출된다. CUDA는 스케줄러에 의한 쓰레드 스케줄링을 제공하고 syncthread라는 컴파일러 고유의 함수를 제공함으로써, 사용자는 병렬 프로그래밍을 위한 적은 노력으로 쓰레드 간의 blocking 문제를 해결할 수 있다. 또한 CUDA 컴파일러가 만들어내는 PTX(Parallel Thread eXcution)코드가 가상코드의 형태를 취하도록 하여 추후에 출시되는 GPU에서도 수행될 수 있게 한다. 물론, 컴파일러가 프로그램의 특성까지 고려할 수는 없기 때문에, 새로운 GPU의 성능 향상에 따른 프로그램의 성능 향상을 기대하기 어렵다. 이러한 이유로, 최적의 성능을 얻기 위해서는 새로운 하드웨어가 발표될 때마다 최적화 작업을 해 주어야 한다.

CUDA의 쓰레드는 단위 쓰레드와 쓰레드 블럭, 쓰레드 블록 그리드로 모델링 되며, 각각 per-thread private, per-block shared, per-application global 메모리가 정의된다. GPU 내부의 그리드를 구성하는 각 쓰레드 블록 안에는 하나의 쓰레드마다 블록 내에 고유한 ID와 프로그램 카운터, 레지스터, per-thread private 메모리와 Global 메모리로부터 입출력 버퍼 등이 할당된다. 각 블록내의 쓰레드는 블록 내에서 동기화와 공유메모리를 통해 통신한다. 각 쓰레드에 할당된 private 메모리는 레지스터 내용의 저장, 함수 호출, 배열 변수 등을 위해 사용된다.

전통적인 C 코드와 CUDA C 코드의 예를 들면 다음과 같다.

```
void serial_s(int n, float a, float *x, float *y)  // C 코드 //
{
    for(int i=0; i<n; ++i)
        y[i]= a * x[i] +y[i];
}

serial_s(n, 2.0, x, y)                              // C 호출 //

void cuda_s(int n, float a, float *x, float *y)     // CUDA 코드 //
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if( i<n ) y[i] = a*x[i] + y[i];
}

int nblocks =(n + 255)/256;                         // CUDA 호출 //
cuda_s <<< nblocks, 256>>>(n, 2.0, x, y)
```

전통적인 C의 경우와 달리, CUDA의 함수 호출은 블럭마다 256 쓰레드를 실행한다. 이때 데이터를 256개의 쓰레드에 나누도록 nblock 값을 설정한 후, 256개 쓰레드를 동시에 호출 하여 실행한다.

6. 요약 및 향후 전망

지금까지 저전력 소비형 멀티코어 칩과 엑셀러레이터의 발전에 따라 사용자에게 보급 증가 추세에 있는 퍼스널 슈퍼 컴퓨터의 구조, 성능과 에너지 효율의 우수성, 그리고 그의 활용 극대화를 위해 사용자에게 제공되어야 할 병렬 프로그래밍 수단에 대해 살펴 보았다.

퍼스널 슈퍼 컴퓨터는 2007년 초에 세상에 알려진 이후, 지속적인 발전을 거듭하여 가격은 점차 내려가고 성능은 올라가고 있다. 엑셀러레이터 개발을 주도하고 있는 NVIDIA는 GPU 테슬라를 탑재한 퍼스널 슈퍼 컴퓨팅 시장을 공략하고 있다. 테슬라는 기존의 GPU에 비해 가격 대비 성능, 에너지 효율이 월등히 높아졌다. 또한 Intel은 CPU와 GPU 특성을 동시에 제공하는 매니 코어 칩 라라비를 출시하여 퍼스널 슈퍼 컴퓨

터 시장에서 NVIDIA와 경쟁하고 있다. 현재 테슬라를 탑재한 데스크 탑이 수백 만원대의 가격에 출시되어 과학기술자를 비롯한 사용자의 주목을 받고 있으며, Intel 라라비를 탑재한 데스크 탑도 출시되어 가격은 더욱 떨어질 것이다. 이렇게 되면, 높은 계산 성능을 요구하는 과학 기술자, 게임을 즐기는 일반 사용자를 중심으로 향후 2~3년 내에 본격적인 퍼스널 슈퍼 컴퓨팅 시대가 도래할 것이며, 아울러 소프트웨어 시장 역시 기존의 어플리케이션의 성능을 향상시키기 위한 노력에 박차를 가할 것으로 전망된다.

참고문헌

[1] www.top500.org

[2] J. Dongarra, 'An Overview of High Performance Computing and Challenges for the Future,' HPC Asia 2009

[3] http://www.calvin.edu/~adams/research/microwulf/

[4] http://www.wallstreetandtech.com/advancedtrading/showArticle.jhtml?articleID=220200055#

[5] K. Asanovic, et.al., 'The Parallel Computing Laboratory at U.C. Berkeley: A Research Agenda Based on the Berkeley View,' EECS Department, University of California, Berkeley, Technical Report No. UCB/EECS-2008-23, 2008

[6] www.openmp.org/

[7] http://www.threadingbuildingblocks.org/

[8] http://www.nvidia.com/object/cuda_home.html

[9] B. He, et.al., 'Mars: A MapReduce Framework on Graphics Processors,' PACT 2008

[10] M. Linderman, et.al., 'Merge: a programming model for heterogeneous multi-core systems,' ASPLOS 2008

[11] K. Yelick, 'How to Waste a Parallel Computer,' ISCA 2009

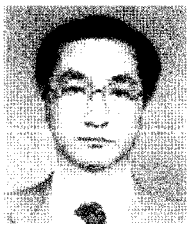
[12] http://kr.nvidia.com/object/product_tesla_m1060_kr.html

[13] http://sss.lanl.gov/presentations/041015-IEEE-DVP.pdf

[14] http://supertech.csail.mit.edu/cilk/

[15] http://mapreduce.stanford.edu/

저자약력



최 동 훈

1981년 2월 서울대학교 계산통계학과 졸업(학사)
1983년 2월 한국과학기술원 전산학과 졸업(석사)
1989년 6월 Northwestern University 전산학과 졸업(박사)
1983년 2월~1986년 8월 한국증권전산㈜ 과장대리
1989년 8월~1992년 2월 한국국방연구원 선임연구원
1992년 3월~1999년 2월 동덕여자대학교 부교수
2005년 2월~현재 한국과학기술정보연구원 책임연구원
관심분야 : 데이터베이스, 병렬 처리
이 메 일 : choid@kisti.re.kr

홍 정 우

1995년 경북대학교 컴퓨터 공학과 석사
1994년 KIST부설 SERI 슈퍼컴퓨팅센터 연구원
1998년 한국전자통신연구원 슈퍼컴퓨팅센터 연구원
1999년 연구개발정보센터 슈퍼컴퓨팅사업단 연구원
2001년~KISTI 슈퍼컴퓨팅본부 선임연구원
이 메 일 : jwhong@kisti.re.kr

이 홍 석

1997년 8월 서강대학교 물리학과 졸업(박사)
1997년 11월 포항공과대학교(PostDoc)
1999년 12월~현재 한국과학기술정보연구원 선임연구원
2005년 6월 독일 막스프랑크 (베를린) 초청연구원
2008년 7월 미국 IBM Watson Research Center 방문연구원
2009년 9월 고려대학교 컴퓨터전파통신공학부 겸임교수
관심분야 : 슈퍼컴퓨팅, 테크니컬 컴퓨팅, GPU 병렬처리
이 메 일 : hsyi@kisti.re.kr

윤 상 윤

1983년 고려대학교 수학과(학사)
1993년 Stevens Institute of Technology 전산과 (석사)
2006년 성균관대학교 전기전자컴퓨터공학 (공학박사)
현 KISTI 기반기술개발실 책임프로젝트연구원
이 메 일 : net2nus@kisti.re.kr