ETRI

전자통신동향분석 제24권 제6호 2009년 12월

병렬 프로세서 기술 및 동향

Technology and Trend of Parallel Processor

융합기술시대의 ICT 부품 연구동향 특집

정무경 (M.K. Chung) 멀티미디어프로세서설계팀 선임연구원
박성모 (S.M. Park) 멀티미디어프로세서설계팀 팀장
엄낙웅 (N.W. Eum) 시스템반도체연구부 부장

목 차

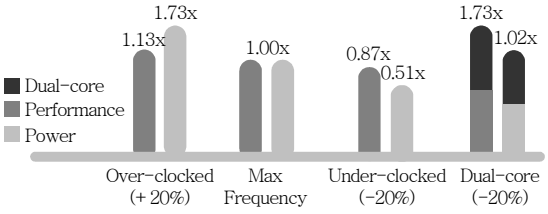
프로세서는 더 이상 동작 주파수를 높이는 방법이 아닌 다수의 프로세서를 집적하는 멀티프로세서로 기술 발전이 이루어지고 있다. 최근 2, 4, 8개의 프로세서 코어를 넘어 64, 128개 이상의 프로세서를 집적한 대규모 데이터 처리 및 과학 연산용 고성능 프로세서들이 개발되고 있다. 본 문서는 이러한 병렬 프로세싱의 개념 및 병렬 프로세서의 기술을 정리하고 최근 동향과 함께 당면한 문제점들을 기술한다.

Ⅰ. 병렬 프로세싱

1. 병렬 프로세서

1970년대 PC 등장 이후로 컴퓨터 성능의 지표는 프로세서의 동작 주파수였다. 하지만 동작 주파수가 수 GHz를 훌쩍 넘어선 지금, 무어의 법칙에 따른 프로세서 성능 발전을 단순 동작 주파수의 향상으로는 달성하기 어려운 시점에 왔다. 이를 대신할 프로세서 성능 향상 방법은 병렬 프로세서 기술이다.

프로세서 기술 발전 방향이 동작 주파수 고도화에서 병렬 프로세싱으로 전환되는 이유 중 하나는 공정 기술의 한계와 그 비용에 있다. 프로세서 동작 주파수의 향상은 그 공정 기술 발전의 영향이라 볼 수 있는데, CMOS 미세 공정 기술의 발전이 물리적인 이유로 인하여 예전과 같은 속도의 발전이 어려울 뿐 아니라, 그 공정 비용이 기하급수적으로 증가하고 있다. 또 다른 이유로 전력 소모를 들 수 있다. 프로세서의 동작 주파수를 높이기 위한 구조 및 고성능 로직 셀들은 큰 전력 소모를 가져온다. (그림 1)은 Intel 프로세서의 동작 주파수에 따른 프로세서 성능과 전력 소모를 나타낸다[1]. 프로세서의 동작 주파수를 20% 향상시켰을 경우 성능은 13% 증가한데 비하여 전력소모는 73%가 증가한다. 또한 두 개의 프로세서 코어(dual core)를 사용할 경우 같은 전력소모로 73%의 성능 향상을 기대할 수 있다. 즉 프로세서의 동작주파수를 높이기 보다 개수를 늘리는 것이 전력 대비 성능 향상에 도움을 준다. 세번째 이유는 프로세서가 수행하는 최근의 많은 응용들에 있다. 이들은 많은 양의 데이터 처리를 요구하고 있으며 컨트롤 기반의 응용에 비하여 병렬화가 쉽다.



(그림 1) 동장 주파수에 따른 프로세서 성능과 전력 소모

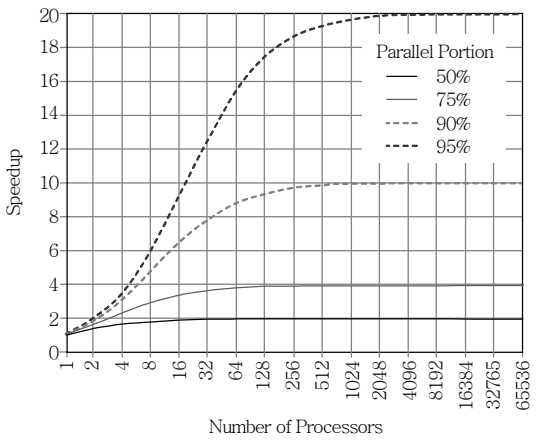
데이터 처리 및 멀티미디어, 2-D, 3-D 연산은 모두 데이터 처리의 병렬성이 매우 크며, 따라서 다수의 프로세서에서 동시에 처리하는 것이 효율적이다.

2. Amdahl의 법칙

병렬화를 통한 성능은 이상적인 경우 연산 장치의 개수에 선형적으로 향상된다. 하지만 응용에 따라 연산의 일부분은 병렬화가 불가능할 수 있다. Amdahl's law는 병렬화를 통한 성능향상을 다음식과 같이 나타냈다[2].

$$S = \frac{1}{1-P}$$

여기서 S는 기존에 비한 성능 향상 비율을 나타내고, P는 전체 수행과정 중 병렬화가 가능한 부분의 비율을 나타낸다. (그림 2)는 프로세서 개수 변화와 P에 따른 성능 향상을 보여준다. 이는 성능 향상의 최대치를 나타내는 것으로 만일 멀티프로세서에서 수행하는 프로그램의 50%가 병렬화가 불가하다면 아무리 많은 프로세서를 사용하더라도 2배의 성능 향상 이상을 얻을 수 없다. 따라서 멀티코어를 통한 성능 향상 효과의 전제는 프로그램의 병렬화 정도라 할 수 있고, 이는 병렬 프로세싱의 효율성을 결정하기 때문에 멀티프로세서 기술의 핵심이라고 할 수 있다.



(그림 2) Amdahl's Law – 프로세서 개수에 따른 성능 향상

3. 병렬 프로세싱

프로세서를 통한 프로그램 수행의 병렬화는 크게 ILP, DLP 그리고 TLP로 구분할 수 있다. ILP는 프로세서가 수행하는 명령어인 인스트럭션 다수를 동시에 수행하는 것을 의미한다. 프로그램은 컴파일 과정을 통해 순차적으로 수행되는 인스트럭션들로 변환되고, 이들 중 상호 의존관계가 없는 인스트럭션들을 찾아 동시에 수행함으로써 병렬 처리를 하게 된다. DLP는 여러 데이터를 동시에 처리하는 것을 의미한다. 많은 응용에서 다수의 데이터를 같은 연산의 반복을 통해 처리하기 때문에, 하나의 인스트럭션으로 다수의 데이터를 동시에 처리함으로써 병렬화 할 수 있다. TLP는 프로그램을 수행 흐름의 단위인 스레드들로 구성하고, 이들의 동시 수행을 통해 병렬화하는 방법이다.

프로세서의 연산 방법은 <표 1>과 같이 네 가지로 구분할 수 있다. 일반적인 single-scalar 프로세서가 하나의 인스트럭션이 하나의 데이터 연산을 하는 SISD 형태의 수행을 하는 반면, ILP를 이용하면 다수의 인스트럭션을 동시에 처리하는 MISD 혹은 MIMD 수행을 할 수 있다. 또한 DLP을 이용하면 SIMD 혹은 MIMD 형태의 다수 데이터의 동시 처리도 가능하다.

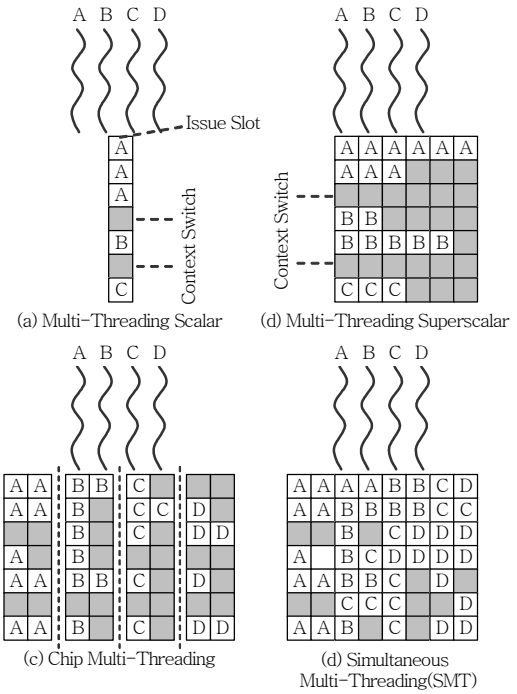
〈표 1〉 Flynn's Taxonomy

	Single Instruction	Multiple Instruction
Single Data	SISD	MISD
Multiple Data	SIMD	MIMD

4. 병렬 프로세서 구조

컨트롤 기반의 프로그램을 주로 수행하던 PC 프로세서들은 명령어의 순차적 수행을 가속하기 위해서 ILP를 높이는 방향으로 발전해 왔다. 즉 최대한 많은 인스트럭션들을 동시에 처리하는 것이 목표이다. 이때 프로그램은 순차적 수행을 전제로 하기 때문에 인스트럭션간 상호 의존관계가 있는 경우 수행 순서가 반드시 지켜져야 한다. Superscalar와 VLIW는 모두 이와 같이 동시에 다수의 인스트럭션을 수행하는 기술이다. 단 상호 의존관계 판단 및 스케줄링을 통한 수행 순서 결정을 하드웨어가 실시간으로 하는 방법이 superscalar이며, 컴파일러가 컴파일 과정에서 미리 하는 방법이 VLIW이다. 주로 PC 프로세서들은 superscalar 구조를 지니고 있으며 많은 DSP들이 VLIW 구조를 가지고 있다.

멀티스레드 프로세서는 TLP을 최대한 활용하기 위한 기술이다. 다수의 스레드를 하드웨어가 실시간으로 관리하며 스레드 스위칭시 발생하는 비용을 최소화 한다. 멀티스레드 프로세서는 그 인스트럭션 수행 방법과 스레드 스위칭 방법에 따라 (그림 3)과 같이 다양한 형태가 있을 수 있다[3]. (a)는 하나의 수행 파이프라인을 가지는 멀티스레드 프로세서이고, (b)는 다수의 수행 파이프라인을 가지는 멀티스레드 프로세서이다. (c)는 수행 파이프라인들이 분리되어 특정 스레드를 수행하는 형태인 chip multi-threading이며, (d)는 다수의 파이프라인이 임의의 스레드를 수행시킬 수 있는 SMT 구조이다. 대표적인 멀티스레드 프로세서로 SPARC[4]이 있으며, 인



(그림 3) 멀티스레드 프로세서

텔의 최근 프로세서를 포함한 PowerPC 등 대부분의 고성능 프로세서들은 SMT 혹은 chip multi-threading 형태의 멀티스레드 프로세서이다.

SIMD 인스트럭션은 DLP를 이용하여 데이터처리를 병렬화 하고자 하는 방법으로, 멀티미디어 처리 요구가 증가하면서 여러 프로세서에서 채용해 왔다. 하나의 연산에 8, 16 혹은 32bit data operand를 다수 두어 같은 연산을 여러 데이터에 대해서 수행해 주는 것으로, 기존의 범용 프로세서에 인스트럭션 추가로 비교적 쉽게 구현이 가능하고 8bit, 16bit의 멀티미디어 데이터를 한 번에 여러 개 처리할 수 있기 때문에 그 효과가 매우 크다. 인텔 IA-32의 MMX, SPARC의 VIS, PA-RISC의 MAX-2, 그리고 PowerPC의 AltiVec 인스트럭션들이 있다.

벡터프로세서 역시 SIMD 같은 병렬화 방법으로 하나의 명령어에 대하여 벡터 형태의 많은 데이터가 동시에 처리되는 구조이다. 같은 연산을 매우 많은 데이터에 대하여 처리해 주어야 하는 과학 계산 응용 등에 적합한 구조로, 수퍼컴퓨터에서 주로 활용되었지만, 최근에는 PC 및 모바일 프로세서에서도 많은 양의 데이터 처리를 요구하기 때문에 벡터구조를 사용하기 위한 연구가 진행되고 있다.

앞서 소개한 기술들은 하나의 프로세서에 다수의 연산 장치들을 두는 방법인데 반하여, 멀티 프로세서 기반 기술은 독립적인 다수의 프로세서를 사용한 병렬화 방법이다. 다수의 컴퓨터를 네트워크로 연결한 형태인 분산 컴퓨팅 방식과 다수의 프로세싱 코어를 하나의 칩에 내장한 멀티코어 컴퓨팅 방식으로 나뉠 수 있다. 분산 컴퓨팅은 그 프로세서 간 거리에 따라 단일 컴퓨터 시스템 내에 다수의 컴퓨터가 집적된 MPP와 다수의 독립된 컴퓨터를 인터넷으로 연결한 grid로 나뉜다. 과거 고성능 분산 컴퓨팅에 관한 많은 연구들이 진행되어 왔으며, 이는 반도체 공정의 발달로 하나의 칩에 다수의 프로세서가 집적되면서 멀티코어 기술로 발전하고 있다.

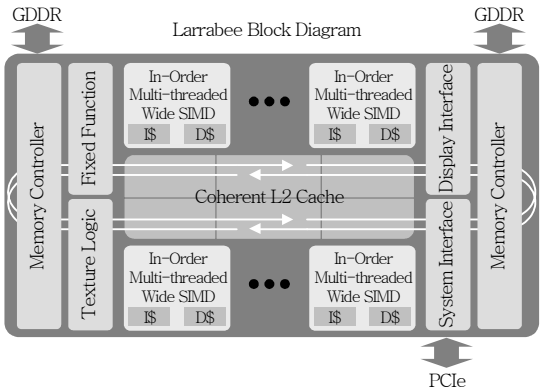
현재의 고성능 프로세서는 대부분 멀티코어 기술에 기반한다. 인텔과 AMD의 고성능 프로세서는 2, 4개 혹은 그 이상의 프로세서 코어를 집적한 멀티코어 프로세서이다. 또한 고성능을 필요로 하는 모바일 기기 역시 저전력 장점으로 인하여 멀티코어 기술이 채택되고 있다. 최근의 멀티코어는 집적되는 프로세서 개수가 크게 증가하면서 매니코어(many-core)라 불리는 기술로 발전하고 있다. 이는 하나의 칩 내에 32~256개의 코어를 집적하고 NoC를 통해 프로세서간 통신을 한다. 그래픽 프로세싱을 위한 GPU가 그 대표적인 예이다. 다음 장에서는 이러한 매니코어 기반 고성능 프로세서들을 소개한다.

Ⅱ. 매니코어

1. Intel Larrabee

Larrabee는 x86 기반의 프로세서 다수를 집적한 고성능 컴퓨팅 및 그래픽 응용을 위한 Intel의 매니코어 기반 차세대 프로세서이다. Wide SIMD vector unit을 통해 데이터 처리 성능을 높였으며, coherent cache를 이용한 메모리 구조를 가진다. (그림 4)는 Larrabee의 블록도이다[5]. 프로세서 코어로 In-Order x86 프로세서를 사용하였으며, L2 캐시를 NoC를 통해 코어들이 공유한다. Larrabee는 다음과 같은 특징을 갖는다.

- Core: original Pentium(x86-compatible)과 16-ALU-wide vector unit으로 구성
- Communication: 1024bit ring bus(NoC)
- Fixed-function 로직을 최소화하고 거의 모든



(그림 4) Larrabee 구조도

그래픽 연산 파이프라인을 소프트웨어로 처리

3-D 그래픽, 게임, 과학 연산을 주 응용으로 하고, ray tracing의 실시간 구현이 가능한 성능을 목표로 한다.

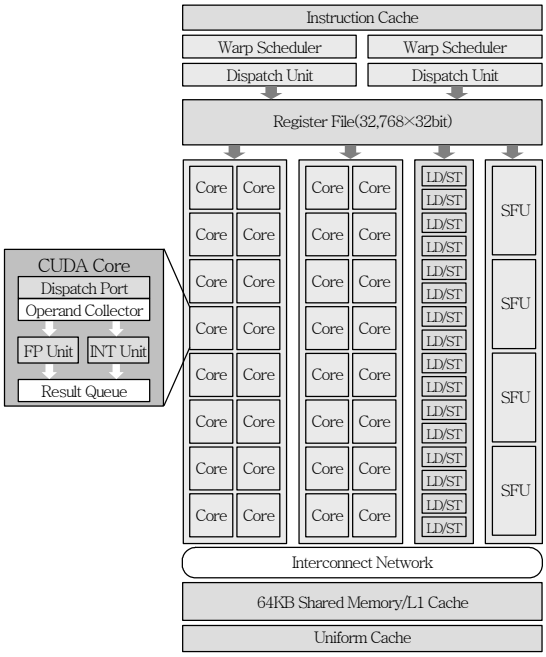
2. NVIDIA Fermi

3-D 그래픽 처리를 위해 비디오카드에 들어가던 NVIDIA의 GPU는 매니코어의 대표적인 예이다. 128개 이상의 GPU를 하나의 칩에 집적하여 그래픽 연산뿐 아니라 과학 계산용으로 활용되고 있으며 과거 슈퍼컴퓨터를 대체하고 있다. Fermi는 G80과 GT200에 이은 NVIDIA의 GPU로써 다음 기능이 향상되었다[6].

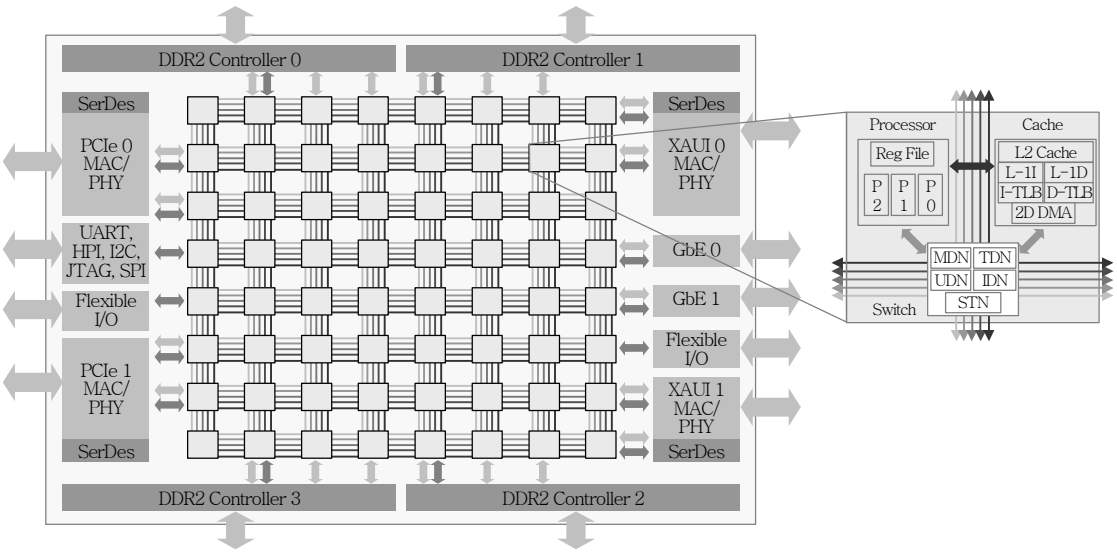
- Improve double precision performance
- ECC support
- True cache hierarchy
- More shared memory
- Faster context switching
- Faster atomic operations

(그림 5)는 Fermi를 구성하는 streaming multi-processor로써 다음 특징을 가진다.

- 32 CUDA cores per SM, 4x over GT200
- 8x the peak double precision floating point performance over GT200
- Dual warp scheduler
- 64KB of RAM with a shared memory and L1 cache



(그림 5) Fermi – Streaming Processor



(그림 6) TILE64 구조

3. Tilera TILE64

TILE64는 Tilera사의 iMesh™ NoC와 분산 coherent L1/L2 cache를 이용하여 다수의 RISC 프로세서를 집적한 매니코어 시스템이다[7]. 분산 메모리와 NoC 기술에 근간한 코어간 통신이 장점으로 네트워킹과 멀티미디어 처리를 응용으로 한다. (그림 6)은 TILE64의 구조를 보여준다.

Ⅲ. 병렬 프로그래밍

1. 프로그래밍 패러다임

병렬 컴퓨팅의 가장 중요한 이슈는 병렬 프로그래밍이다. 멀티프로세서 하드웨어 구조에 관한 연구는 이미 많이 진행되어 있으며 다수의 상용 제품들이 이미 있다. 하지만 이들 병렬 컴퓨터의 효과적인 활용을 위한 소프트웨어 개발은 아직 매우 어려운 난제이다. 과거 프로그램이라 함은 하나의 프로세서가 수행하는 일련의 순차적 명령어였지만, 병렬 프로그래밍에서는 다수의 프로세서의 동시 동작을 프로그래머가 염두해야 한다. 또한 동시에 동작하는 프로세서의 개수와 그 수행 순서가 정해지지 않는다. 다수의 프로세서에서 동시에 다수의 스레드 혹은 태스크가 진행됨으로 인해 발생하는 기능상의 변화 외에도 프로세서 간 로드 균형화, 동기화 및 통신의 최적화 등을 고려한 프로그래밍이 필요하다.

2. 병렬 프로그래밍

효과적인 병렬 프로그래밍을 위한 요소로 병렬 프로그래밍 모델, 병렬 프로그래밍 언어, 병렬 컴파일러가 있다. 병렬 컴퓨터는 프로그램이 수행될 멀티 프로세서의 구조와 동기 및 통신 방법에 따라서 지원되는 프로그래밍 모델이 정해진다. 주로 사용되는 프로그래밍 모델로, 멀티프로세스 혹은 멀티스레드, 공유메모리 통신, 메시지 전달 등이 있다.

병렬 프로그래밍 언어 혹은 병렬 프로그래밍 플랫폼은 효과적인 병렬 프로그래밍을 위한 언어/API로 다양한 프로그래밍 모델을 지원할 수 있고, 쉽게 순차적 프로그램을 병렬화 할 수 있도록 한다. 다양한 멀티 프로세서 플랫폼에 모두 적용이 가능한 표준 인터페이스 형태를 지녀야 활용 범위가 높다. OpenMP, Khronos의 OpenCL, NVIDIA의 CUDA 등이 있다.

병렬 컴파일러는 멀티 프로세서 플랫폼의 하드웨어 및 운영체제에 따라 정해지는 컴파일러로, 앞서의 병렬 프로그래밍 언어 및 API를 지원해야 하며 시스템에 최대한 최적화된 결과를 내야 한다. 병렬 컴파일러의 성능에 따라 프로그래머가 고려해야 할 사항과 최종 성능에 차이가 크다.

3. 병렬 소프트웨어 개발 환경

프로그래밍을 위한 필수 툴로 소프트웨어 개발 환경을 빼놓을 수 없다. 소프트웨어 개발 환경은 소프트웨어 분석 및 디버깅을 위한 갖가지 기능들을 가진다. 병렬 프로그래밍을 위해서는 이 외에 프로세서간 동기화 및 로드 균형화를 위한 분석 기능과 멀티 프로세서 시뮬레이션 기능이 추가되어야 한다. 다수의 프로세서 및 메모리를 비롯한 동기화, 통신 등이 정확히 시뮬레이션 되어야 하며, 시뮬레이션 중의 리소스 활용 등의 정보를 축출할 수 있어야 한다. 또한 멀티프로세서 시스템 및 운영 체제 등의 동작상의 불확실성에 의한 여러 상황을 미리 재현하기 위한 방법들이 제공되어야 한다.

4. 소프트웨어 재사용

프로세서가 멀티프로세서 시스템으로 발전하면서 기존의 단일 프로세서 시스템에서 동작하던 프로그램이 호환성 있게 동작하여야 한다. 단일프로세서를 위해 컴파일된 결과를 멀티프로세서 시스템에서 그대로 사용하여 병렬화를 하기는 사실상 어렵다. 따라서 기존의 많은 소프트웨어를 재사용하기 위해서는 legacy 소프트웨어를 병렬 소프트웨어로 변환하여 재설계하거나, 병렬 컴파일러가 자동으로 그

일을 해주어야 한다. 따라서 legacy 소프트웨어의 자동 병렬화가 주요 이슈가 되고 있다.

Ⅳ. 메모리, 통신

1. 메모리 계층 구조

멀티프로세서에서의 프로세서간 데이터 공유 및 통신 메커니즘은 메모리 구조에 의해 결정된다. 캐시 메모리는 주로 계층구조를 가지는데 프로세서간 공유하는 캐시를 L2 혹은 L3 캐시로 할 수 있다. 메모리 구조는 다수의 프로세서가 하나의 메모리를 중심으로 연결된 centralized 메모리와 다수 메모리를 분산시킨 모양인 분산 메모리 구조가 있을 수 있다. 다양한 구조가 있을 수 있으며, 응용에 따른 프로세서 간 데이터 공유 정도와 통신 방법에 장단점을 가진다.

2. 통신 네트워크

프로세서 및 시스템 내의 IP 간 통신 구조가 간단하거나 그 개수가 적은 경우 버스를 이용한 통신이 주로 사용되었다. 동시에 두 개 이상의 트랜잭션이 발생할 수 없는 버스의 단점으로 인해 프로세서 코어의 개수가 늘어난 멀티코어/매니코어의 경우 단일 버스 대신 NoC를 주로 사용한다. 다양한 토폴로지의 NoC가 있으며 프로토콜의 성격에 따라 다양한 특성이 있을 수 있다. 응용 프로그램의 데이터 통신 특성에 따라 유연성 있게 NoC를 구성하는 것이 중요하다.

3. 외부 메모리 대역폭

프로세서 성능 향상에 비하여 메모리 대역폭 발전 속도가 느리기 때문에 프로세서 시스템에서는 외부 메모리 대역폭 부족에 의한 문제가 계속해서 있어 왔다. 멀티프로세서 시스템에서는 하나의 칩 안에 다수의 프로세서 및 IP가 집적되면서 더욱 많은 하드웨어 구성요소들이 각각의 용도로 외부 메모리를 사용하게 된다. 따라서 외부 메모리의 대역폭의 부족 현상이 더욱 심각해진다. 실제로 멀티프로세서의 성능 저하의 가장 큰 원인은 내부 문제가 아닌 메모리 접근에 있는 경우가 많다. 최근 SoC는 이 문제를 해결하기 위하여 다수의 외부 메모리를 사용하는 경우가 늘고 있다. 최근 실리콘 die의 stacking 및 TSV 기술이 현실화 되면서 멀티프로세서 크기 문제와 외부메모리 대역폭 문제가 일부 해결될 수 있을 것으로 보인다.

● 용 어 해 설 ●

무어의 법칙(Moore's law): 인터넷 경제의 3원칙 가운데 하나로, 고든 무어(Gordon Moore)가 마이크로칩의 용량이 18개월마다 2배가 될 것으로 예측하여 만든 법칙이다. 또한 컴퓨터의 성능은 5년마다 10배, 10년마다 100배씩 개선된다는 내용도 포함된다.

약어 정리

API	Application Programming Interface
DLP	Data-Level Parallelism
DSP	Digital Signal Processor
ILP	Instruction-Level Parallelism
IP	Intellectual Property
MIMD	Multiple Instruction Multiple Data
MISD	Multiple Instruction Single Data
MPP	Massive Parallel Processing
NoC	Network-on-Chip
PC	Personal Computer
RISC	Reduced Instruction Set Computer
SIMD	Single Instruction Multiple Data
SISD	Single Instruction Single Data
SM	Streaming Multiprocessor
SMT	Simultaneous Multi-Threading
SoC	System-on-Chip
TLP	Thread-Level Parallelism
TSV	Through-Silicon Via
VLIW	Very Long Instruction Word

참 고 문 헌

[1] "Intel® Multi-core Processor – Making the

Move to Quad-Core and Beyond," Intel, 2008.
[2] Gene M. Amdahl, "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities," *AFIPS Conf. Proc.*, Vol.30, 1967, pp.483-485.
[3] Theo Ungerer et al., "A Survey of Processors with Explicit Multithreading," *TOCS 2003*, 2003.
[4] http://www.sun.com/products/microelectronics/index.jsp SPARC, Sunmicrosystems.
[5] http://www.intel.com/technology/visual/microarch.htm, Larrabee, Intel.
[6] http://www.nvidia.com/object/fermi_architecture.html, Fermi, NVIDIA
[7] http://www.tilera.com, TILE64, Tilera.