제4장

병렬 구조와 데이터 연산

7

병렬 구조는 데이터 연산을 더 빠르게 수행할 수 있는 컴퓨터 구조로 꾸준히 각광받고 있다. 중앙처리장치와 그래픽 처리장치는 병렬 처리 구조를 적용시킨 대표적인 데이터 연산 장치다. 병렬 구조에서 데이터 연산은 병렬 연산이라 하며, 병렬 연산은 연산 수행 방법에 따라 분류한다. 이들은 각기 다른 연구·응용 분야에서 활용되고 있다. 본 장에서는 병렬 구조에 대한 개념과 병렬 구조에서의 데이터 연산에 대해 자세히 알아보고 병렬 구조의 미래를 전망해 본다.

1. 병렬 구조

병렬 구조는 주어진 문제를 작게 나눠 동시에 해결할 수 있도록 설계한 연산 구조다. 초기에 등장한 병렬 구조는 슈퍼컴퓨터(supercomputer)와 워크스테이션(workstation)에서 채택한 컴퓨터 구조로, 대규모 연산을 고속 처리하는 것이 목적이었다. 집적 회로 기술이 발달하면서 이후에는 개인용 컴퓨터에도 병렬 구조를 적용해 병렬 연산을 수행할 수 있게 했다. 대표적인 병렬 구조 장치는 CPU(Central Processing Unit))와 GPU(Graphic Processing Unit)다. 두 장치는 입력 데이터 집합을 복수개의 부분 데이터 집합으로 분할하고, 다수의 코어(core)와 프로세스가 분할된 부분 데이터 집합의 데이터 연산을 수행할 수 있도록 했다. 이때, 각 프로세스가 하나 이상의 스레드(thread)를 수행하게 해 자신이 담당한 데이터에 대한 연산을 수행할 수 있도록 했다. 여기서 스레드는 어떤 프로세스 내에서 실행되는 흐름의 단위다.

CPU와 GPU는 하드웨어적으로 보유한 코어의 성능과 개수에서 차이가 난다. 동일한 데이터 처리 성능을 갖는 CPU와 GPU에서 CPU의 코어 수는 GPU의 코어 수보다 월등히 적지만, CPU의 각 코어 성능은 GPU의 각 코어 성능보다 뛰어나다. 이런 특징 때문에 CPU는 관계형 데이터베이스 중에서도 관계 연산과 같이 각 코어의 성능이 중요한 데이터 마이닝 분야에서 활용하고 있다. 이와 달리, GPU는 최

• 필자 | **민준기**(한국기술교육대 컴퓨터공학부 교수)

근 주목되고 있는 딥러닝(deep learning)처럼 각 코어의 성능보다는 다수의 코어가 필요한 인공지능(artificial intelligence) 분야에서 활용하고 있다.

2. 병렬 구조에서의 데이터 연산

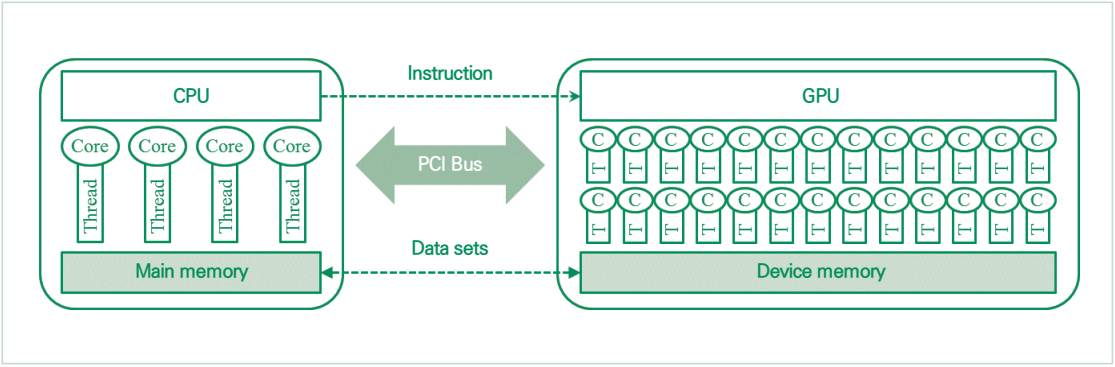
병렬 구조에서의 데이터 연산은 병렬 연산이라고 정의하는데, 연산 수행 방법에 따라 스레드 수준 병렬화, 명령어 수준 병렬화, 데이터 수준 병렬화로 분류할 수 있다.

가. 스레드 수준 병렬화

스레드 수준 병렬화는 데이터 집합을 복수 개의 부분 데이터 집합으로 분할하고, 각 스레드가 분할된 부분 데이터 집합에 대한 데이터 연산을 수행하는 방법이다. 이 방법은 다시 CPU에서 사용하는 멀티코어 연산과 GPU에서 사용하는 GPGPU(General-Purpose computing on Graphics Processing Units) 연산으로 나뉜다. 멀티코어 연산에서는 복수 개의 데이터에 대해 각 프로세스가 하나 이상의 스레드를 수행시키는데 스레드가 수행하는 데이터 연산 명령어는 서로 다를 수 있다. 이처럼 멀티코어 연산에서 데이터를 처리하는 방식을 MIMD(Multiple Instructions Multiple Data)라 한다. GPGPU 연산에서는 멀티코어 연산과는 다르게 동일한 명령어에 대해 다수의 스레드가 수행되므로, GPGPU 연산에서 데이터를 처리하는 방식을 SIMT(Single Instruction Multiple Thread)라 한다.

스레드 수준 병렬화의 단점은 스레드 간의 균등한 작업 분배가 어렵다는 점이다. 이는 다른 스레드가 자신이 담당한 데이터에 대해 모든 연산을 수행했더라도 다수의 데이터가 집중된 스레드의 데이터 연산이 종료될 때까지 다른 모든 스레드들이 대기해야 하는 문제를 야기한다. 이런 단점을 보완하려고 데이터 분포에 따른 데이터 집합의 균등 분할(data partition), 스레드 이동(thread migration), 작업 스틸링(work stealing) 등의 기법이 등장했다. 또한 스레드 수준 병렬화에서 GPGPU 연산은 하드웨어 면에서 구조적인 문제점을 갖고 있다. GPGPU 연산에서는 CPU의 메인 메모리로부터 입력된 데이터 집합이 PCI(Peripheral Component Interconnect) 버스를 통해 GPU의 디바이스 메모리로 복사된다. 이 후 GPU의 스레드가 복사된 입력 데

[그림 7-4-1] CPU와 GPU에서의 스레드 수준 병렬화 예시



이터 집합에 대해 데이터 연산을 병렬로 수행한다. 때문에 입력 데이터 집합의 크기가 과도하게 크다면, PCI 버스에서의 데이터 병목 현상이 발생할 수 있다.

최근 이런 데이터 병목 현상을 해소하기 위해, NVLink와 같이 기존 PCI 버스보다 최대 12배 빠른 고속 버스가 등장했다. 또한 CPU의 메인 메모리와 GPU의 디바이스 메모리 간의 데이터 복사를 수행하지 않고 GPU의 스레드가 직접 메인 메모리에 접근이 가능한 통합 가상 주소(UVA, Unified Virtual Addressing) 기술도 소개됐다.

나. 명령어 수준 병렬화

명령어 수준 병렬화는 명령어 파이프라인 방식을 의미한다. CPU에서 여러 명령어를 처리할 때, 명령어를 순차적으로 실행시키는 것이 아니라 여러 명령어를 명령어 파이프라인에 적재해 병렬로 실행시키는 방법이다. 예를 들어 명령어를 동시에 실행시키기 위해 각 명령어를 다섯 단계(명령어 인출, 명령어 해독, 실행, 메모리 접근, 메모리 저장)로 나눠 순차적으로 실행시킨다. 이때 현재 실행되고 있는 명령어의 한 단계가 끝나면, 현재 실행되고 있는 명령어와 종속 관계가 없는 다른 명령어가 실행돼 동시에 여러 명령어들이 실행될 수 있다.

명령어 수준 병렬화의 단점은 분기 예측 실패(branch misprediction)에 따른 과도한 CPU의 명령 주기 낭비다. 분기 예측은 어떤 조건 분기에 대해 CPU가 다음에 수행할 명령어를 미리 명령어 파이프라인에 적재해 명령어 파이프라인에 시간 공백이 생기지 않도록 하는 기술이다.

문제는 분기 예측이 실패하면 명령어 파이프라인에 이미 적재된 명령어를 모

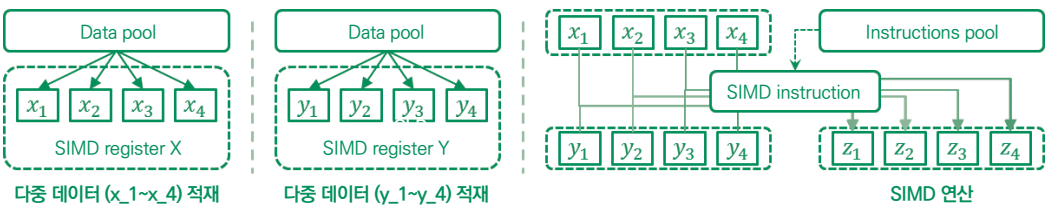
[그림 7-4-2] CPU에서의 명령어 수준 병렬화와 분기 예측 실패 예시

두 취소하고 올바른 명령어를 다시 명령어 파이프라인에 적재해야 한다는 점이다. 이때 과도한 CPU의 명령 주기를 낭비하는 문제가 발생한다. 이 문제를 해결하기 위해 어떤 조건 분기도 사용하지 않거나 조건 분기를 최소한으로 사용하는 비분기(branchless) 기법이 연구되고 있다.

다. 데이터 수준 병렬화

기존 데이터 처리 방식은 SISD(Single Instruction Single Data)로, 명령어 하나로 하나의 데이터만을 처리한다. 이에 반해 데이터 수준 병렬화는 명령어 하나로 복수 개의 데이터를 병렬로 처리하는 SIMD(Single Instruction Multiple Data) 방식이다. 즉 동일한 CPU 명령 주기를 수행하더라도 SIMD 방식은 SISD 방식에 비해 더 많은 데이터 연산을 수행할 수 있다. SIMD 방식에서는 복수 개의 데이터를 읽어와 벡터(vector)처럼 취급하고, SIMD 명령어 집합에서 존재하는 어떤 한 명령어로 다중 데이터를 병렬로 처리한다. 여기서 다중 데이터는 SIMD 레지스터라는 기억 장소에 적재된다. SIMD 레지스터의 크기는 CPU의 종류에 따라 다르다. 2018년 상반기 기준으로 그 크기가 최대 512비트이므로, 최대 정수형 데이터(32비트) 16개가 SIMD 레지스터에 적재돼 병렬로 처리할 수 있다. 인텔에서는 데이터 수준 병렬화

[그림 7-4-3] 데이터 수준 병렬화에서 SIMD 연산 예시



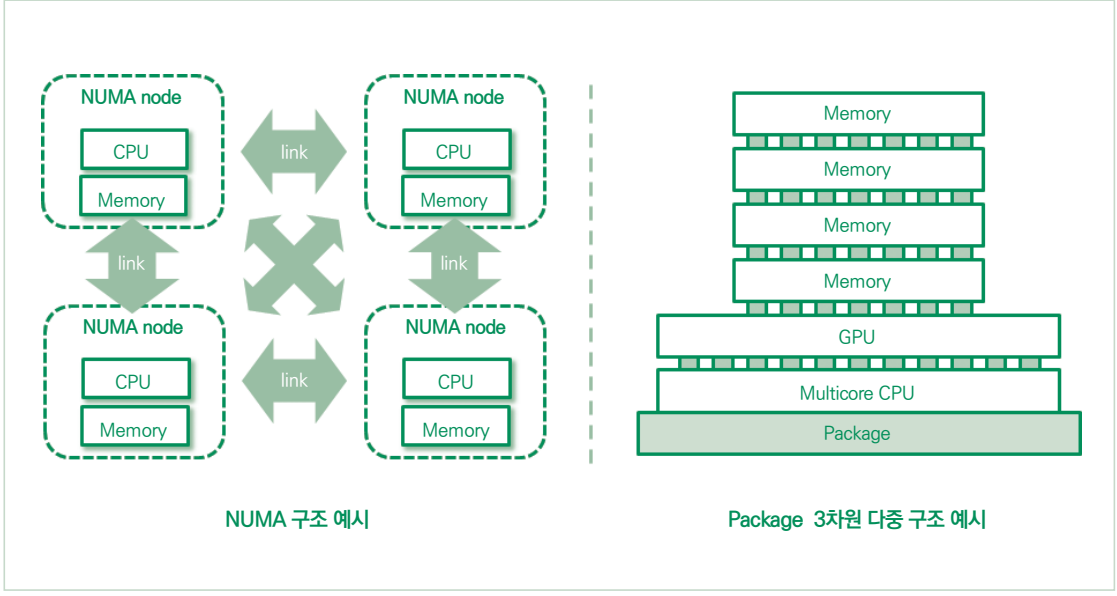
를 위해 SIMD 명령어 집합 AVX(Advanced Vector eXtensions)를 2008년부터 배포하고 있다. 이런 SIMD 명령어들은 기본적으로 분기 예측 실패를 회피하기 위해 비분기 기법으로 설계돼 있어 기존 명령어보다 성능이 뛰어나다.

데이터 수준 병렬화의 단점은 프로그램 코드가 하드웨어 특성에 종속되고 매우 복잡하다는 것이다. SIMD 레지스터의 크기, 개수와 SIMD 명령어들은 CPU의 종류에 따라 매우 다르다. 예를 들어 SIMD 레지스터의 크기가 256비트를 지원하는 CPU에서는 그에 맞는 SIMD 명령어만이 사용된다. 실례로 256비트를 지원하는 SIMD 명령어로 작성된 데이터 연산 프로그램은 64비트의 SIMD 레지스터를 지원하는 CPU에서는 수행되지 않는다. SIMD 방식은 비분기 기법을 지향하기 때문에 최소한의 반복문과 조건 분기문을 사용해야 하므로 프로그램 코드가 복잡해질 수 있다.

3. 앞으로의 병렬 구조

앞서 기술한 바와 같이, 병렬 구조는 CPU와 GPU를 기반으로 점차 발전하고 있는 추세다. 최근에는 NUMA(Non-Uniform Memory Access) 구조와 3차원 다층 구조처

[그림 7-4-4] NUMA 구조와 3차원 다층 구조 예시



럼 기존 병렬 구조를 탈피한 새로운 병렬 구조들이 설계되고 있다.

NUMA는 기존 SMP(Symmetric Multi-Processing)의 단점을 보완하기 위해 설계된 컴퓨터 구조다. SMP는 2개 이상의 CPU가 하나의 공유된 메인 메모리를 하나의 버스로 접근하기 때문에 데이터 병목 현상이 발생할 수 있었다. 이에 반해 NUMA 구조는 CPU와 지역 메모리로 구성된 NUMA 노드들로 구성된 컴퓨터 구조다.

NUMA 노드들은 물리적으로 분산돼 있고 NUMA 링크로 연결된다. 각 NUMA 노드에서 접근 가능한 다른 NUMA 노드의 지역 메모리를 원격 메모리라 한다. NUMA 구조에서 각 CPU는 자신의 지역 메모리와 하나의 버스로 연결돼 있다. 때문에 다른 CPU와의 경쟁이 SMP 구조보다 상대적으로 발생하지 않고 SMP 구조에서의 병목 현상도 발생하지 않는다.

NUMA 구조도 문제는 있다. 한 NUMA 노드의 CPU가 다른 NUMA 노드의 지역 메모리, 즉 원격 메모리에 접근하는 시간은 자신의 지역 메모리에 접근하는 시간보다 오래 걸린다. 이유는 원격 메모리를 지닌 NUMA 노드의 CPU에게 메모리 접근 권한을 할당 받아야 하기 때문이다.

3차원 다층 구조는 CPU와 GPU가 별도로 구성된 기존 병렬 구조의 단점인 병렬 연산의 성능이 CPU와 GPU 사이에 존재하는 버스의 대역폭에 종속된다는 문제를 해결하고자 설계됐다. 이 구조는 CPU, GPU, 메모리를 하나의 집적 회로에 집중시켜 다층 구조로 설계함으로써 CPU와 GPU 상의 데이터 전송에 따른 성능 저하 문제를 줄였다.

앞서 소개한 것과 같이 전자 기술의 발달에 따라서 병렬 데이터 처리를 지원하는 다양한 하드웨어 구조가 개발되고 있다. 이런 하드웨어 변화에 발맞춰 기존 데이터베이스에 적용했던 데이터 연산 기법도 병렬화해 데이터 처리 성능을 향상시키는 것이 필요할 것이다.