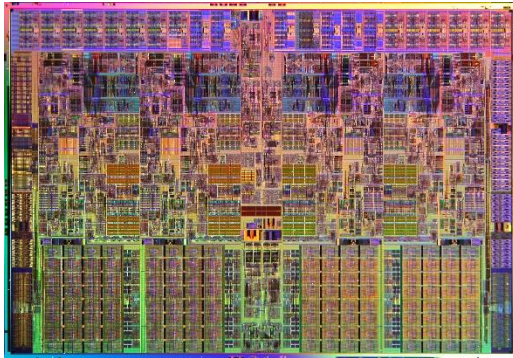


Lecture. 2

병렬 컴퓨팅 개요

Introduction to parallel computing

병렬 아키텍처 (Parallel architecture) 란?



Intel Quad core i7



Sony playstation



IBM Blue Gene



Nvidia Kepler architecture



Google data center locations

Parallel computing
vs Distributed
computing

병렬처리 하드웨어 (Parallel HW)

- Flynn's Taxonomy

Single core processor

Vector processor

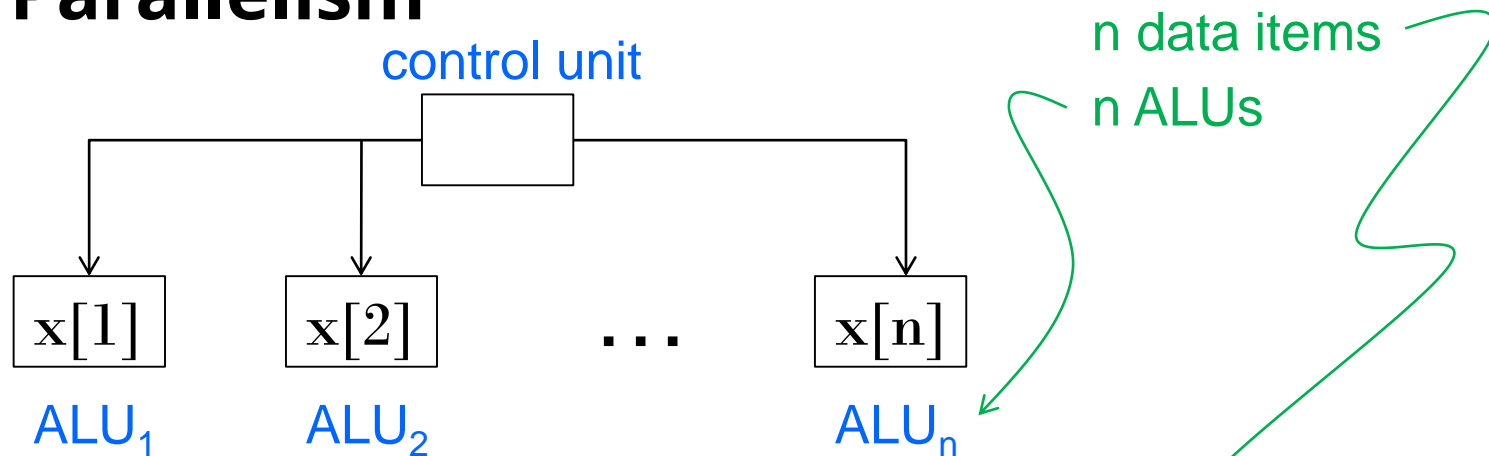
SISD Single instruction stream Single data stream	SIMD Single instruction stream Multiple data stream
MISD Multiple instruction stream Single data stream	MIMD Multiple instruction stream Multiple data stream

Not covered

Multi-core processor

SIMD

- **Single Instruction, Multiple Data**
 - 하나의 명령어를 여러개의 데이터에 적용
- **Data Parallelism**



```
for (i = 0; i < n; i++)  
    x[i] += y[i];
```

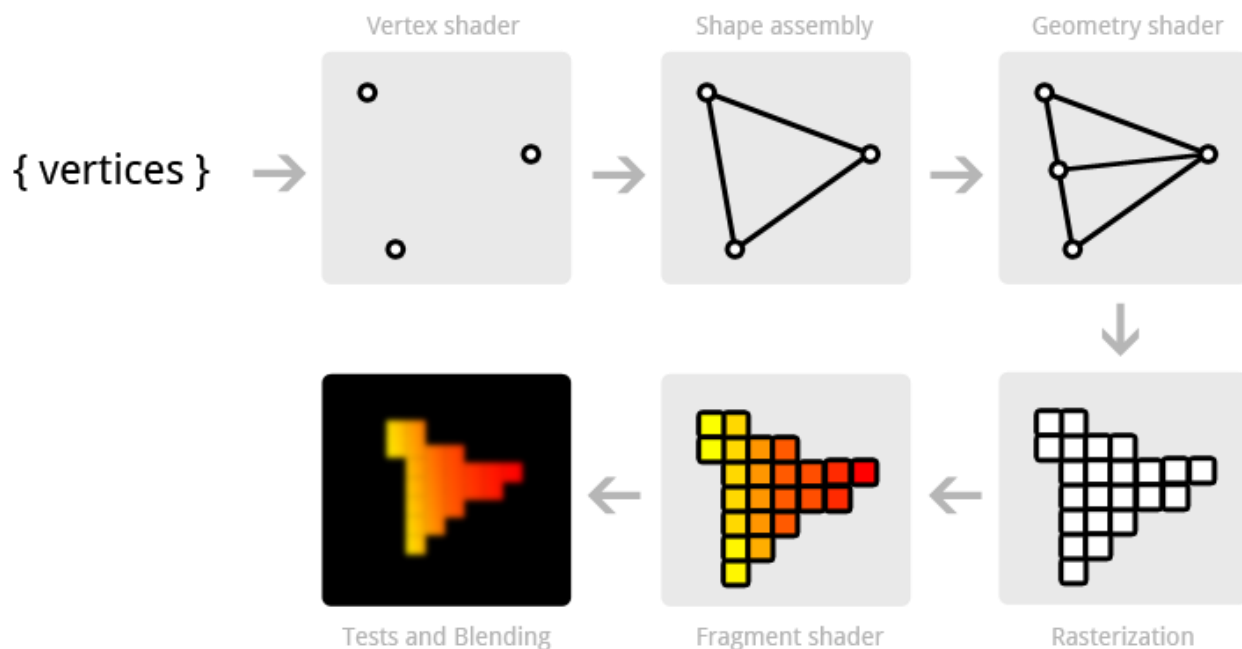
SIMD

- **Single Instruction, Multiple Data**
 - 하나의 명령어를 여러 개의 데이터에 적용
- **Data Parallelism**
- 예) 4 ALUs, 15 data

Round	ALU ₁	ALU ₂	ALU ₃	ALU ₄
1	X[0]	X[1]	X[2]	X[3]
2	X[4]	X[5]	X[6]	X[7]
3	X[8]	X[9]	X[10]	X[11]
4	X[12]	X[13]	X[14]	

Vector Processors

- 데이터의 배열(array or vector)을 사용하여 동작
 - SIMD의 대표적인 예
 - 예, MXX/SSE/AVX(x86), XeonPhi, **GPU (SIMT)** 등



Vector Processors – 장점

- 빠르고, 사용이 쉬움
- 컴파일러의 도움을 받기 쉬움
 - 병렬화 가능 영역 판단 및 자동 병렬화 등



Images from <https://open.gl>

Vector Processors – 장점

- 빠르고, 사용이 쉬움
- 컴파일러의 도움을 받기 쉬움
 - 병렬화 가능 영역 판단 및 자동 병렬화 등
- 높은 메모리 대역폭(**bandwidth**)
 - 다중 बैं크(bank) 구성
 - Issues: bank conflict, memory access pattern
- 캐시 활용 효율이 높음
 - 높은 지역성(locality)

*** 알고리즘의 유형에 따라 달라질 수 있음 ***

Vector Processors – 단점

- 불규치한 데이터 구조에 적용이 어려움
 - 연산장치 활용 효율성 저하
 - Divergence

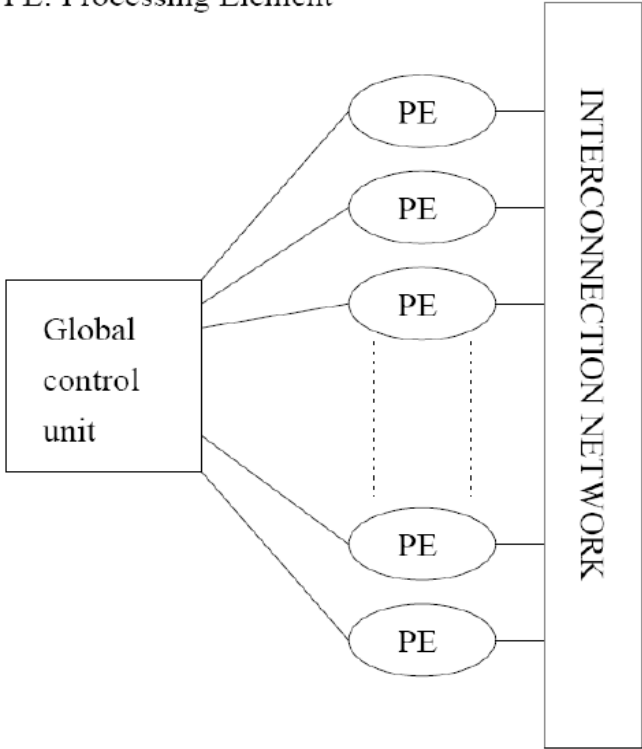
```
switch (a[i]){  
  case a:  
    // Do something  
  case b:  
    // Do something  
  ... }
```

- 개별 연산장치에 대한 제한된 자원 할당
 - 레지스터 수, 캐시 용량 등
 - 복잡한 연산에 대한 적용이 제한적
 - 최적화된 알고리즘 설계가 필요

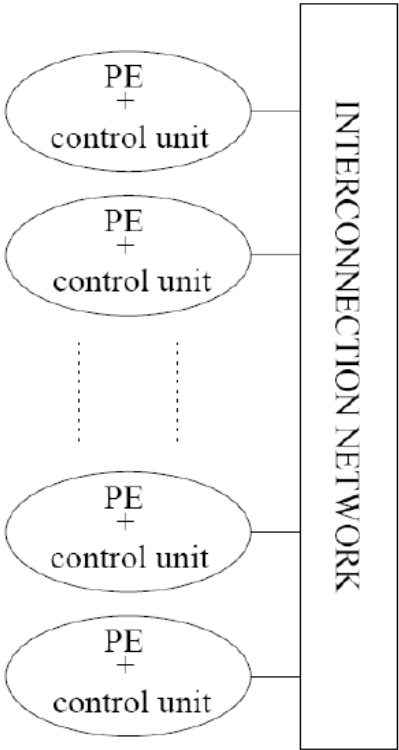
Images from <https://open.gl>

SIMD vs. MIMD

PE: Processing Element



SIMD architecture



MIMD architecture

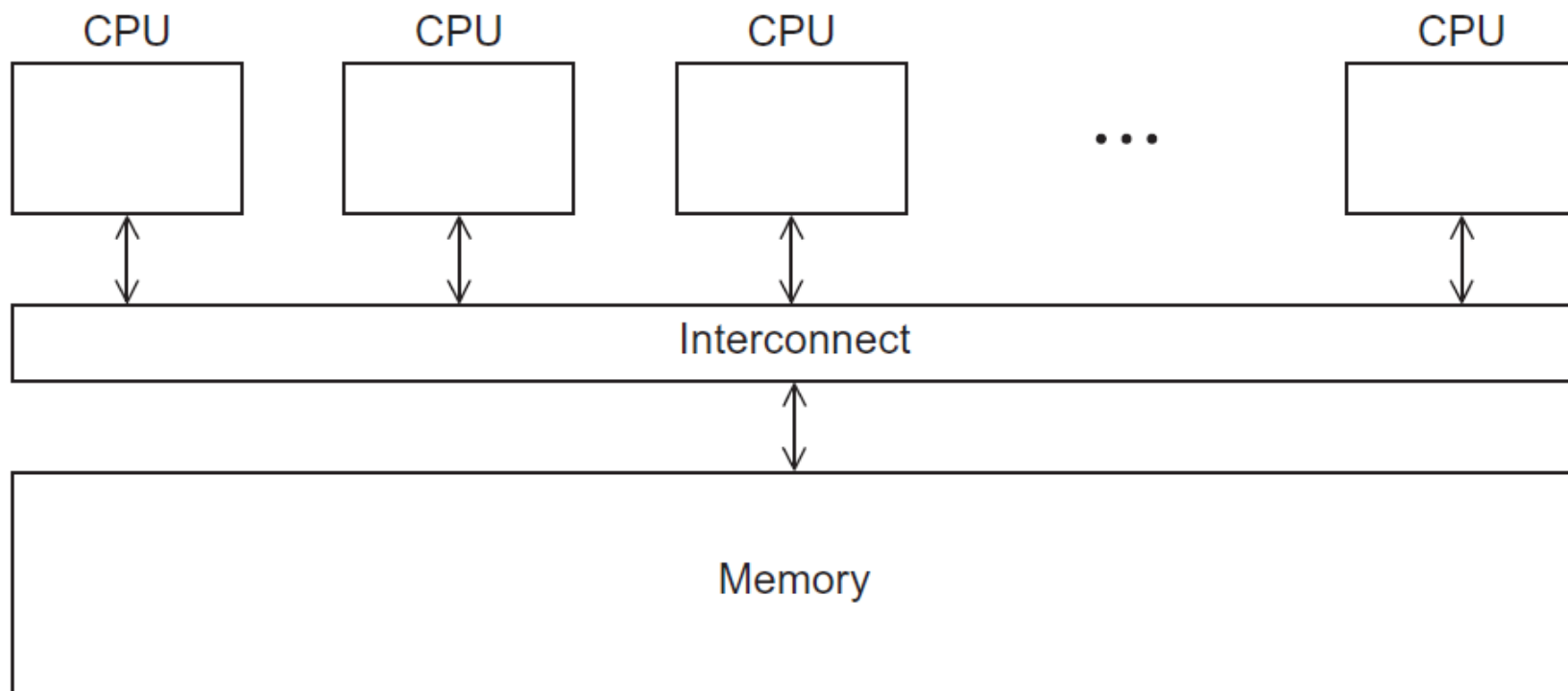
MIMD

- **M**ultiple **I**nstruction, **M**ultiple **D**ata
 - 여러개의 명령어를 각각의 데이터에 적용
- 독립된 프로세서들의 집합
 - 예) Multi-core CPUs
- **Thread-level parallelism**



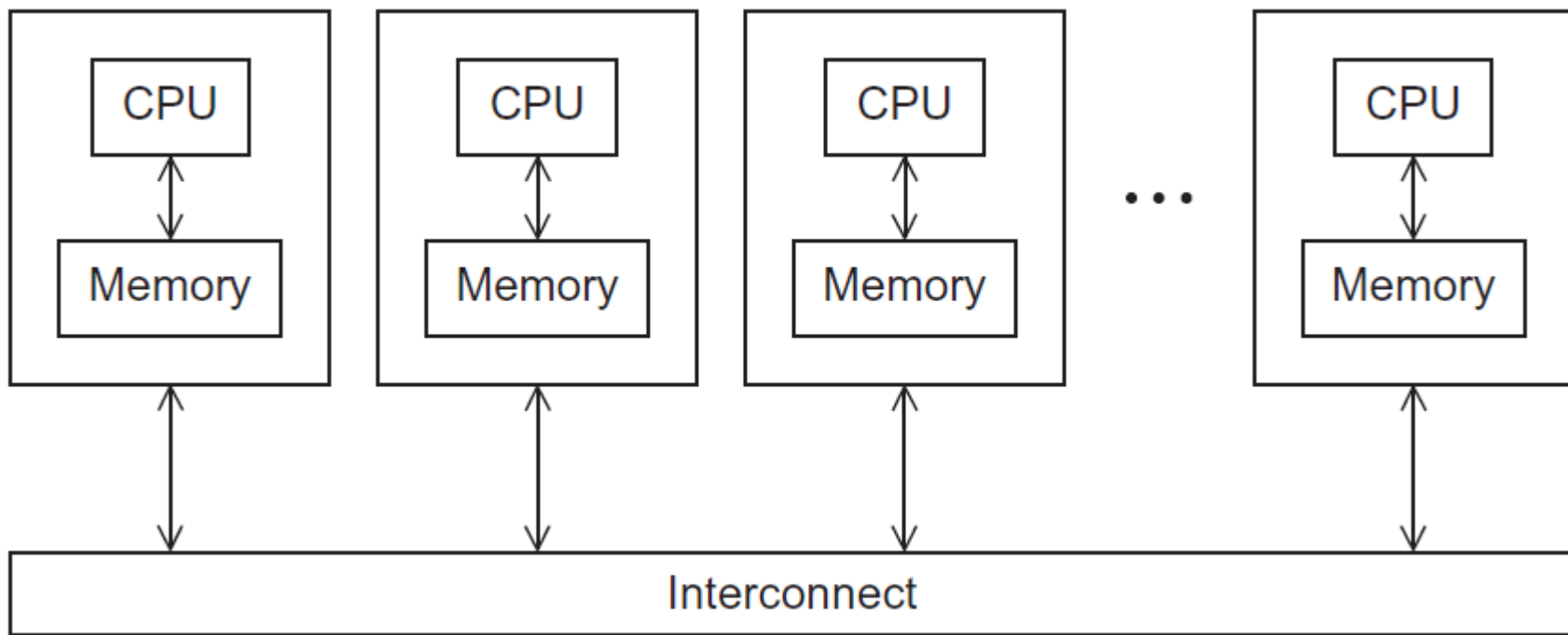
Shared Memory Systems

- 코어들이 같은 메모리 공간을 공유
- 메모리 접근 시 주의가 필요



Distributed Memory Systems

- 코어 각각이 독립된 메모리 공간을 가짐
- 데이터 공유 및 교환을 위해 명시적 통신이 필요함



Outline

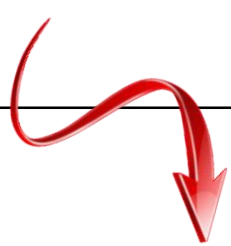
- **Parallel Computing Architecture**
 - Flynn's taxonomy
 - MIMD vs SIMD
 - Share memory vs distributed memory system
- **Nondeterminism**
- **Performance of Parallel Computing**

Nondeterminism

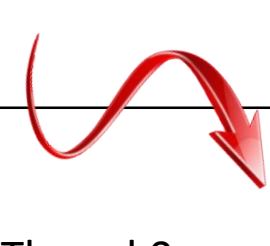
- MIMD 시스템의 프로세서는 비동기적으로 동작

Thread 0: my_x = 7, Thread 1: my_x = 19

```
...  
printf ( "Thread %d > my_val = %d\n" ,  
        my_rank , my_x ) ;  
...
```



Thread 1 > my_val = 19
Thread 0 > my_val = 7



Thread 0 > my_val = 7
Thread 1 > my_val = 19

Nondeterminism

- MIMD 시스템의 프로세서는 비동기적으로 동작

Thread 0: my_x = 7, Thread 1: my_x = 19

$x = 0$

`my_val = Compute_val (my_rank) ;`

`x += my_val ;`

$x = ?$

Time	Core 0	Core 1
0	Finish assignment to my_val	In call to Compute_val
1	Load $x = 0$ into register	Finish assignment to my_val
2	Load my_val = 7 into register	Load $x = 0$ into register
3	Add my_val = 7 to x	Load my_val = 19 into register
4	Store $x = 7$	Add my_val to x
5	Start other work	Store $x = 19$

Nondeterminism

- 동기화를 고려해야 함
 - Race condition
 - Critical section
 - Mutually exclusive
 - Mutual exclusion lock (mutex, or simply lock)

```
my_val = Compute_val ( my_rank ) ;  
Lock(&add_my_val_lock ) ;  
x += my_val ;  
Unlock(&add_my_val_lock ) ;
```

Outline

- **Parallel Computing Architecture**
 - Flynn's taxonomy
 - MIMD vs SIMD
 - Share memory vs distributed memory system
- **Nondeterminism**
- **Performance of Parallel Computing**

병렬 프로그램의 성능

- **Speedup**

$$S = \frac{T_{\text{serial}}}{T_{\text{parallel}}}$$

- **Linear speedup**

- $|\text{Processor}| = p$

$$T_{\text{parallel}} = T_{\text{serial}} / p$$

병렬 프로그램의 성능

- 실제 병렬 프로그램의 수행시간

$$T_{\text{parallel}} = T_{\text{serial}} / p + T_{\text{overhead}}$$

- 병렬 프로그램의 효율성 (Efficiency)

$$E = \frac{S}{p} = \frac{\left(\frac{T_{\text{serial}}}{T_{\text{parallel}}} \right)}{p} = \frac{T_{\text{serial}}}{p \cdot T_{\text{parallel}}}$$

병렬 프로그램의 성능

p	1	2	4	8	16
S	1.0	1.9	3.6	6.5	10.8
$E = S/p$	1.0	0.95	0.90	0.81	0.68

얼마나 빨라질 수 있을까?

- 병렬화 가능 부분: 90%
- 프로세서 수: 3
- 예상 성능 향상은?



얼마나 빨라질 수 있을까?

- 암달의 법칙 (Amdahl's Law)

- 프로세서의 수가 아무리 많아도, 모든 부분을 병렬화 하지 않으면 성능 향상은 제한적

$$S = \frac{1}{(1 - p) + \frac{p}{s}}$$

병렬화 가능한 비율

병렬화 성능
(병렬처리 영역에 대한)

확장성 (Scalability)

- 프로세서의 수 변화에 따른 병렬화 효율 변화를 기술
- 높은 확장성 (High scalability)
 - 프로세서의 수가 늘어도 병렬화 효율이 유지됨
- 병렬 알고리즘 설계 시, 확장성을 고려해야 함

Summary

- **Parallel Computing Architecture**
 - Flynn's taxonomy
 - MIMD vs SIMD
 - Share memory vs distributed memory system
- **Nondeterminism**
- **Performance of Parallel Computing**