

AI 컴퓨팅 성능 공학 입문

튜링상 수상자들로 본 AI 컴퓨팅 성능

신정훈 (KISTI)
2024. 12.

AI 컴퓨팅 성능학 입문

학습 내용

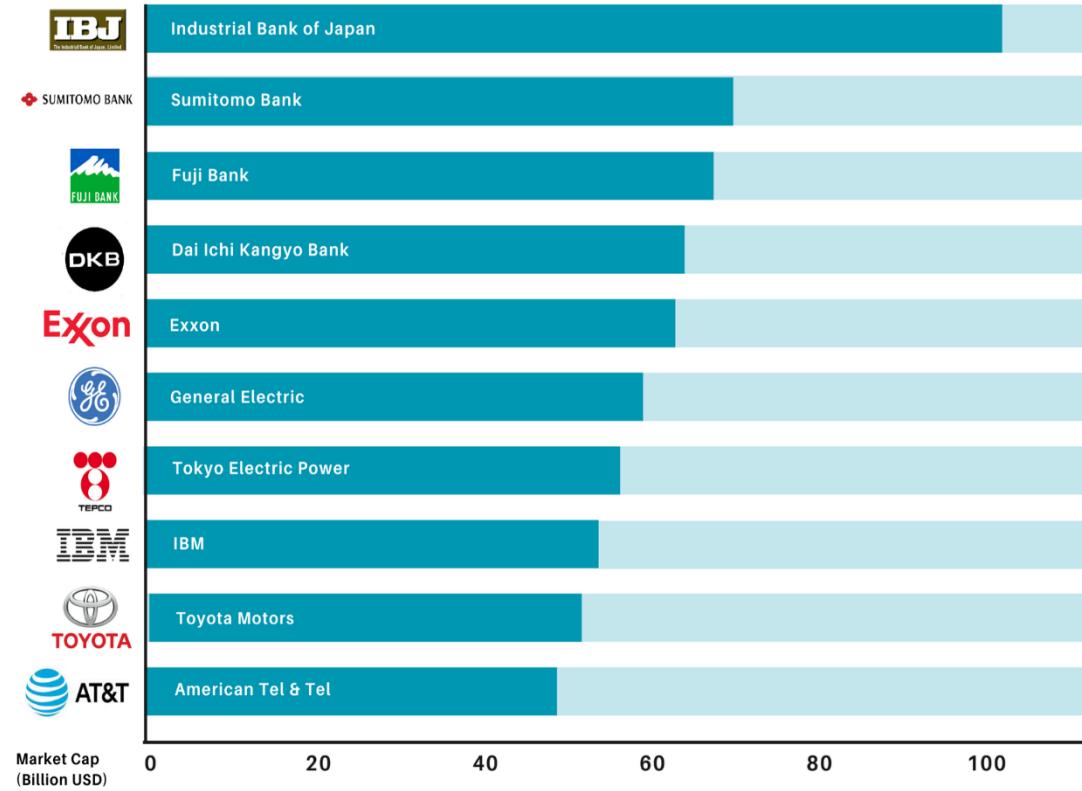
- 컴퓨팅 아키텍처: 프로세서, 메모리, 명령어 등
- Turing Award 수상자들로 본 ‘계산 성능 Computing Performance’
- 빠르고 효율적인 계산을 위한 여정: 하드웨어/소프트웨어/알고리즘
- Domain-specific: HPC 분야, 컴퓨터 그래픽스 분야, AI 분야
- 무어의 법칙, 데나드 스케일링 법칙, 그리고 암달의 법칙
- AI 기술 구성 및 발전사: 알고리즘(역전파 등)과 GPU
- AI 컴퓨팅 성능 최적화 기술들

학습 목표

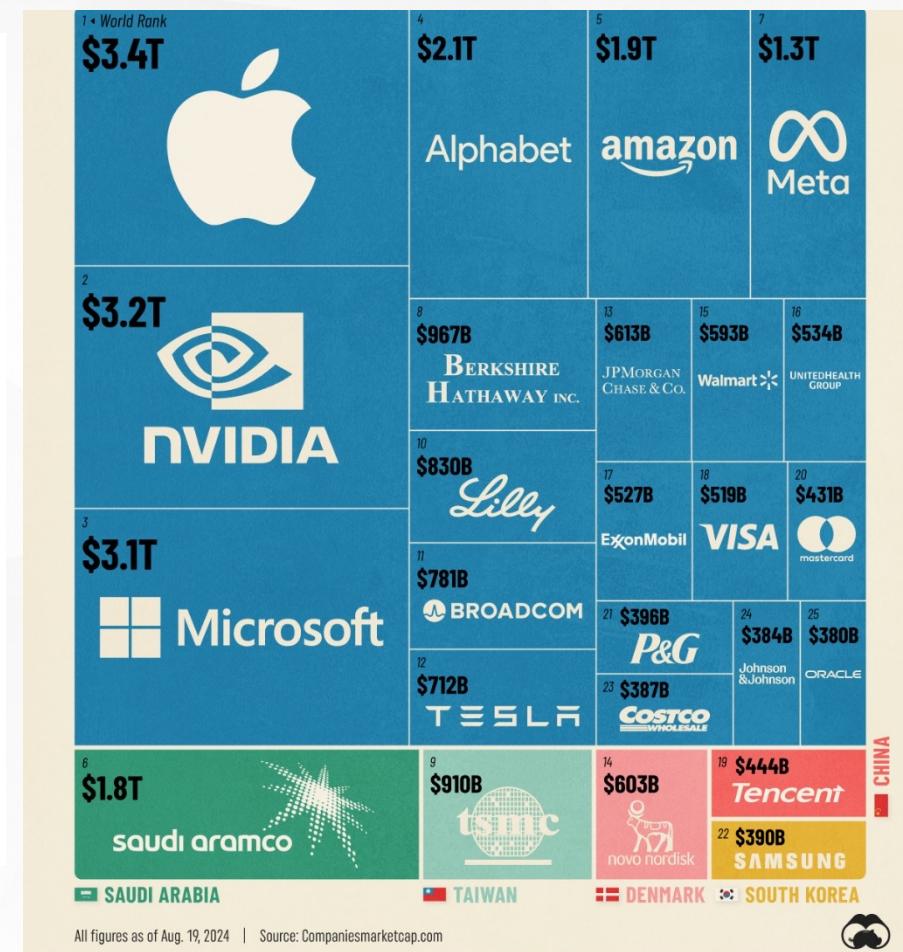
- 컴퓨팅 성능 공학 분야의 동향과 기술이슈들을 알아본다.

글로벌 기업 시가총액 – IT테크 기업의 시대로

1989년 1월

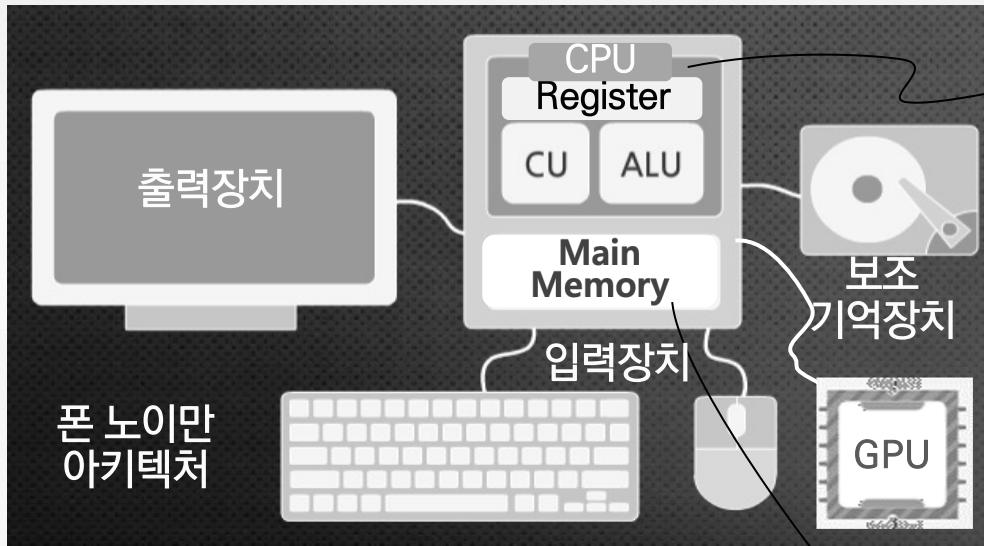


2024년 12월

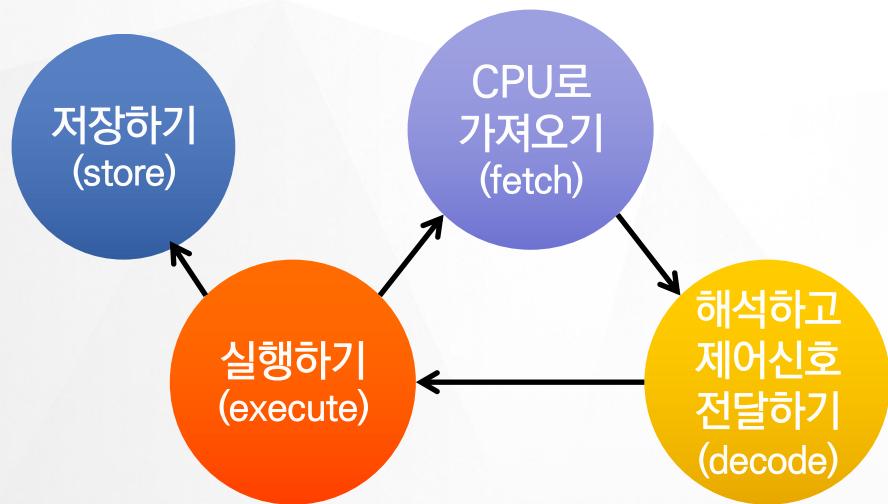


All figures as of Aug. 19, 2024 | Source: Companiesmarketcap.com

전통적 컴퓨터 구조 – CPU, 주기억장치, 보조기억장치, I/O, GPU



명령어(Instruction) 사이클



◆ **중앙처리장치(Central Process Unit)**
: 프로그램에 포함된 명령들을 하나씩 차례대로 실행

- 제어장치(Control Unit, CU): 주기억장치에서 명령을 하나씩 가져와(fetch) 해석하고(decode) 실행하여(execute) 결과를 임시보관하거나 저장함(store)
- 산술논리연산장치(Arithmetic Logic Unit, ALU): 사칙연산, 논리연산 등을 수행

◆ **명령어(Instruction):** 프로세서가 이해하는 지시

- 1) 기본적인 사칙 연산, 논리 연산
- 2) 메모리에 쓰고 읽는 명령
- 3) 프로그램의 실행 흐름을 제어하는 분기, 호출 명령

◆ **주기억장치(Main Memory)**

- : 현재 실행 중에 있는 프로그램과 이 프로그램이 필요로 하는 데이터를 일시적으로 저장하는 장치
- 프로그램과 데이터는 서로 다르나 저장방식에 있어 프로그램을 데이터와 동일하게 취급
 - 모두 주기억장치에 저장

튜링상 (ACM Turing Award)



- ◆ ACM에서 컴퓨터과학 분야에 업적을 남긴 사람에게 매년 시상하는 상
- ◆ 현대 컴퓨터과학의 아버지라 할 수 있는 앤런 튜링의 이름을 땠음
- ◆ "컴퓨터과학의 노벨상"이라고 불림
- ◆ Google이 매년 총 100만 달러의 상금을 후원

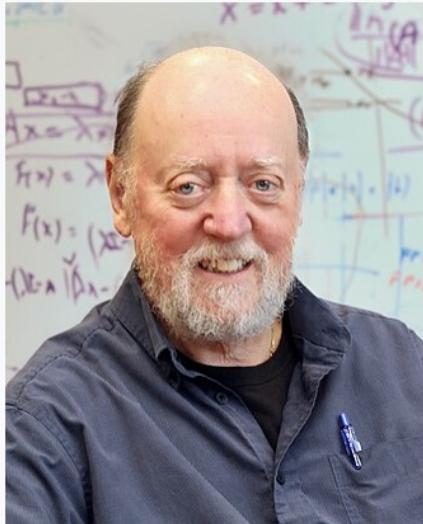


최근 수상자		
아비 위그더슨	High Performance Computing	로버트 메칼프
2023년		2022년
잭 동가라	✓	제프리 울만
2021년	High Performance Computing	2020년
앨프리드 에이호	✓	에드워 캐트멀
2020년	Graphics	2019년
팻 핸러핸	✓	제프리 힌턴
2019년	Graphics	2018년
요슈아 벤지오	✓	얀 르쿤
2018년	AI	2018년
데이비드 A 패터슨	✓	존 헤네시
2017년	Processor	2017년

튜링상 수상자들

Jack Dongarra

FRS



Dongarra in 2022

Born

July 18, 1950 (age 74)
Chicago, Illinois, U.S.

Alma mater Chicago State University (BS)
Illinois Institute of Technology (MS)
University of New Mexico (PhD)

Known for EISPACK, LINPACK, BLAS,
LAPACK, ScaLAPACK,^{[2][3]} Netlib,
PVM, MPI,^[4] NetSolve,^[5]
Top500, ATLAS,^[6] and PAPI^[7]

에드溫 캐트멀(Edwin Catmull, 1945년 3월 31일 ~)은 미국의 전산학자이자 픽사의 사장이다.

유타 대학교에 입학하여 물리학과 전산학을 전공하면서 컴퓨터 그래픽스의 중요한 기본 개념인 Z 버퍼링, 텍스처 맵핑, B 스플라인을 만들었다. 또한 3차원 컴퓨터 그래픽스를 이용한 최초의 영화를 제작하였다.

1979년 루카스필름에 입사하여 이미지 합성 기법을 개발하였다. 1986년에는 픽사를 창업하였다. 픽사에서 제작한 3차원 렌더링 소프트웨어 렌더맨의 주요 개발자이기도 하다.

렌더맨과 이미지 합성 기법의 개발로 아카데미 공로상을 세 번 수상하였다. 컴퓨터 그래픽스 분야에 많은 업적을 남겼으며,^{[1][2]} 이 업적으로 2019년 튜링상을 수상하였다.

저서로 **창의성을 지휘하라**가 있다. 루카스필름에서 시작하여 픽사를 창업하고 월트 디즈니 컴퍼니에 합병된 후까지의 일대기를 다룬 일종의 자서전이다.

에드溫 캐트멀

Edwin Catmull



에드溫 캐트멀 (2010년)

출생

1945년 3월 31일(79세)
미국 웨스트버지니아주 파커즈버그

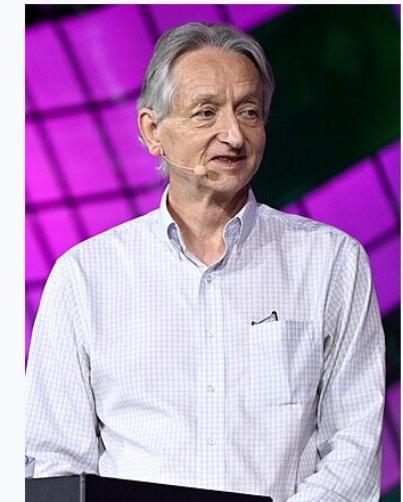
국적

미국

출신 학교 유타 대학교 (Ph.D. Computer Science; B.S. Physics and Computer Science)

Geoffrey Hinton

CC FRS FRSC



Hinton speaking at the University of Toronto in 2023

Born

Geoffrey Everest Hinton
6 December 1947
(age 76)^[11]

Wimbledon, London, England

Education

University of Cambridge (BA)
University of Edinburgh (PhD)

Known for

Applications of backpropagation
Boltzmann machine
Deep learning
Capsule neural networks

존 L. 헤네시

문서 토론

읽기 편집 역사 보기 도구 ▾

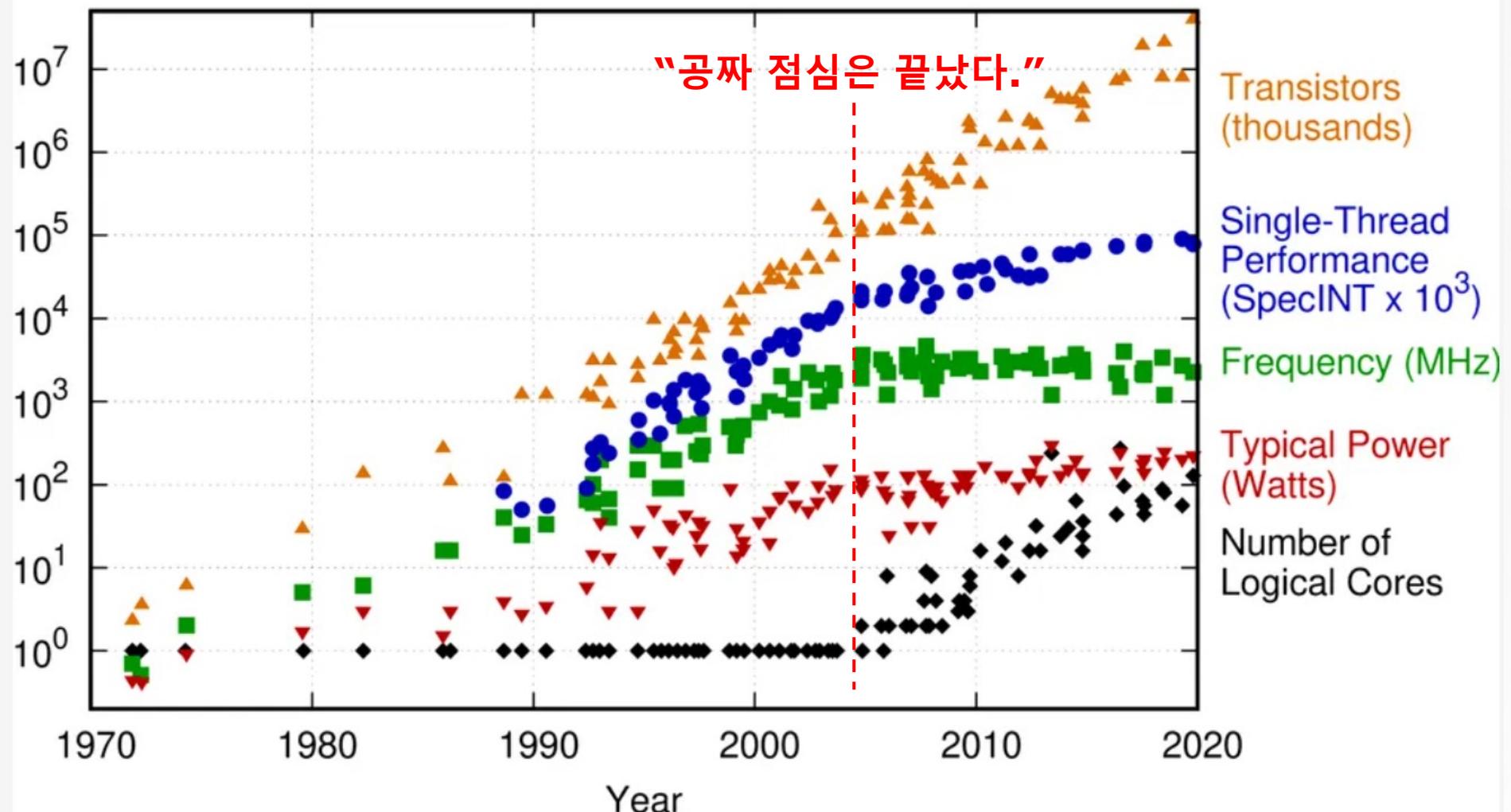
위키백과, 우리 모두의 백과사전.

존 L. 헤네시(John L. Hennessy, 본명: John Leroy Hennessy, 1952년 9월 22일 ~)는 알파벳 의장을 역임한 미국의 컴퓨터 과학자, 학술자, 사업가이다.^[1] 헤네시는 MIPS 컴퓨터 시스템스, 아테로스의 설립자 중 한 명이며 스탠퍼드 대학교의 10대 학장을 역임하였다. 헤네시는 2016년 여름에 사직을 발표하였고, 이후 마크 테시에러빈으로부터 학장을 인계 받았다.^[2] 마크 앤드리슨은 그를 실리콘밸리의 대부로 불렀다.^[3]



하드웨어 아키텍처 성능 장벽 since 2004

48 Years of Microprocessor Trend Data



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2019 by K. Rupp

The Take Away

- HPC Hardware is Constantly Changing
 - Scalar
 - Vector
 - Distributed
 - Accelerated
 - Mixed precision
- Three computer revolutions
 - High performance computing
 - Deep learning
 - Edge & AI
- Algorithm / Software advances follows hardware
 - And there is “plenty of room at the top”

견인 분야
(Domain)

1. 하드웨어
2. 소프트웨어
3. 알고리즘

특정 domain 기반으로...

“There's plenty of room at the Top: What will drive computer performance after Moore's law?”

Leiserson et al., *Science* 368, 1079 (2020) 5 June 2020

The Top

Technology	Algorithms	Hardware architecture
01010011 01100011 01101001 01100101 01101110 01100011 01100101 00000000		
Software	Algorithms	Hardware architecture
Software performance engineering	New algorithms	Hardware streamlining
Removing software bloat	New problem domains	Processor simplification
Tailoring software to hardware features	New machine models	Domain specialization

The Bottom

for example, semiconductor technology

Leiserson et al., *Science* 368, 1079 (2020) 5 June 2020

2021 AM Turing Award Recipient Jack Dongarra Turing Lecture: "A Not So Simple Matter of Software"



Association for Computing Machinery (ACM)
구독자 4.3만명

구독중 ▾

225

👎

공유

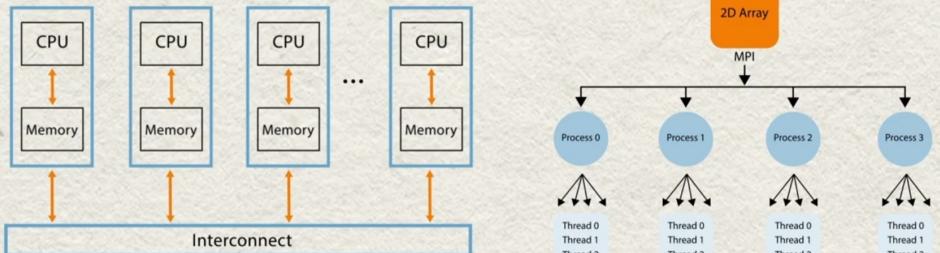
오프라인 저장

▶



High Performance Computer (HPC): 전통적 계산 집약 Domain

Parallel Computing 2000s Shared & Distributed Memory Systems



Again, a number of commercial companies created to develop systems:
BBN, Elxsi, Myrias, Tera, Thinking Machines, Intel Sci Computer, Meiko,
Maspar, nCube, AMT/DAP ...

현재(2024) 대부분의 Supercomputer에서
Heterogeneous (CPU+GPU) 형태를 채용

TOP500 LIST - JUNE 2024

R_{max} and R_{peak} values are in PFlop/s. For more details about other fields, check the TOP500 description.

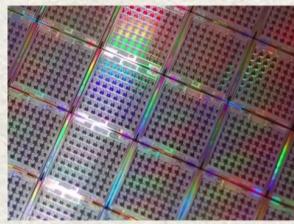
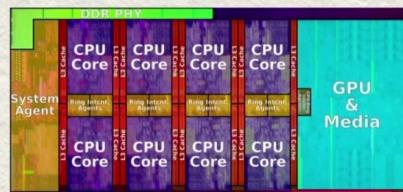
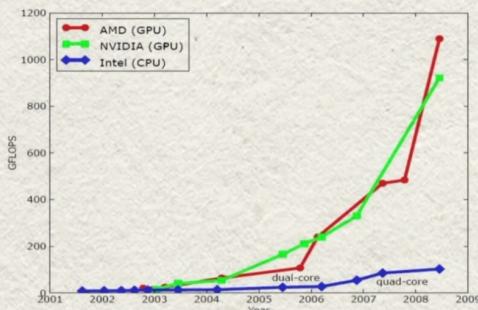
R_{peak} values are calculated using the advertised clock rate of the CPU. For the efficiency of the systems you should take into account the Turbo CPU clock rate where it applies.



LINPACK 벤치마킹 성능시험
(주로 행렬계산 관련)

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	Frontier - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,699,904	1,206.00	1,714.81	22,786
2	Aurora - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	9,264,128	1,012.00	1,980.01	38,698
3	Eagle - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Azure Microsoft Azure United States	2,073,600	561.20	846.84	
4	Supercomputer Fugaku - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
5	LUMI - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC	2,752,704	379.70	531.51	7,107

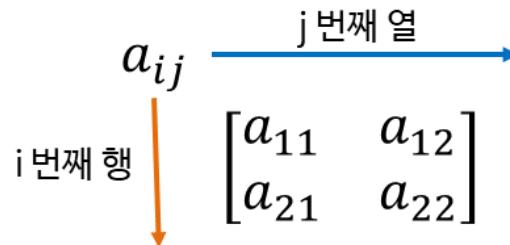
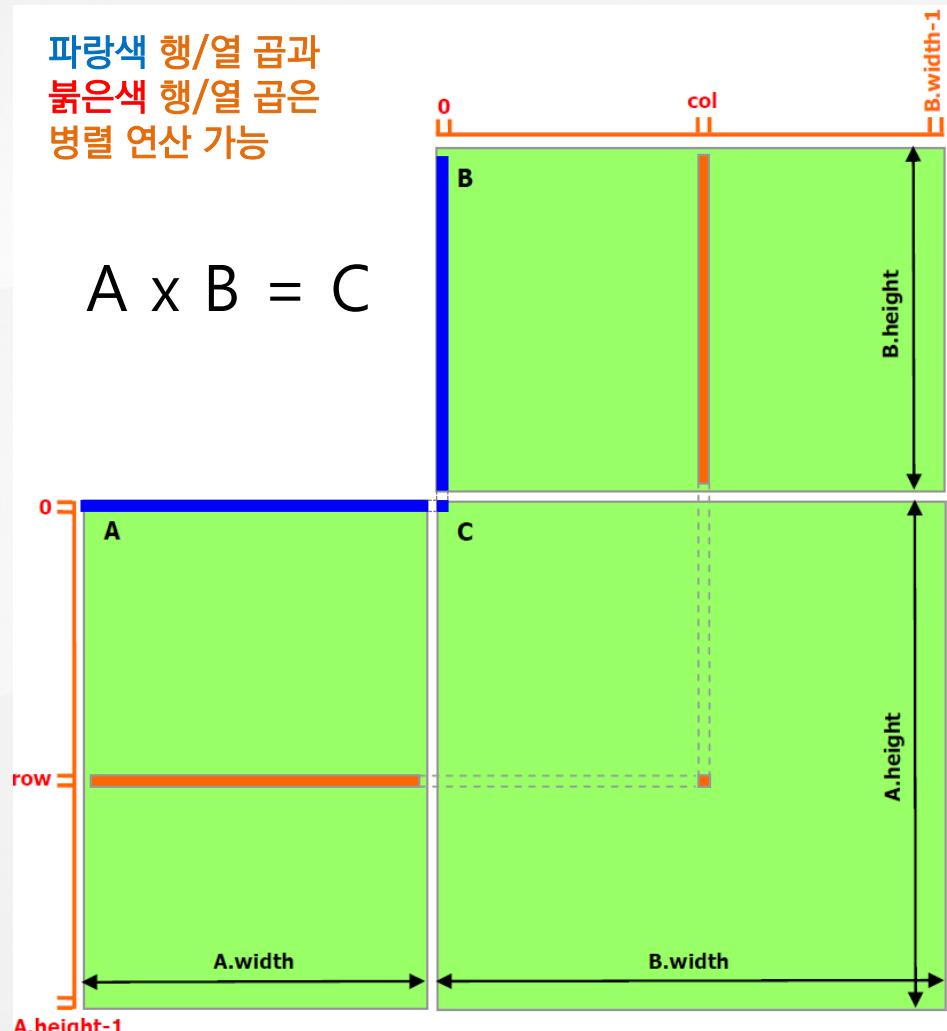
Multicore/Hybrid Architectures 2010s



HPC 핵심 알고리즘: 행렬 연산

파랑색 행/열 곱과
붉은색 행/열 곱은
병렬 연산 가능

$$A \times B = C$$

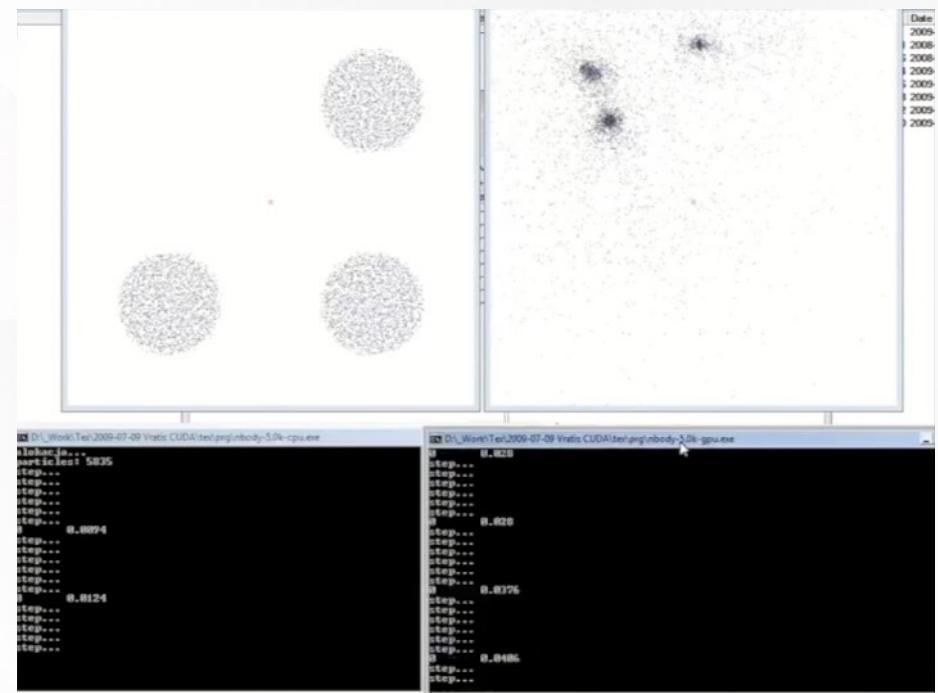


ex) BLAS 알고리즘 Set

◆ 예시: 입자 운동 물리 계산 가속

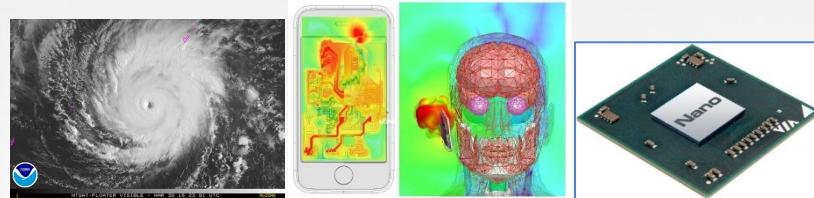
CPU 사용

CPU+GPU 사용



출처 - CUDA C++ Programming Guide, <https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html>

과학 계산 Domain – 전통적 계산 집약 분야



[입자] N-body simulation (naive)

$$\vec{a} = \frac{1}{m} \sum \vec{F}$$

[고체] Hooke's law Equations

$$\sigma = \lambda(\nabla \cdot E)I + 2\mu E$$

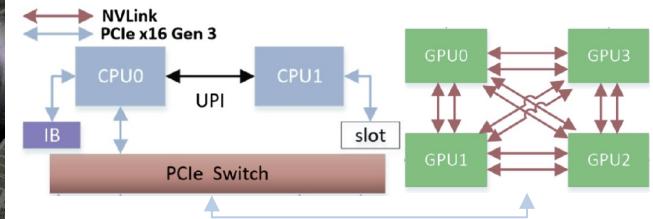
[열유체] Navier–Stokes Equation

$$\frac{\partial}{\partial t}(\rho \mathbf{u}) + \nabla \cdot (\rho \mathbf{u} \mathbf{u} + p \mathbf{I}) = \nabla \cdot \boldsymbol{\tau} + \rho$$

[양자] The Schrödinger Equation

$$i\hbar \frac{\partial}{\partial t} \Psi = \left(-\frac{\hbar^2}{2m} \nabla^2 + V \right) \Psi$$

High Performance Computer (슈퍼컴퓨터, 병렬연산 등)

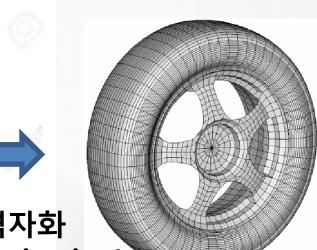


예) 타이어 변형문제: 70 km/h, 1회전, 16 cores일 때, 계산시간은 40분

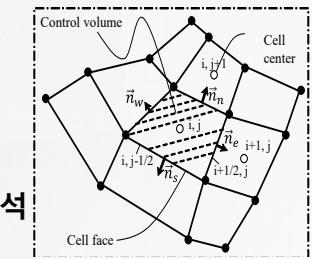
Modeling & Simulation (수식화 → 디지털화 → 수치해석)



CAD model



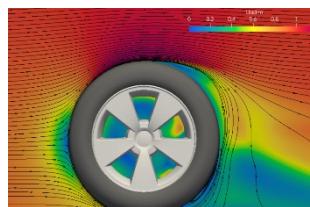
Mesh model



수치해석

수치해석(FDM, FEM, ...)

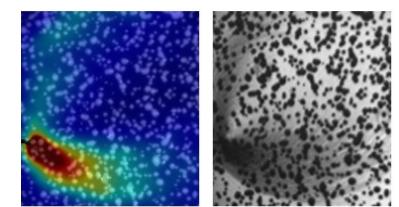
시뮬레이션 결과의 가시화 및 분석



공기저항 + 발열

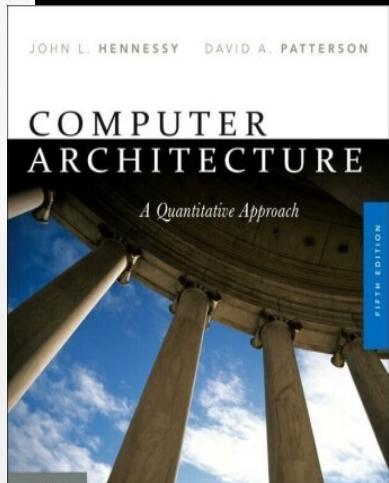


변형 분포



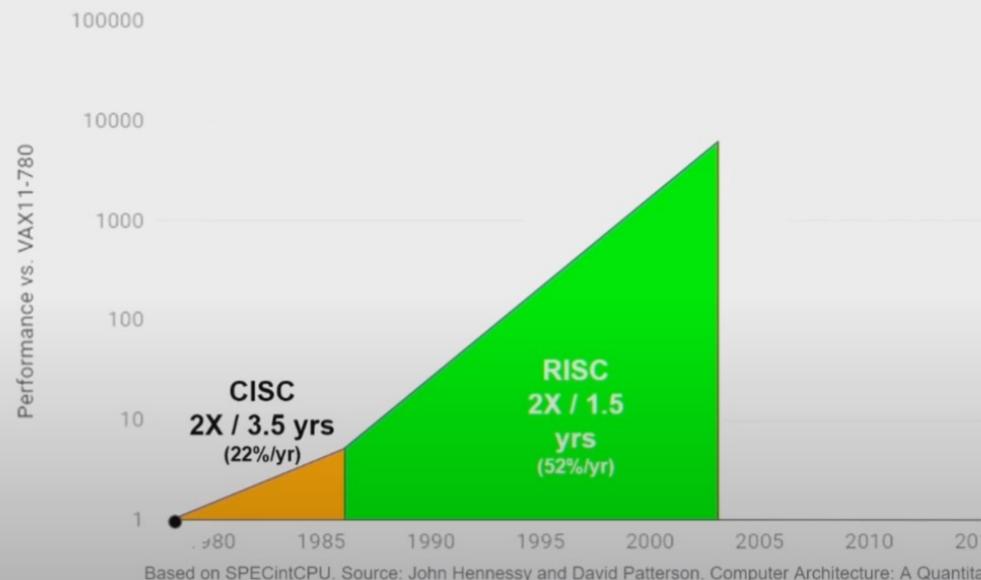
재료의 파손 해석(좌) 및 실제(우)

- ✓ 자연과학/공학 분야를 불문하고 물리현상을 묘사할 수 있는 **미분방정식**이 존재함
- ✓ 미분방정식 중에는 연필로 풀 수 있는 수학논리적 해법이 없는 난제들이 존재
- ✓ 그러한 난제들을 풀 수 있는 유일한 방법은 (**고성능**)컴퓨터를 활용한 **계산**으로 접근



End of Growth of Single Program Speed?

40 years of Processor Performance



22

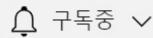
37 · Processors >



John Hennessy and David Patterson 2017 ACM A.M. Turing Award Lecture



Association for Computing Machinery (ACM)
구독자 4.31만명



출처 - <https://www.youtube.com/watch?v=3LVeEjsn8Ts&t=3681s>

- 12 -

마이크로 아키텍처 주요 개념: ISA, ALU, FPU, Cache, Register, ...

◆ Microarchitecture (u-arch): 마이크로프로세서 하나를 만드는데 필요한 알고리즘 및 회로 수준의 구조를 자세히 정의해 놓은 것

◆ 명령어(Instruction): 프로세서가 이해하는 지시

- 1) 기본적인 사칙 연산, 논리 연산
- 2) 메모리에 쓰고 읽는 명령
- 3) 프로그램의 실행 흐름을 제어하는 분기, 호출 명령

◆ 명령어 집합 구조(ISA, Instruction Set Architecture)

- : 프로그래머와 프로세서가 직접적으로 소통할 수 있게 하는 언어
- 프로그래밍 방법론을 정의할 뿐 아니라 프로세서 구현의 많은 부분도 ISA가 결정
- 명령어 종류, 피연산자 타입, 레지스터 개수, 인코딩 방법 등 여러 가지를 ISA가 정의 예) CISC (X86), RISC(ARM)
- 인코딩(encoding): 명령어를 기계어로 변환하는 것
- 디코딩(decoding): 기계어를 명령어로 변환하는 것

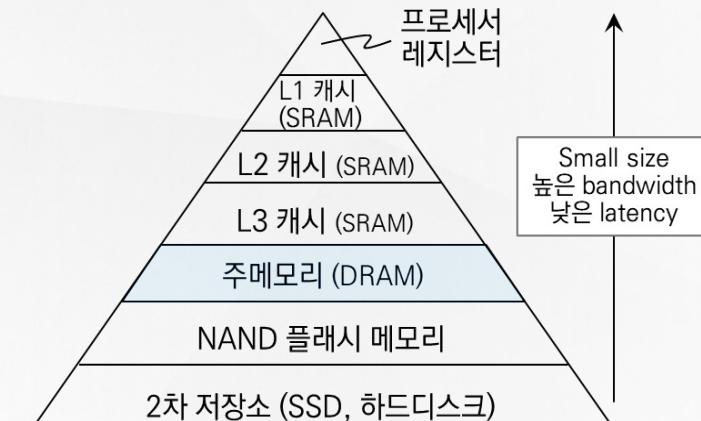
◆ 산술 논리 장치 (ALU, Arithmetic Logical Unit)

- ALU가 지원하는 연산
- 1) 정수 사칙 연산: + - * /
- 2) 비트 논리 연산: AND, OR, XOR, 1의 보수
- 3) 비트 시프트 연산: 왼쪽(<<), 오른쪽(>>)

◆ 부동소수점 (FP, floating point)



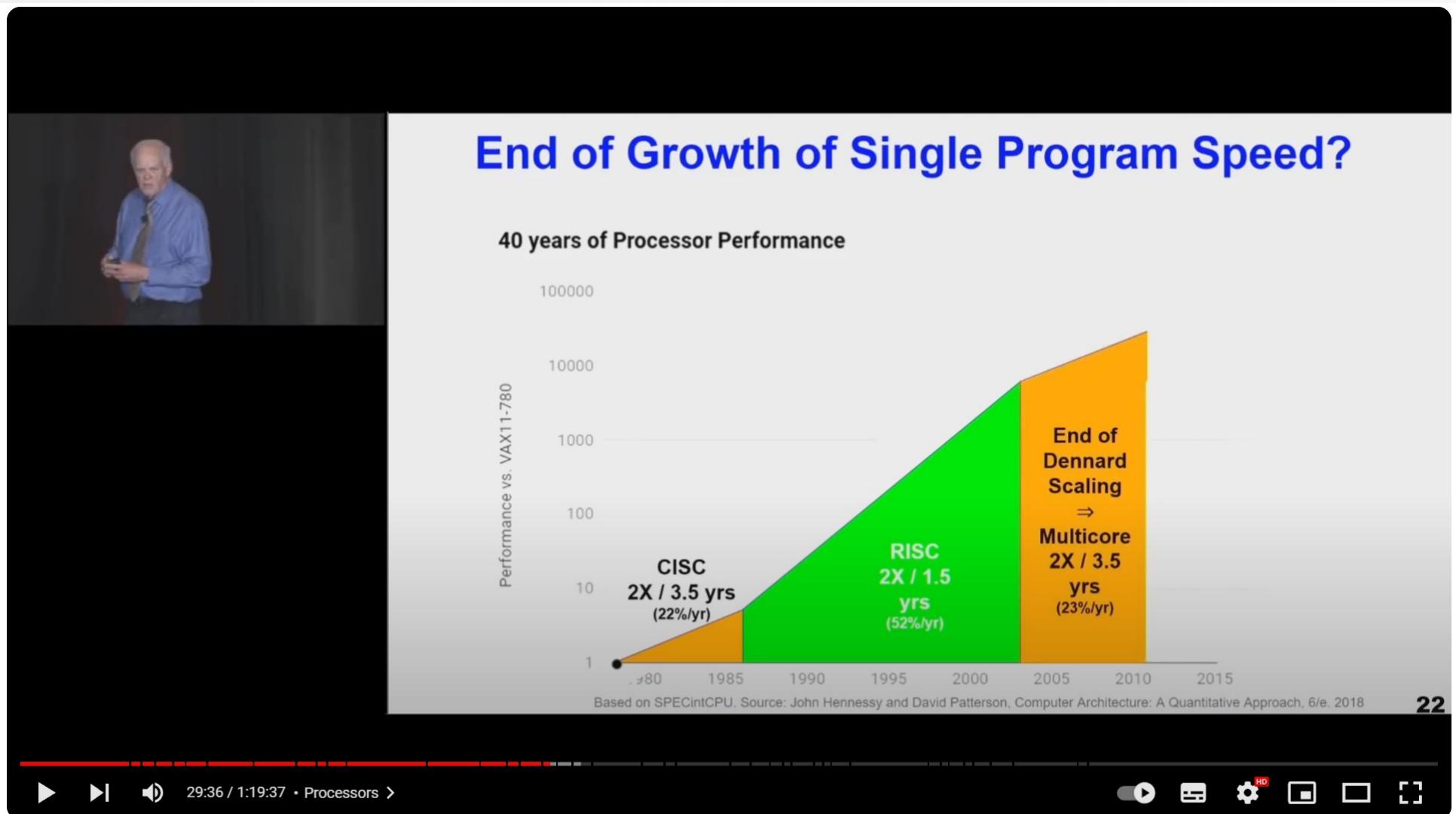
◆ Memory Hierarchy (메모리 계층)



◆ Control Unit: 명령어를 메모리에서 읽어와서 해독하고 이를 적절한 ALU에 넣어 결과 값을 얻게 하는 기능부를 통칭하는 개념 ex) 클럭수

◆ Process & Thread

- Process: 실행 중인 프로그램
- Thread: 프로세서를 사용하는 최소 단위, 여러 스레드가 하나의 프로세스 내에 만들어질 수 있음

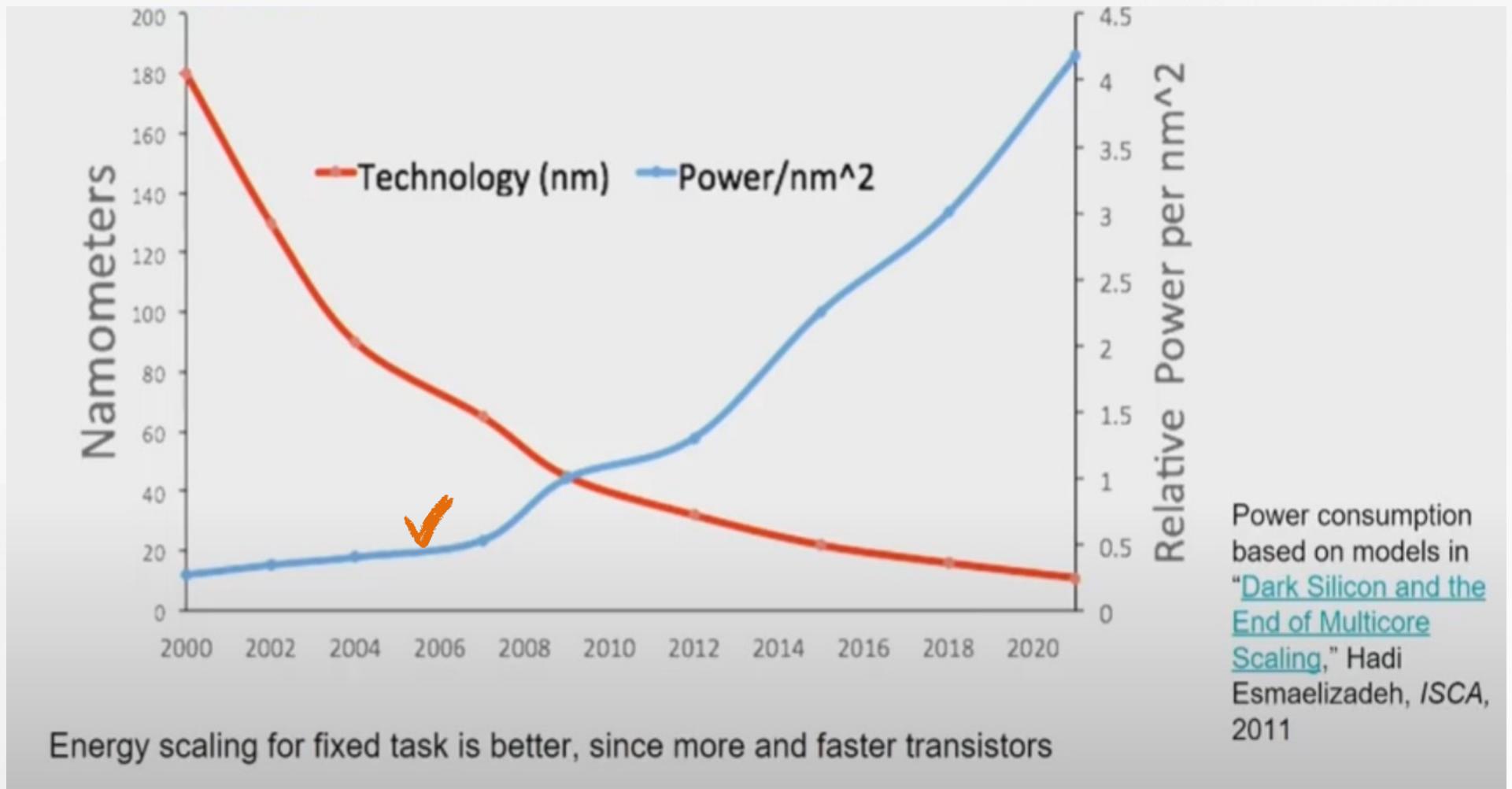


John Hennessy and David Patterson 2017 ACM A.M. Turing Award Lecture

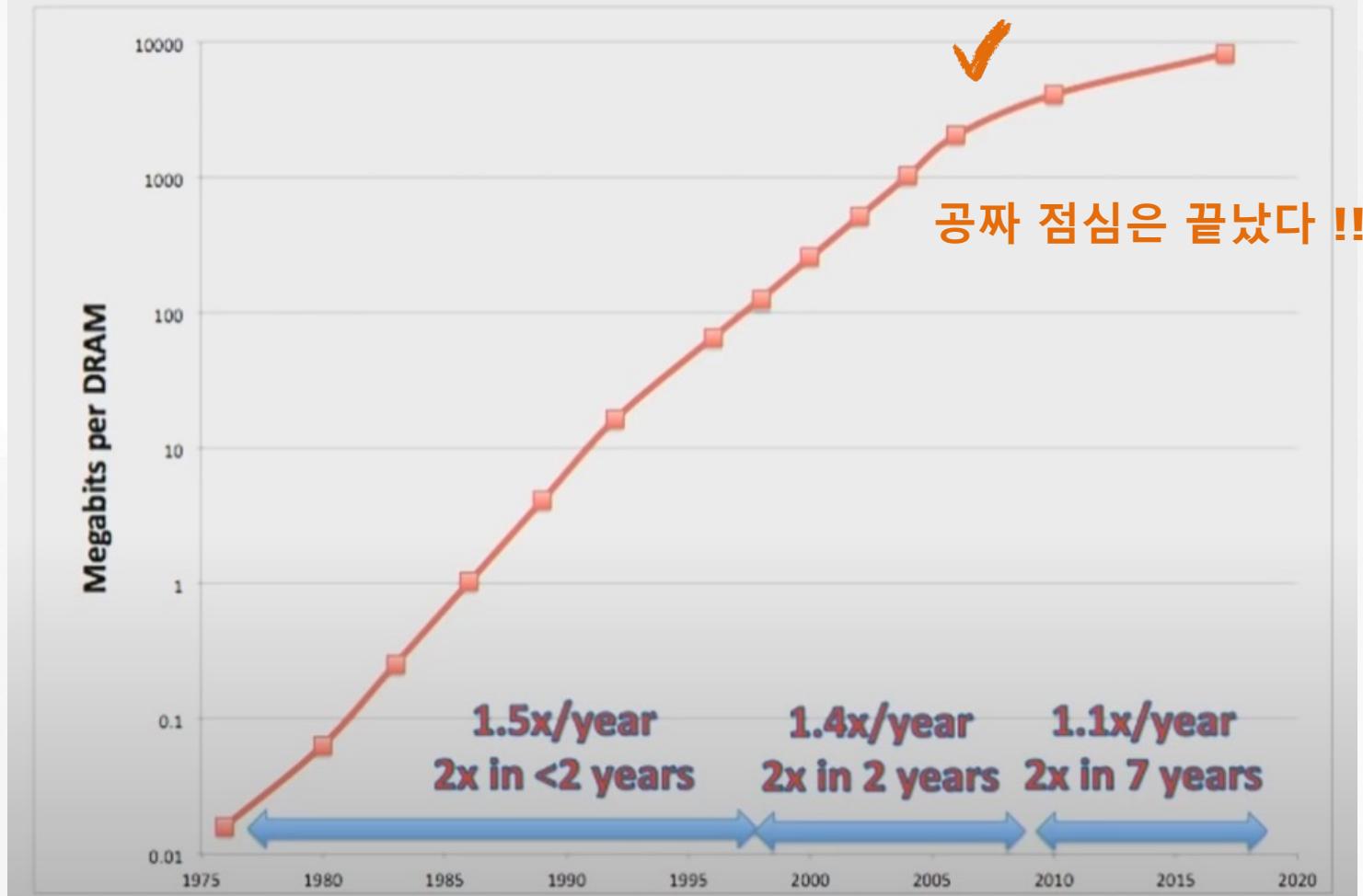
프로세서 아키텍처: 데너드 스케일링 법칙

Courtesy of John Hennessy & David Patterson

Moore 법칙이 만난 한계와 동일

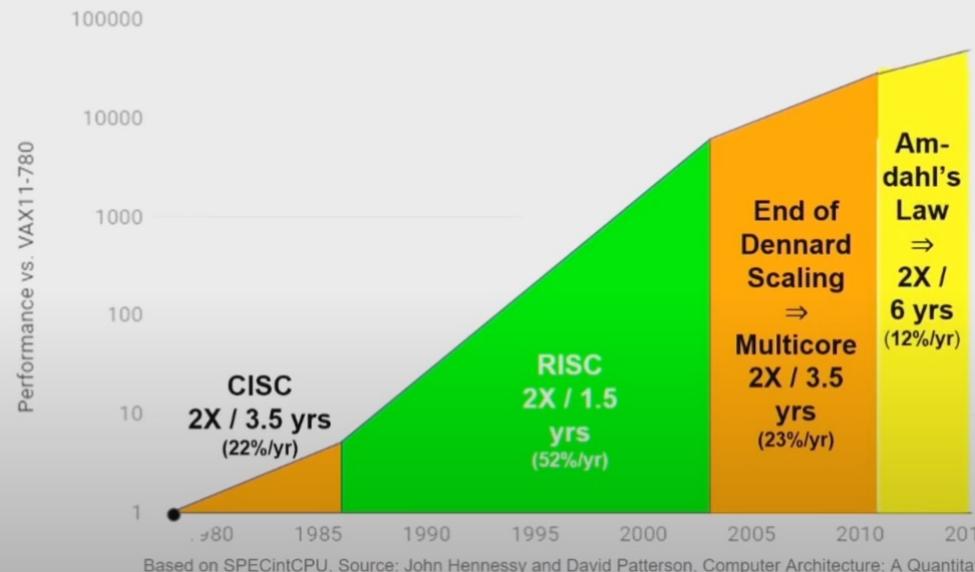


Moore's Law in DRAMs



End of Growth of Single Program Speed?

40 years of Processor Performance



22

▶ ▶ 🔍 29:45 / 1:19:37 · Processors >

▶ 🔍 ⚙️ 📺 🎞️ 🎵

John Hennessy and David Patterson 2017 ACM A.M. Turing Award Lecture



Association for Computing Machinery (ACM)
구독자 4.31만명

구독중 ▾

1.3천



공유

오프라인 저장

클립

저장

...

하드웨어 아키텍처: 암달의 법칙

✓ 계산 코어를 n배 증가시킨다고 해서, 계산 시간이 1/n이 되는 것은 아님

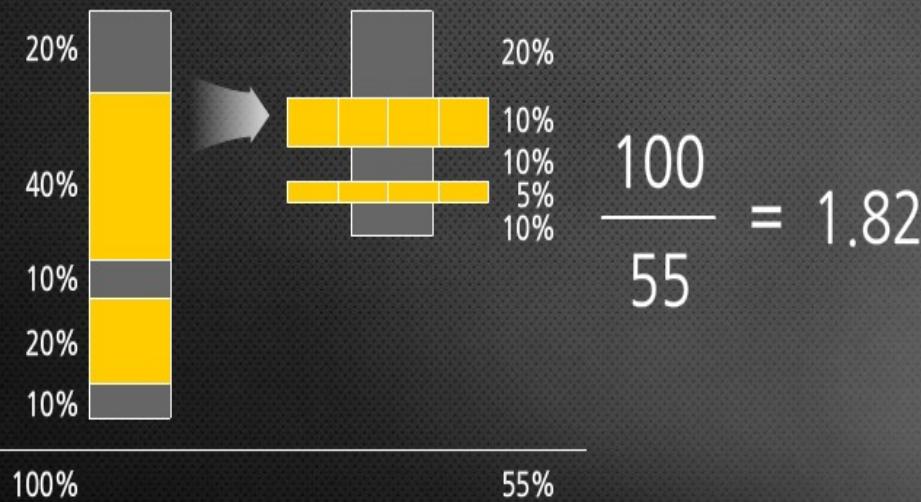
Amdahl의 법칙

1 컴퓨터 프로그램의 일부를 n개의 프로세서를 위해 병렬화 하였을 때 전체적으로 얼마만큼의 최대 성능 향상이 있는지 계산하는 법칙

- Speedup의 계산
- 순차 프로그램에 비해 병렬 프로그램이 몇 배 빨라졌느냐?

2 다른 오버헤드를 생각하지 않은 이상적인 경우를 가정

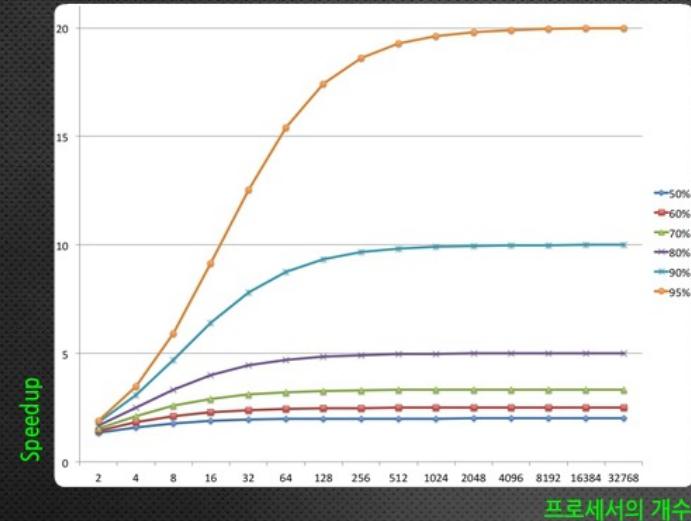
3 순차 실행시간의 60%를 차지하는 부분을 4개의 프로세서를 사용하여 병렬화



- ✓ p : 병렬화 된 부분이 순차 실행에서 차지한 비율
- ✓ 1 - p : 병렬화 되지 않은 부분이 순차 실행에서 차지한 비율
- ✓ n : 프로세서의 개수

$$\text{speedup} = \frac{1}{(1-p) + \frac{p}{n}}$$

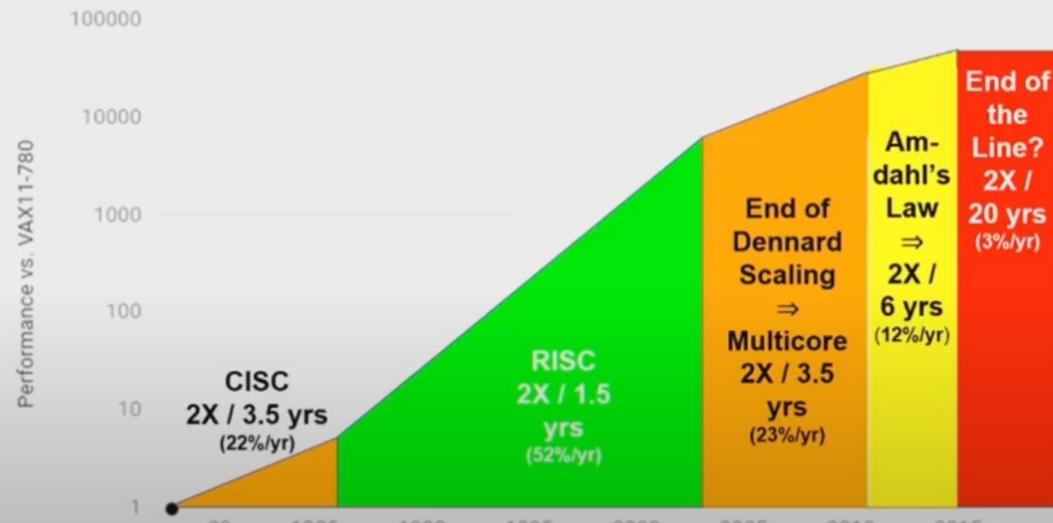
$$\text{speedup} = \frac{1}{(1-0.6) + \frac{0.6}{4}} = \frac{1}{0.55} = 1.82$$



- ✓ 순차실행시간의 95%를 차지하는 부분을 32768개의 프로세서를 써서 병렬화 하더라도 speedup은 약 20밖에 되지 않음
- ✓ 프로그램을 병렬화 했을 때의 speedup은 프로그램 내에서 병렬화 할 수 없는 부분이 차지하는 실행시간에 의해 주된 영향을 받음

End of Growth of Single Program Speed?

40 years of Processor Performance



22

▶ ▶ 🔍 29:58 / 1:19:37 · Moores Law >

▶ 🔍 ⚙️ 📺 ⚡

John Hennessy and David Patterson 2017 ACM A.M. Turing Award Lecture



Association for Computing Machinery (ACM)

구독자 4.3만명

구독중

1.3천

👎

▶ 공유

▷ 오프라인 저장

⌘ 클립

저장

...

What's the Opportunity?

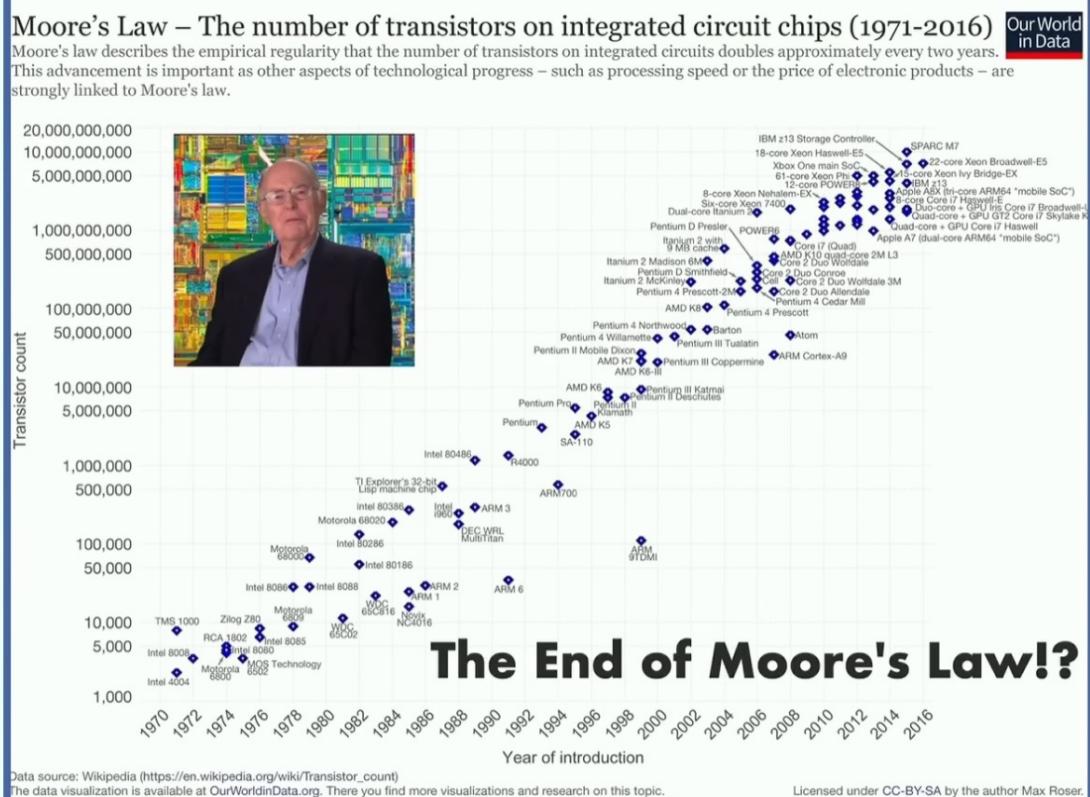
Matrix Multiply: relative speedup to a Python version (18 core Intel)

Version	Speed-up	Optimization
Python	1	
C	47	Translate to static, compiled language
C with parallel loops	366	Extract parallelism
C with loops & memory optimization	6,727	Organize parallelism and memory access
Intel AVX instructions	62,806	✓ Use domain-specific HW

from: "There's Plenty of Room at the Top," Leiserson, et. al., to appear.

Computer Graphics 분야: 또 다른 계산 집약 Domain

Courtesy of Pat Hanrahan & Ed Catmull



2019 AM Turing Award Recipients Ed Catmull and Pat Hanrahan Turing Lectures



Association for Computing Machinery (ACM) ✓

구독자 4.3만명

구독중 ▾

90

...

공유

오프라인 저장

클립

저장

...

Computer Graphics 분야: 무어 법칙의 승리

Courtesy of Pat Hanrahan & Ed Catmull



100 million hours of CPU time!

29 hours/frame =
29*60*60 seconds/frame
104,400 seconds/frame

100K seconds/frame *
10 GFLOPs = ~1 PFLOP
1 PFLOP / 1 Megapixel =
~ 1 GFLOP / pixel



“이 영화를 계산하는 데 제가 처음 픽사에 왔을 때(1980년) 성능으로 1억 시간의 CPU 시간이 걸립니다. 계산을 해보면, 프레임당 1페타플롭의 컴퓨팅을 사용합니까 1983년 당시 우리 계산 능력보다 약 100만 배 더 많은 것입니다. 그리고 1983년부터 2013년까지 생각해 보면 30년입니다. 그리고 30년 동안, 모두가 무어의 법칙을 기억할 것입니다. 또는 성능에 대한 무어의 법칙입니다. 다시 이야기하겠지만, 대략 컴퓨터 성능만큼 증가하고, 컴퓨터 성능은 5년마다 대략 10배씩 증가합니다. 그래서 계산을 한다면 30년은 10의 6승 또는 백만 배가 됩니다. 그리고 이 모든 것이 컴퓨팅 파워의 증가로 가능해졌다는 것을 알 수 있습니다.”

2019 AM Turing Award Recipients Ed Catmull and Pat Hanrahan Turing Lectures



Association for Computing Machinery (ACM)
구독자 4.3만명

구독중 ▾

90

공유

Computer Graphics 분야: GPGPU의 탄생

Courtesy of Pat Hanrahan & Ed Catmull

Data-Parallel Programming

map - apply function to a collection

- Draw triangle and apply fragment shader

filter

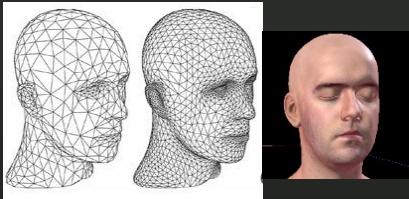
- Stenciling, kill fragment

gather

- Texture lookup

scatter

reduce



GPGPU

Stanford Brook, Buck et al., 2004

- Data-parallel virtual machine
- No need to be a graphics programmer

CUDA, 2007

- Blocks of thread groups
- Expose memory
- Synchronization primitives

삼각메쉬에서 생성된 모든 조각에 대해 셰이더 프로그래밍을 실행하는 것을 생각했다면, 조각 모음에 함수를 적용할 수 있었습니다. 그리고 그것은 map 함수 기능과 동일했습니다. 그런 다음 filter라는 것이 있었는데, 이것으로 여러 가지를 제거할 수 있습니다. 이제 일부를 가져와 메모리를 가지고 주소를 지정하고 수집하는 gather라는 것이 있었고, 데이터 병렬 프로그래밍이 있어서 전체 주소 집합을 지정하고 모두 수집하지만 GPU가 그다지 잘하지 못했던 두 가지 기본 함수가 있었습니다. 무작위 위치에 물건을 쓰고 모든 메모리를 분산(scatter) 시키고 축소(reduce)하는 것은 병렬연산에서 벡터를 합하는 것과 같습니다. 이것은 꽤 간단했고, GPU를 조금 조정하면 이런 일을 할 수 있을 겁니다. 그리고 이런 일을 했다면, 우리는 일종의 범용 데이터 병렬 컴퓨터와 동등해집니다. 그리고 그것이 제 학생 Ian Buck이 2004년에 만든 Brook이라는 시스템으로 이어졌습니다. 그리고 그것은 정말 간단한 아이디어였습니다. 여러분은 "GPU를 데이터 병렬 가상 머신처럼 작동하게 만들면 그래픽 프로그래머가 아니어도 사용할 수 있습니다."라고 말씀하셨을 뿐입니다. 그래서 사람들은 이 무렵 GPU에서 다양한 알고리즘을 실행하는 실험을 시작했는데 그래픽 프로그래머가 되어야 했습니다. GPU 프로그램을 실행하려면 삼각형을 렌더링해야 하고, 그 사용법을 배워야 했습니다.

Computer Graphics 분야: GPGPU-CUDA

CUDA (Compute Unified Device Architecture)

IT > ICT

[인터뷰] AI 컴퓨팅 혁명 주도 엔비디아 ‘쿠다’ 창시자 이언 벅… “모든 기업과 협력하는 AI 플랫폼 기업”

이언 벅 엔비디아 가속 컴퓨팅 부문 총괄 겸 부사장 인터뷰
“엔비디아, 모든 기업과 협력하는 유일한 AI 회사”
20년 전부터 ‘개방성’ 강조... 개발자 400만명이 생태계 지탱
“AI 시장 경쟁자 환영... 누구와도 협력 가능”

황민규 기자 샌타클라라(미국)=최지희 기자

업데이트 2024.01.16. 16:56 ▾



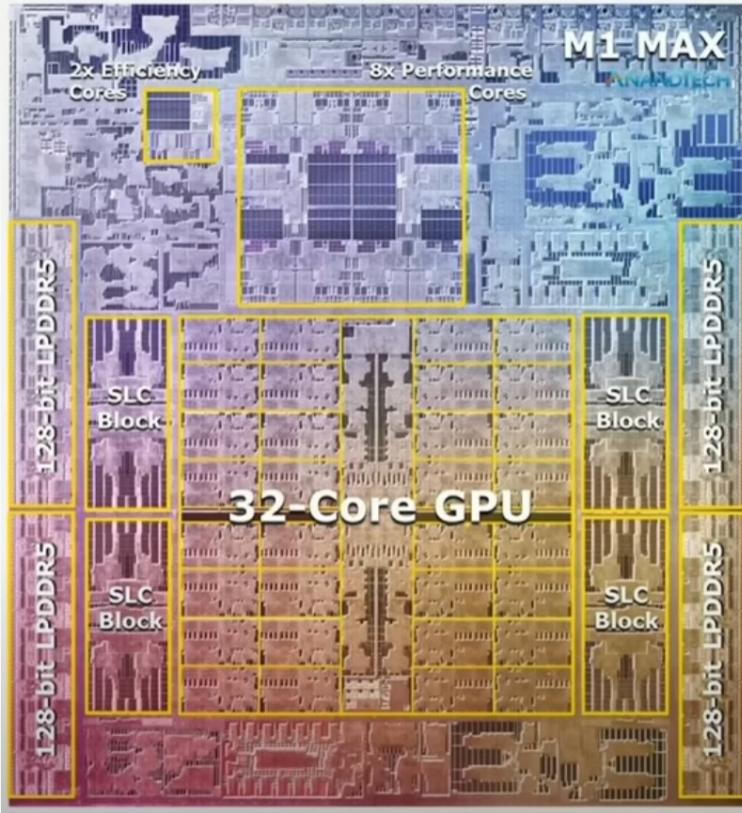
“반도체의 회로 집적도를 높여 성능을 향상시키는 ‘무어의 법칙’은 이제 끝났다. 무어의 법칙은 미세공정의 진화에 따른 하드웨어적 성능 향상만을 강조하지만, 엔비디아는 모든 소프트웨어 계층에서 혁신이 가능하다.”

“X86으로는 불가능한 AI 솔루션을 개발하기 위해서는 새로운 발상의 전환이 필요했으며, 엔비디아는 이를 내다보고 지난 20여년간 새로운 컴퓨팅 명령어 체계를 구상해 왔고 그 결과물이 바로 쿠다(CUDA)이다.”



이언 벅 엔비디아 가속 컴퓨팅 부문 초고객 부사장 /엔비디아 제공

하드웨어 아키텍처의 지속 진화



“여기 Apple의 M1 Max가 있습니다. 다이 평면도입니다. 그리고 여기에 있는 모든 컴퓨팅 장치들을 보세요. 코덱이 있고, 압축 칩이 있고, 보안 칩이 있습니다. 일반 코어 8개, 고성능 코어가 있습니다. I/O를 위한 저성능 코어 2개가 있습니다. 그리고 32 개 코어 GPU가 있습니다. GPU가 CPU보다 훨씬 더 컸다는 것을 알기 때문에 이 그림은 아름답습니다. 컴퓨팅 성능 측면에서 생각해 보면 더 아름답습니다. 그리고 초기 GPU에는 1,700만 개의 트랜지스터가 있었습니다. 이 칩에는 570억 개의 트랜지스터가 있습니다.”

Courtesy of Pat Hanrahan

TOP500 LIST - JUNE 2024



R_{max} and R_{peak} values are in PFlop/s. For more details about other fields, check the TOP500 description.

R_{peak} values are calculated using the advertised clock rate of the CPU. For the efficiency of the systems you should take into account the Turbo CPU clock rate where it applies.

Rank	System	Supercomputer의 내부 아키텍처 조합은 매년 바뀝니다	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	Frontier - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE	DOE/SC/Oak Ridge National Laboratory United States	8,699,904	1,206.00	1,714.81	22,786
2	Aurora - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel	DOE/SC/Argonne National Laboratory United States	9,264,128	1,012.00	1,980.01	38,698
3	Eagle - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Azure	Microsoft Azure United States	2,073,600	561.20	846.84	
4	Supercomputer Fugaku - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu	RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
5	LUMI - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC		2,752,704	379.70	531.51	7,107

Key Takeaways

- ✓ ‘계산’의 중요성: 최근 튜링상 수상자들의 공헌
- ✓ 계산성능 향상의 3가지 접근: 하드웨어/소프트웨어/알고리즘
- ✓ 혁신의 전이: HPC → 병렬연산 → 그래픽스 분야


```
graph TD; A[HPC] --> B[병렬연산]; B --> C[그래픽스 분야]; C --> D[새로운 칩 M1, NPU 등]; C --> E[GPGPU CUDA]; E --> F[AI 분야]; D <--> E;
```
- ✓ 몇몇 개념들: 마이크로 아키텍처

AI 컴퓨팅 인프라 역사

그림출처 – Human-centered Artificial Intelligence

Geoffrey Hinton
CC FRS FRSC



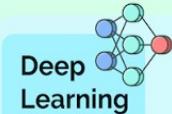
Hinton speaking at the University of Toronto

Emergence of...
Homogenization of...

Machine Learning



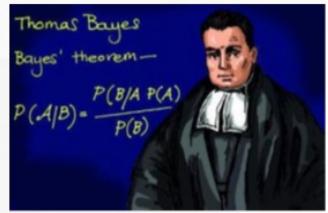
Deep Learning



Foundation Models

features
architectures

functionalities
models



-Bayes(1701~1761):
“새로운 정보가지식을 양신”
(기계학습기초 철학)

-각종 계산기계 출현

01

Computing power used in training AI systems
(OpenAI 참고)

- 언어 ● 대화 ● Vision
- 게임 ● 기타

18C~

1950~70

1970~90

AlphaGo, game Go

AlexNet, CNN image 분류

LLM

100K
10K
1K
100
10
1
0.1
0.01
0.001
0.0001
0.00001
0.000001

* PetaFLOPS = 10^{15}

02

First Perceptron : 최초 학습기계

03

Moore's Law(1965)
2년에 2배 증가

First Era

Modern Era

2010~2020

2020~

04

1990~2010

05

LLM Era

-Turing: 컴퓨팅 계산이론 정립, 생각하는 기계 테스트
-AI 용어 출현: 디트머스 컨퍼런스
-챗봇 Eliza
-DARPA의 AI 연구비 투자
-전자식, Von Neumann 범용 컴퓨터 등

-Minsky: 인간수준 기계 예고
-한계: Moravec's paradox, XOR 문제 등
-뉴럴넷, 전문가 시스템
-카네기멜론 대자율주행차
-마이크로프로세서, 상용 PC 등

-AI 상업화: DeepBlue 체스
-돌파: 역전파 알고리즘, 컴퓨팅 파워(GPU 등)
-Dragon 음성인식 서비스 인터넷 등

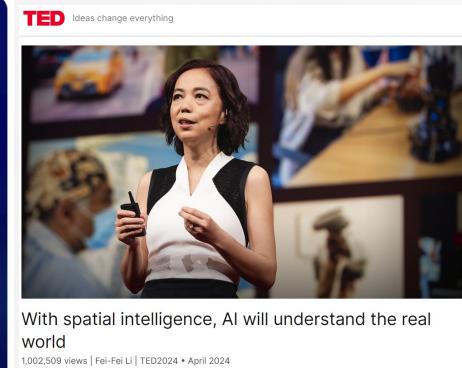
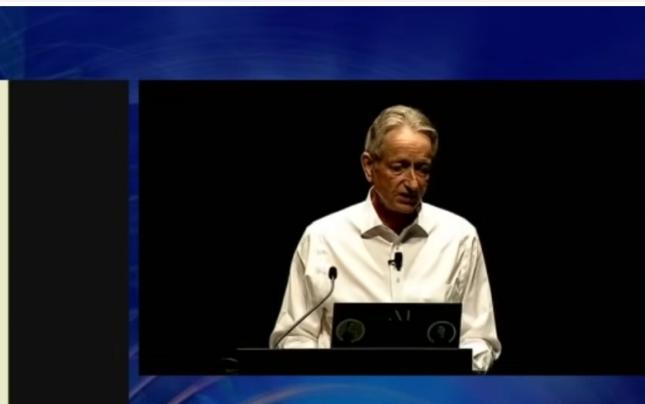
-IBM Watson 퀴즈 쇼
-Google 알파고 바둑
-AI Ubiquitous
-TPU/XPU 등 새로운 아키텍처, 스마트폰, 빅데이터, 새로운 컴퓨팅 등

인공지능 도메인의 성공 요인: 알고리즘 – BigData – GPU

Courtesy of Geoffrey Hinton

The return of backpropagation

- Between 2005 and 2009 researchers (in Canada!) made several technical advances that enabled backpropagation to work better in feed-forward nets.
 - Unsupervised pre-training; random dropout of units; rectified linear units.
 - The technical details of these advances are very important to the researchers but they are not the main message.
 - The main message is that backpropagation now works amazingly well if you have two things:
 - a lot of labeled data
 - a lot of convenient compute power (e.g. GPUs)



With spatial intelligence, AI will understand the real world
1,002,509 views | Fei-Fei Li | TED2024 • April 2024

“(AI 알고리즘 성공에서) 주목할 메시지는 몇 가지 기술적 발전[슬라이드 내용]으로 역전파가 놀라울 정도로 잘 작동한다는 것입니다. 그리고 주된 이유는 이제 레이블이 지정된 데이터가 많고 편리한 컴퓨팅 파워가 많기 때문입니다. 불편한 컴퓨팅 파워는 별로 쓸모가 없습니다. 하지만 GPU와 최근의 TPU 같은 것들은 많은 계산을 할 수 있게 해주고 엄청난 차이를 만들어냈습니다. 그래서 결정적인 요인은 컴퓨팅 파워의 증가라고 생각합니다. 그래서 저는 딥러닝의 공로가 Fei Fei Lee와 같은 대규모 데이터베이스를 만든 사람들과 David Patterson처럼 컴퓨터를 빠르게 만든 이들에게 있다고 생각합니다.”

Geoffrey Hinton and Yann LeCun, 2018 ACM A.M. Turing Award Lecture "The Deep Learning Revolution"

Association for Computing Machinery (ACM) [구독자 4.3만명](#)

구독중

좋아요

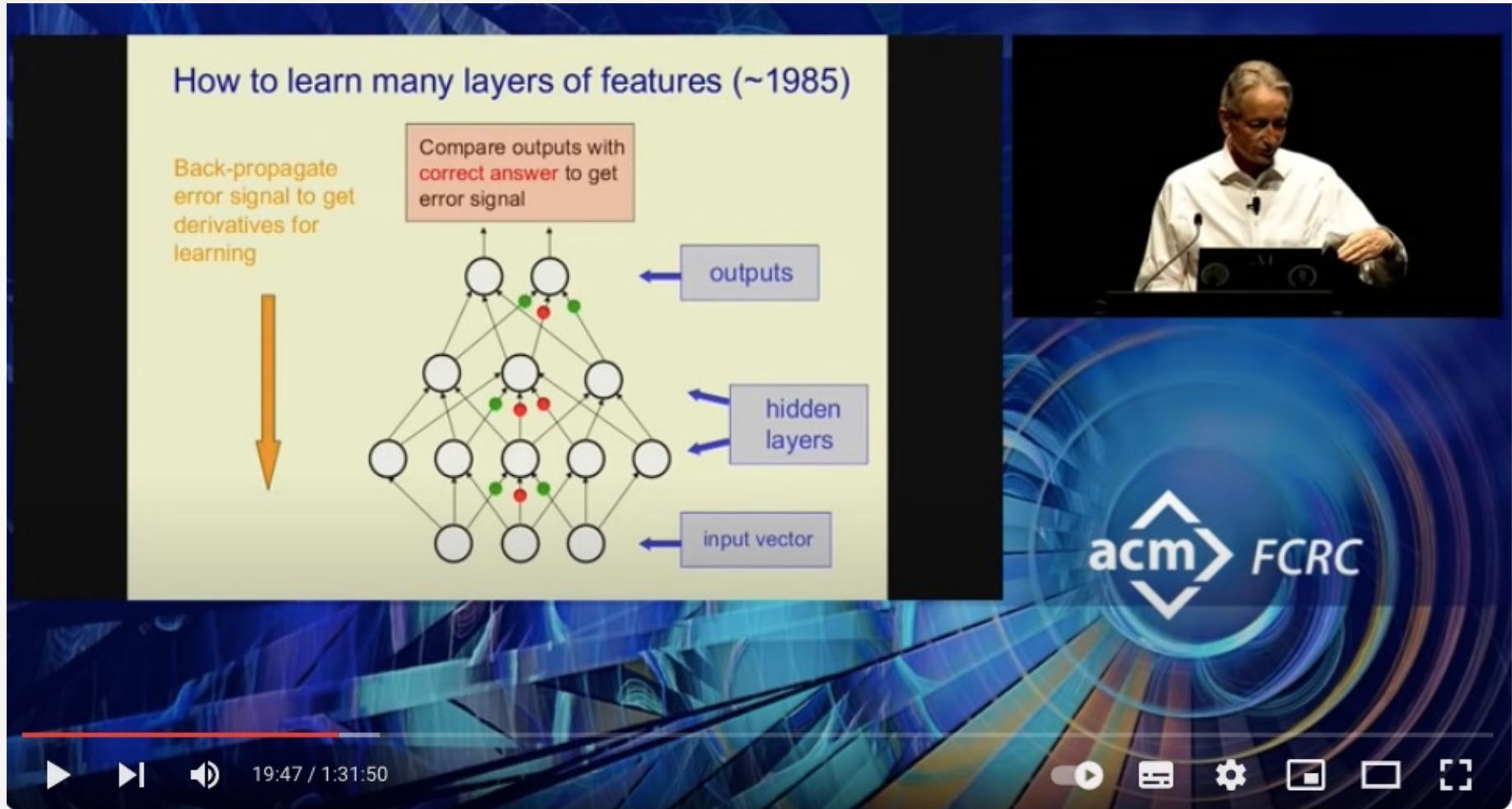
댓글

공유

↓

딥러닝 핵심 알고리즘: Back-Propagation

Courtesy of Geoffrey Hinton



Geoffrey Hinton and Yann LeCun, 2018 ACM A.M. Turing Award Lecture "The Deep Learning Revolution"



Association for Computing Machinery (ACM) 

구독자 4.3만명

구동

 좋아요

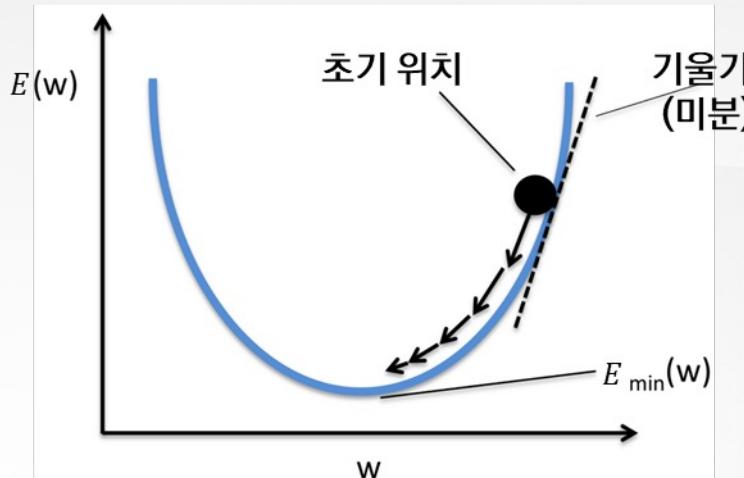
9

▶ 공유

↓ 오프라인 저장

* * *

딥러닝 핵심 알고리즘: Back-Propagation 이해 해보기 (1) - 기본 수학



미분 = 접선 기울기
Differentiation

$$z = f(y), y = g(x)$$

$$\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx} = f'(y)g'(x)$$

미분 연쇄 법칙
Chain rule

$$Z = WX$$

$$\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

벡터, 행렬
Vector, Matrix

[참고] 딥러닝 관련 수학 요약

데이터
Data

$$Z = WX$$

$$\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

선형대수학
Linear Algebra, Vector, Matrix

예측
Prediction

$$P(x|z) = \frac{P(x)P(z|x)}{P(z)}$$

$$P(B|A) = \frac{P(A \cap B)}{P(A)}$$

확률론 Probability theory

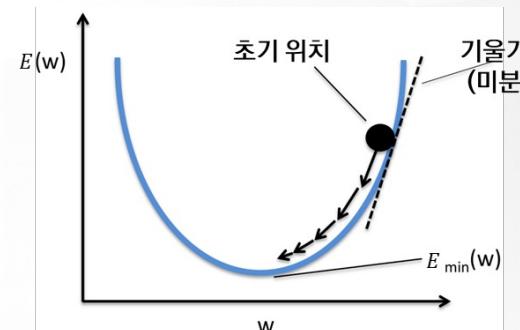
판단
Loss function

$$H(X) = - \sum_x P(x) \log P(x)$$

통계학 Statistics

최적화
Optimization

$$\theta' = \theta - \alpha \frac{\partial \text{Error}}{\partial \theta}$$



미적분학 Calculus

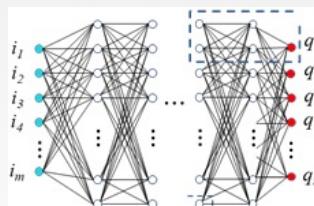
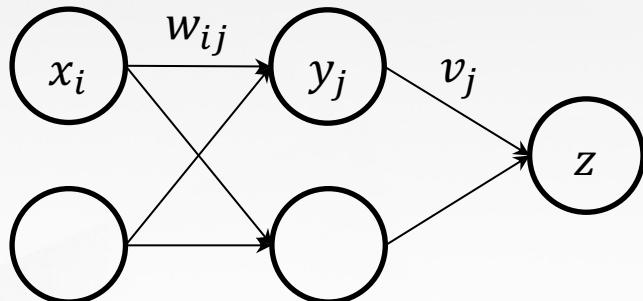
$$z = f(y), y = g(x)$$

$$\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx} = f'(y)g'(x)$$

미분 연쇄 법칙
Chain rule

딥러닝 핵심 알고리즘: Back-Propagation 이해 해보기 (2)

Courtesy of Geoffrey Hinton



✓ 알고리즘으로 대규모 컴퓨팅 가속

NATURE VOL. 323 9 OCTOBER 1986

Learning representations
by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton†
& Ronald J. Williams*

* Institute for Cognitive Science, C-015, University of California,
San Diego, La Jolla, California 92093, USA

† Department of Computer Science, Carnegie-Mellon University,
Pittsburgh, Philadelphia 15213, USA

$$y_j = \sigma \left(\sum_i w_{ij} x_i \right)$$

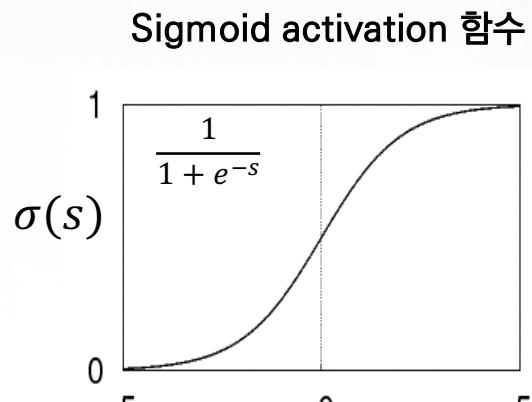
$$z = \sigma \left(\sum_j v_j y_j \right)$$

$$\text{Error} = \frac{1}{2} (z - \hat{z})^2$$

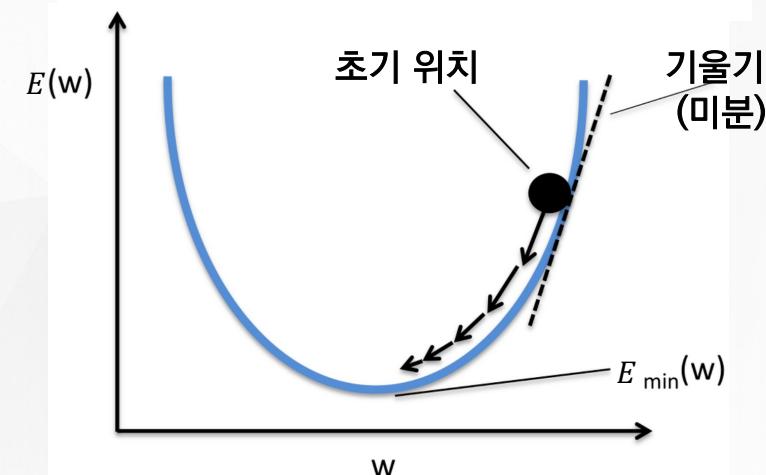
$$\theta' = \theta - \alpha \frac{\partial \text{Error}}{\partial \theta}$$

$$\frac{\partial \text{Error}}{\partial v_j} = \frac{\partial \text{Error}}{\partial z} \frac{\partial z}{\partial v_j} = (z - \hat{z}) \frac{\partial z}{\partial v_j} = (z - \hat{z}) \sigma'(s) y_j = \delta \cdot \sigma'(s) y_j$$

$$\frac{\partial \text{Error}}{\partial w_{ij}} = \frac{\partial \text{Error}}{\partial z} \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial w_{ij}} = [\delta \cdot \sigma'(s) v_j] \sigma'(s_j) x_i = \delta_j \cdot \sigma'(s_j) x_i$$



$$\text{미분 } \sigma' = \sigma \cdot (1 - \sigma)$$



AlexNet (2012)

Courtesy of Geoffrey Hinton

Google search results for "AlexNet paper". The top result is the paper "ImageNet Classification with Deep Convolutional Neural Networks" by Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. The diagram illustrates the AlexNet architecture, showing two parallel GPU paths (GPU 1 and GPU 2) processing input images through various layers including convolutional and fully connected layers.

ImageNet Classification with Deep Convolutional Neural Networks

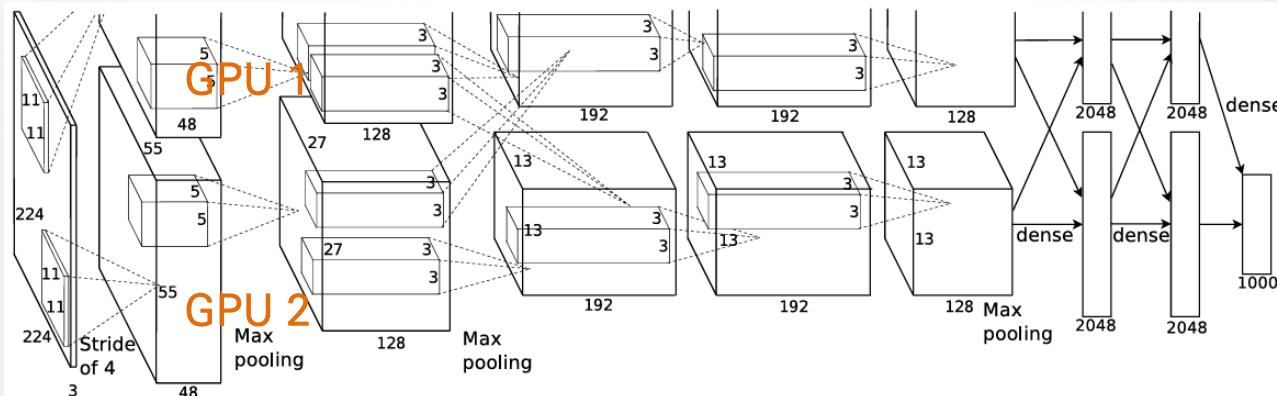
Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

ImageNet Classification with Deep Convolutional Neural ...

A Krizhevsky 저술 · 132223회 인용 ✓ The specific contributions of this paper are as follows: we trained one of the largest convolutional neural networks to date on the subsets of ImageNet...



AlexNet을 개발할 당시 사용한 GPU는 GTX580로, 3GB 메모리만 가지고 있기 때문에 한 개에 GPU에서 연산하기에는 한계가 있었고, 따라서 개발자들은 아래 그림과 같이 Network를 두 개의 GPU로 나눌 수 있는 구조로 설계

cf. NVIDIA A100은 80 GB 메모리

CNN 알고리즘 적용의 정점: 자율주행

Courtesy of Yann LeCun

Driving Cars with Convolutional Nets

Y. LeCun

▶ MobilEye (2015)

▶ NVIDIA

acm > FCRC

53:19 / 1:31:50

Geoffrey Hinton and Yann LeCun, 2018 ACM A.M. Turing Award Lecture "The Deep Learning Revolution"



Association for Computing Machinery (ACM)

구독자 4.3만명

구독

좋아요

싫어요

공유

오프라인 저장

...

언어모델과 성능 이슈: GPU 메모리

Courtesy of Geoffrey Hinton & Yann LeCun

“2014년 이래로 주요 개발 중 하나는 문장의 의미를 디코딩할 때 인코딩했던 문장을 다시 보는 것입니다. 이를 소프트 어텐션이라고 합니다. 따라서 새로운 단어를 생성할 때마다 번역하고 있는 문장에서 어디를 볼지 결정합니다. 그게 많은 도움이 됩니다. 또한 이제 단어 임베딩을 사전 학습합니다. 그리고 그게 많은 도움이 됩니다… 그들은 트랜스포머라는 것을 사용하는데, 이 신경망에서 각 단어가 트랜스포머를 통과할 때 근처의 단어를 보고 의미가 무엇인지 구분합니다. 따라서 "may"와 같은 단어가 있고 그 옆에 "13일"이 있으면 그것이 그 달을 의미하는 것을 잘 알고 있습니다. 그러므로 다음 영역에서는 그것을 모호하지 않게 할 수 있고 그 5월이라는 의미가 됩니다.

그들은 또한 많은 문법을 배우는 것으로 밝혀졌습니다. 따라서 언어학자들이 본능적으로 넣어야 한다고 생각했던 모든 것들을 이 신경망이 이제 거기에 넣고 있습니다. 그들은 많은 구문적 이해를 얻고 있습니다. 하지만 그것은 모두 오직 데이터로부터 학습이 됩니다… 기본적으로 그들은 문장 자체만 보고도 어린아이가 문법을 배우는 방식으로 문법을 배우고 있습니다.”

“신경망을 사용하여 번역을 할 수 있다는 사실의 매우 놀랍고 흥미로운 발전이라고 생각합니다. 그리고 이를 위해 사용되는 아키텍처 종류에는 많은 혁신이 있습니다. 그래서 Geoffrey는 어텐션 메커니즘, 트랜스포머 아키텍처에 대해 이야기했습니다. 이것은 동적 합성곱이라고 하는 새로운 아이디어를 약간 재활용한 것입니다. 그리고 그곳에서 모든 것이 정말 잘 작동합니다. 그러한 네트워크는 매우 큅니다. 그 언어모델에는 수억 개의 매개변수가 있습니다. 그래서 거기에서 몇 가지 과제는 실제로 GPU에서 실행하는 것입니다. 우리는 기본적으로 GPU 메모리에 의해 제한되므로, 실행할 충분한 메모리가 있어야 합니다.”

언어모델에서의 성능 이슈: GPU 메모리

Google 삼성 엔비디아 HBM 테스트 X | ☰ ⌂ 🔍 ⌂

 디지털투데이
<https://www.digitaltoday.co.kr> > news > articleView :

삼성전자 HBM, 엔비디아 퀄테스트 결과 언제쯤?...내달 공개 ...

2024. 7. 17. — 업계 관계자는 삼성전자의 12단 HBM3E에 대해 "36GB 용량을 가져 AI 가속기에 탑재될 HBM 수를 줄일 수 있다"며 "테스트 결과 공개는 보다 앞당겨져 8월 ..."

 한국경제
<https://www.hankyung.com> > article :

삼성전자, 8월 목표로 엔비디아 테스트 통과 준비...지금 사 ...

2024. 7. 5. — 밸류에이션(실적 대비 주가 수준) 매력이 있어 엔비디아의 고대역폭메모리(HBM) 품질 테스트를 통과하면 주가가 바로 상승할 것이라 전망에서다. 다만 ...

 비즈니스포스트
<https://www.businesspost.co.kr> > ... :

외신 "삼성전자 HBM3 엔비디아 인증 통과" ...

2024. 7. 23. — [비즈니스포스트] 삼성전자가 엔비디아의 고대역폭메모리(HBM) 품질테스트를 공식 통과했다는 외신 보도가 나왔다. 하지만 삼성전자 측은 공식 답변 ...

 AI타임스
<https://www.aitimes.com> > news > articleView :

삼성, 엔비디아 HBM 테스트 실패 소식에 반박..."원활하게 ...

2024. 5. 24. — 삼성전자의 고대역폭 메모리(HBM) 칩이 엔비디아의 테스트를 통과하지 못한 것으로 알려졌다. 이 때문에 삼성은 최근 메모리 사업부 사령탑을 교체 ...

 스마트에프엔
<https://www.smartfn.co.kr> > view > sfn202407040011 :

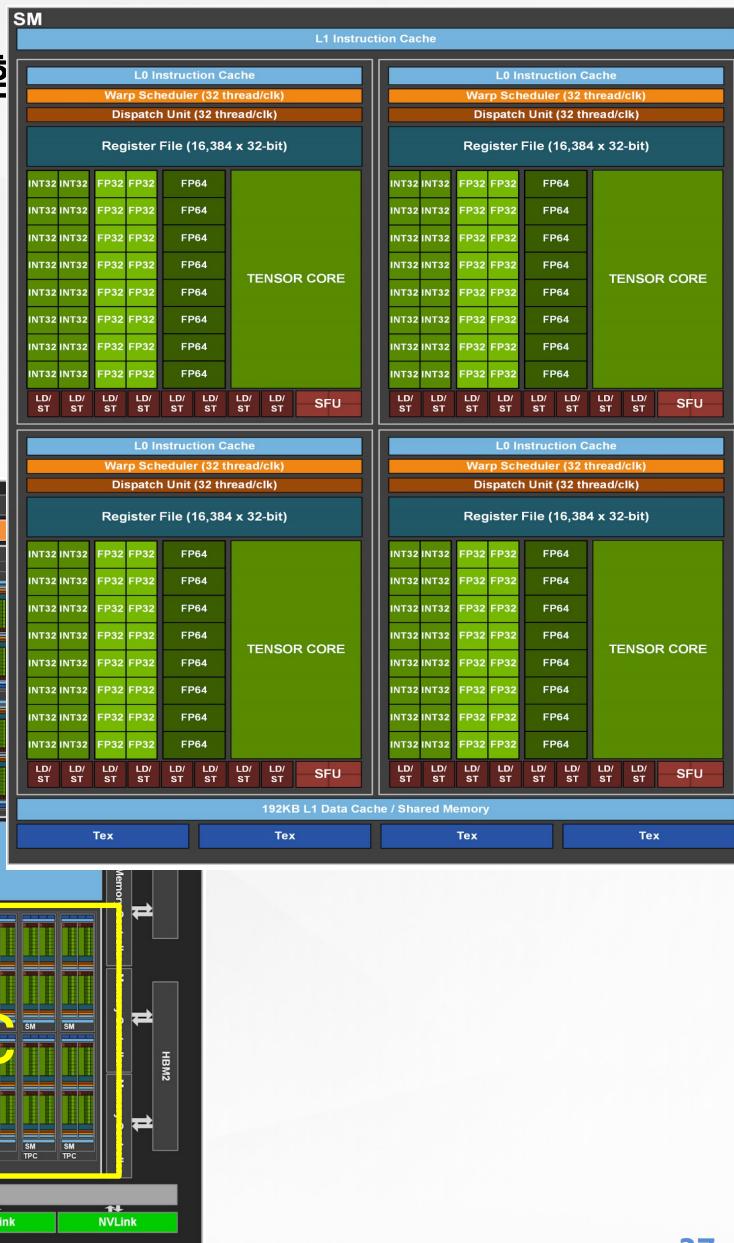
삼성전자, 엔비디아 HBM 테스트 통과?...시장 관심 집중

2024. 7. 4. — 4일 한 매체가 삼성전자가 엔비디아에 5세대 HBM(고대역폭 메모리)인 HBM3E 퀄테스트(품질 검증)에 통과했다는 오보를 냈다. 이에 대해 삼성전자는 ...

GPU 프로세서 물리 구조

– 예) A100 최근 쓰레드 계층 수가 늘었음

- ◆ 108 SM 내부에 64개 CUDA core와 4개 Tensor core 포함
→ GPU 전체 7680 CUDA cores + 480 Tensor cores
- ◆ 54개 TPC (Textile Processing Cluster = 2개 SM 포함)
- ◆ 8개 GPC (Graphic Processing Cluster = 8개 TPC 포함)



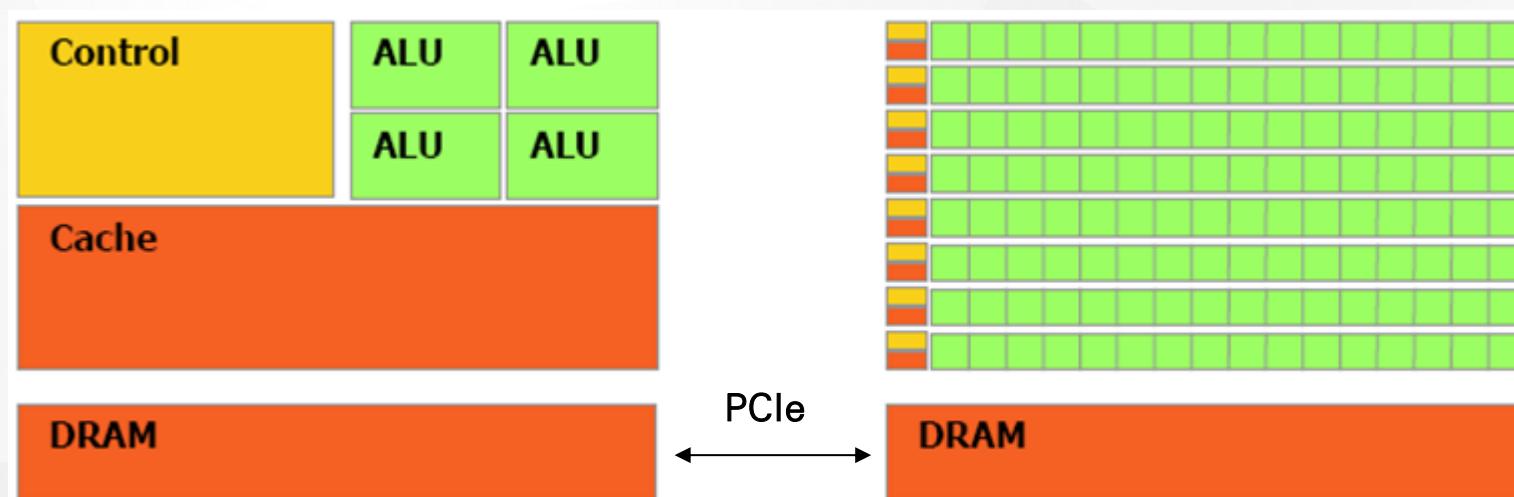
CPU와 GPU의 주요 차이점

◆ CPU

- 지연(latency, 단위: ms) 성능 중심
- 순차 실행에 최적화: 파이프라인, 명령어 병렬(ILP), 분기예측 등
- 단일 복잡 작업에 능함
- 캐시 면적이 큼: 주메모리-ALU 가교 역할
→ DRAM에 접근하는데 수백 사이클(Hz) 소모
- 단일 ALU가 처리능력 높음
- 낮은 연산 지연: 다른 명령어의 동시 처리
- 복잡한 제어 유닛 기능: 본체의 다른 부품과 소통

◆ GPU

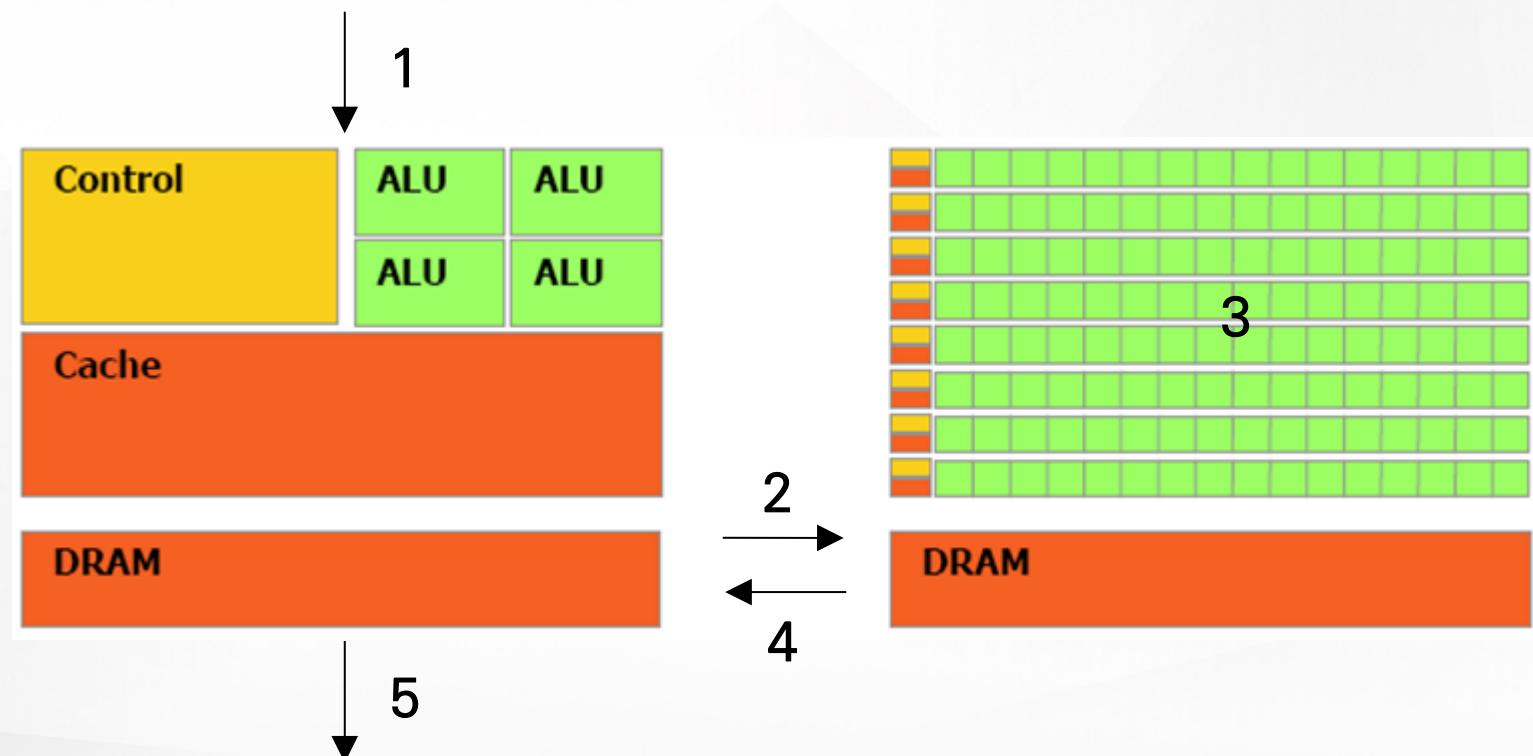
- 처리율(throughput, 단위: MB/s) 성능 중심
- Massively parallel architecture: ALU 개수
- 동시에 많은 동일한 작업에 능함
: Data parallelism, ex) 그래픽 퍽셀, 신경망
- 적은 캐시 메모리
- 그러나 더 빠른 메모리 throughput
- 에너지 효율이 좋은 ALU
- 단순한 제어 유닛 기능: 타 부품과 의존 없음



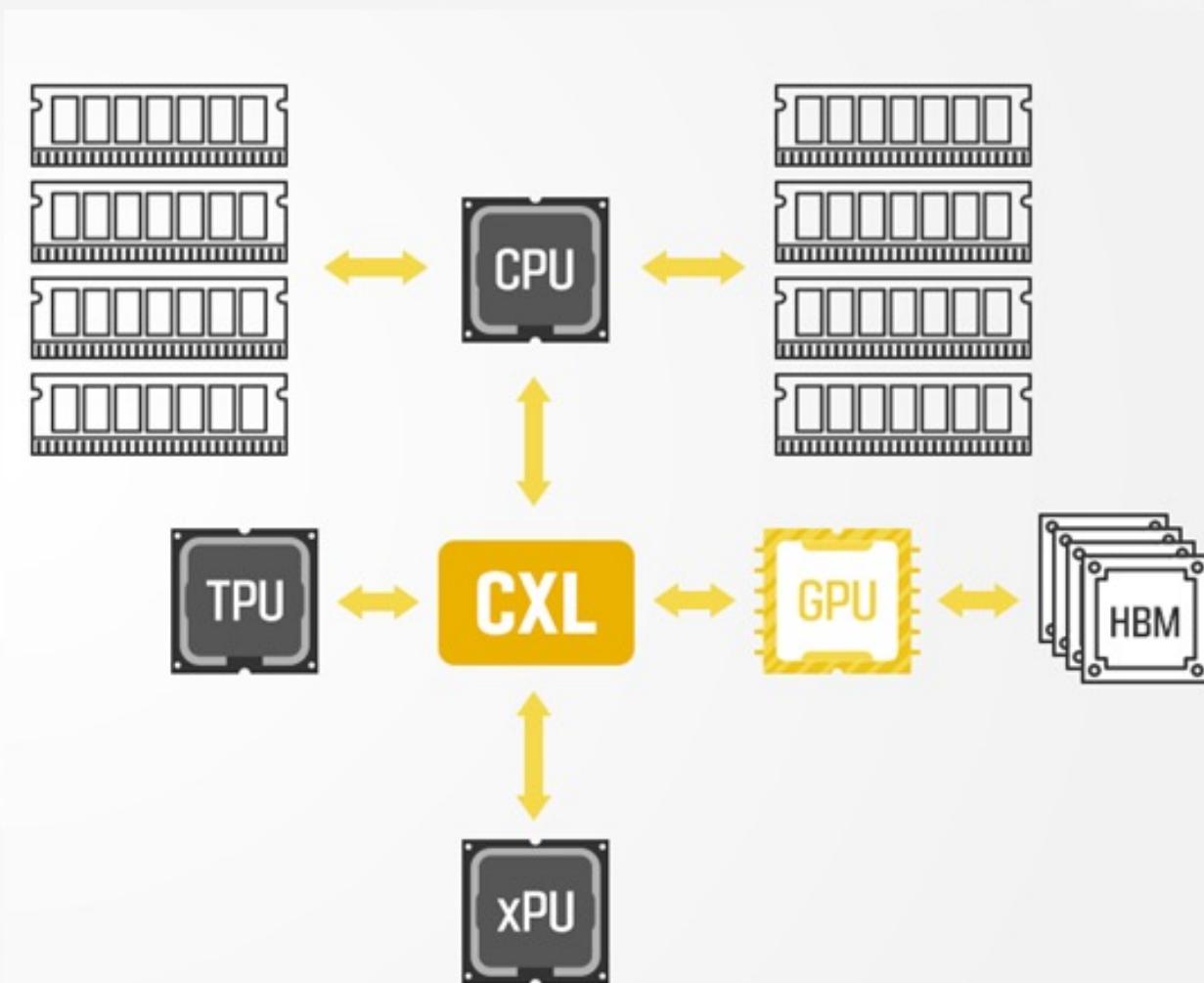
CPU-GPU 연산에서의 Data 흐름

◆ Offloading

- 1 단계) 코드 실행 준비, CPU/GPU 메모리 초기화
- 2 단계) 데이터를 GPU로 전송
- 3 단계) GPU 연산 수행
- 4 단계) CPU 메모리로 역 전송
- 5 단계) 실행 종료 및 CPU/GPU 메모리 삭제



⁴⁰ 이종 컴퓨터(Heterogeneous Computer)로 전환



CXL (Compute eXpress Link)

: 고성능화를 위해 메모리, 로직 반도체 등 장치별로
서로 다른 인터페이스를 하나로 통합해 주는 메모리
기술

SK hynix
NEWSROOM

사례) AI 모델의 성능 요인 분석

◆ Transformer의 Workload를 분석한 결과

- AI 분야는 전반적으로 행렬곱셈(MatMul)이 부하의 상당 부분을 차지
- 동일 모델에 입력의 크기가 클 수록 기타 요인이 커짐
- AI 모델의 크기가 커져서 다중 계산 디바이스로 가면 통신부하가 행렬곱 부하만큼 중요해짐

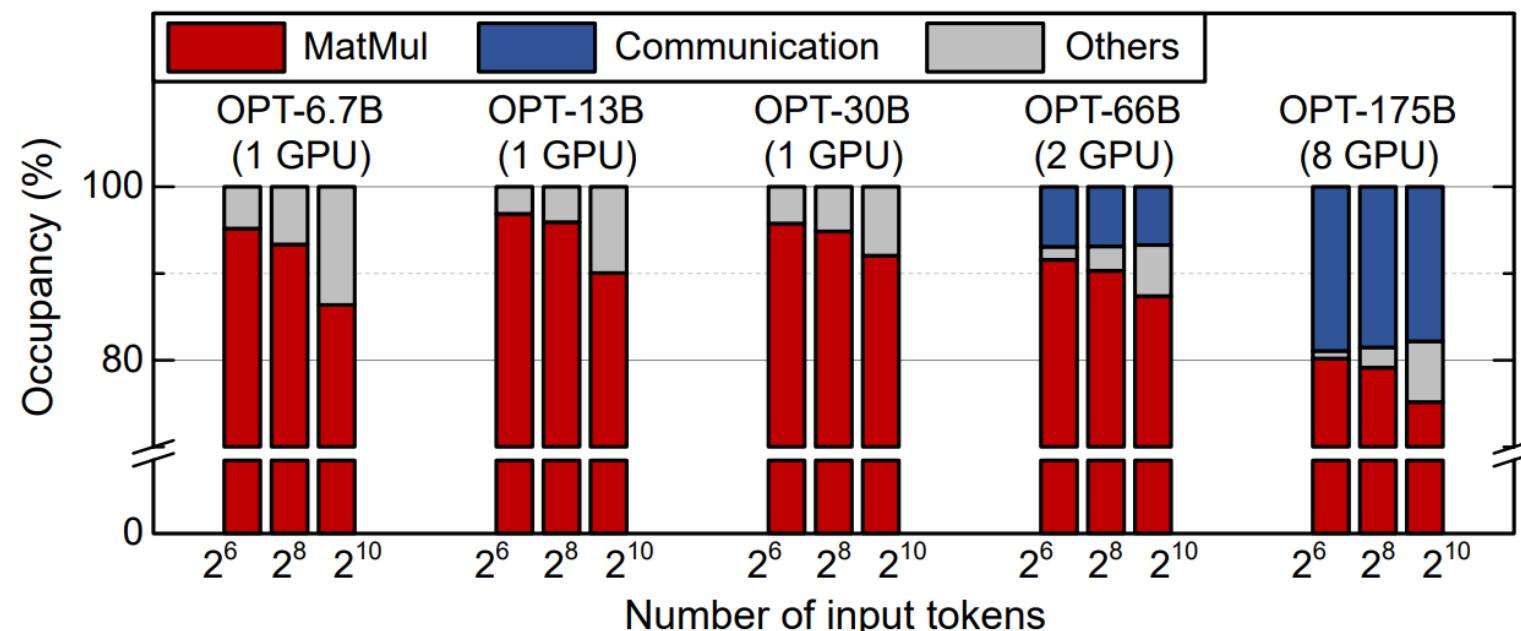


Figure 5: Latency breakdown of various OPT models. The overall performance is dominated by MatMul and communication operations. Experiments are conducted with A100 80GB and FasterTransformer (v4.0) framework.

[참고] 컴퓨팅 성능지표 - 대표적인 것만

프로세서와 메모리 성능을 정량화하기 위해 다양한 성능지표들을 사용

◆ 프로세서 – FLOPS (Floating point Operations Per Second)

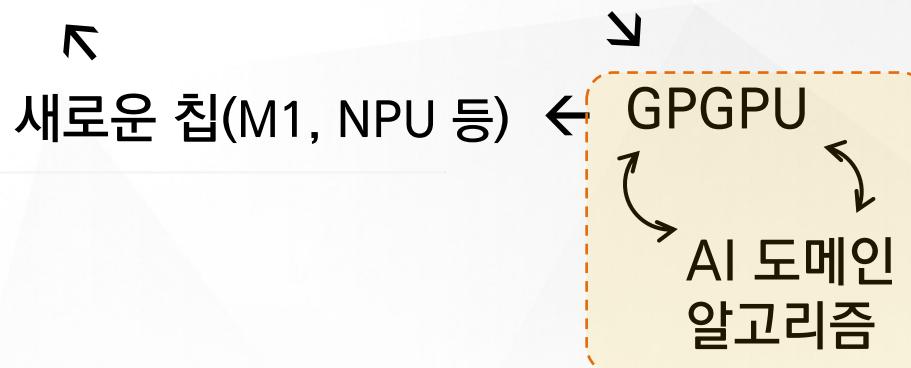
- 초당 수행할 수 있는 부동 소수점(floating-point) 연산 횟수
 - 단일 프로세서에 대한 이론 성능 FLOPS 계산
 - 프로세서 Clock 속도 × 명령처리개수 / cycle × FPU개수 × FPU당 연산 처리방식 변수
- 예) 1.3 GHz X 1 inst./cycle X 2 FPU = 2.6 GFLOPs
(FPU : 부동소수점 연산장치)

◆ 메모리 – 처리량(throughput, 예. Bytes/s), 지연(latency, 예. s/inst.)

- 계산에 필요한 데이터를 연산장치로 공급하는 전송 능력
- CPU ↔ 계층 메모리, 주메모리 ↔ 디스크, GPU ↔ CPU, 시스템 ↔ 시스템

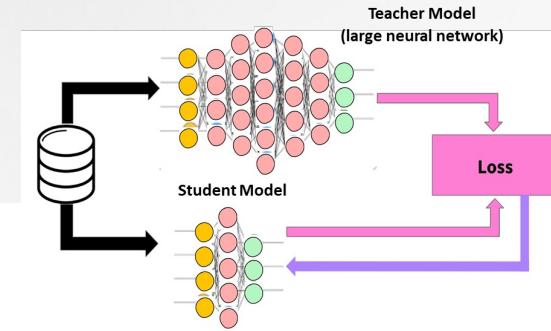
Key Takeaways

- ✓ AI 컴퓨팅 발전 동향: 머신러닝, 딥러닝, 거대언어모델
- ✓ 알고리즘을 통한 컴퓨팅 가속 사례 : '오차 역전파'의 힘
- ✓ 혁신의 전이: HPC → 병렬연산 → 그래픽스 분야



- ✓ 추가 중요한 개념들: CPU-GPU 차이, GPU 아키텍처, 메모리, 통신

딥러닝 최적화 방법



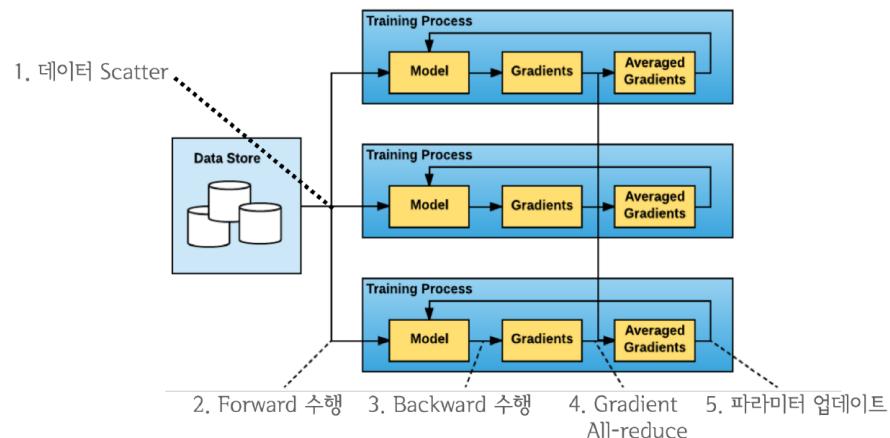
딥러닝 학습/추론 가속화

분산 학습

각종 API 활용

- horovod.tensorflow
- torch.nn.DataParallel

예) DataParallel



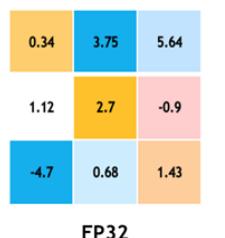
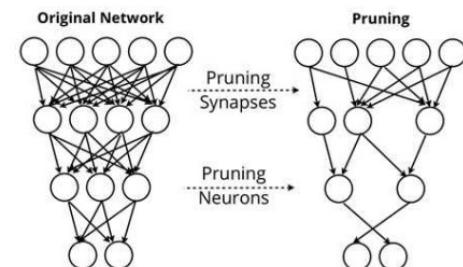
Hot-spot 병렬처리

• 지식증류(Knowledge Distillation)

• 가지치기(Pruning)

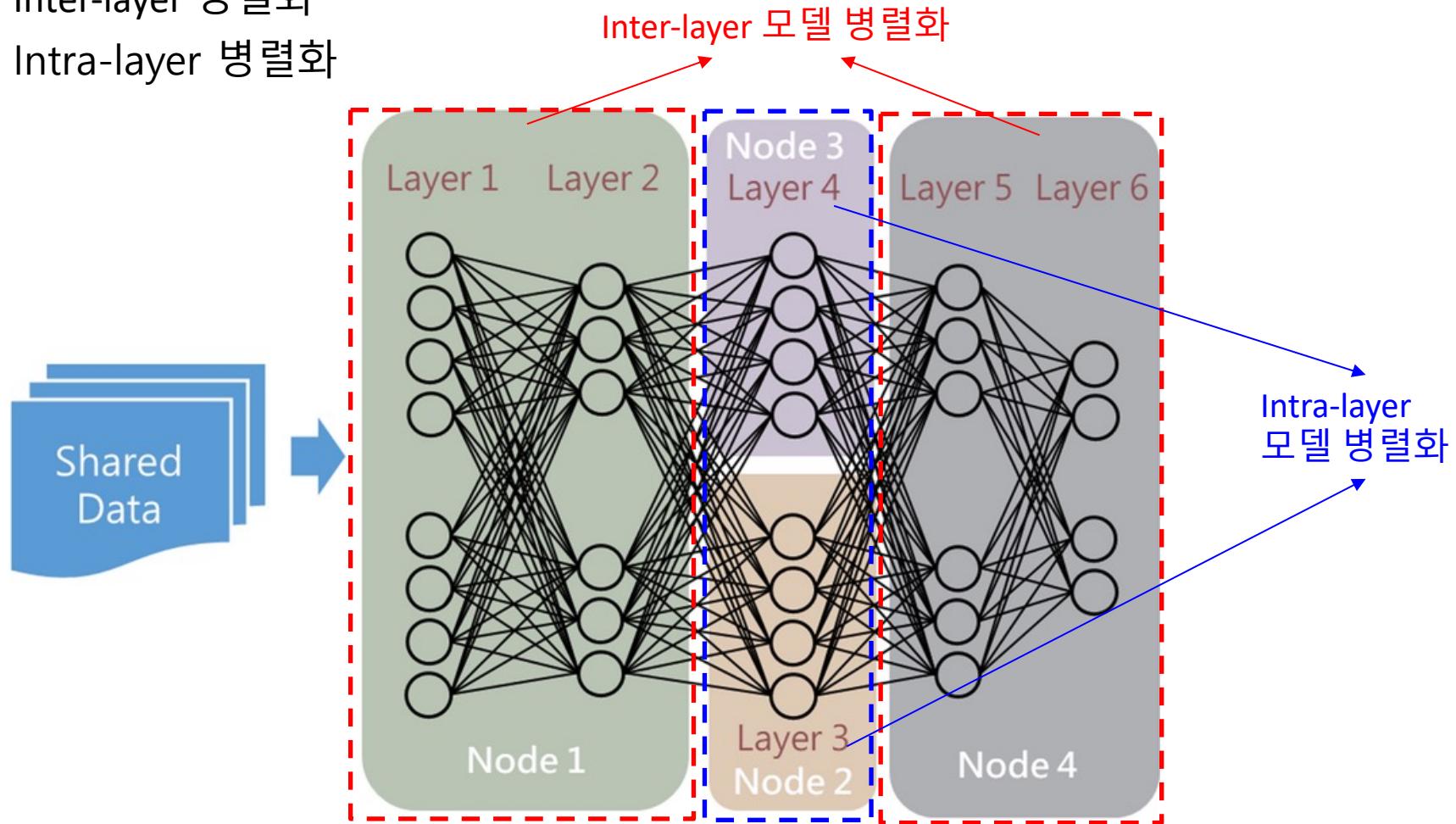
• 양자화(Quantization)

모델 경량화



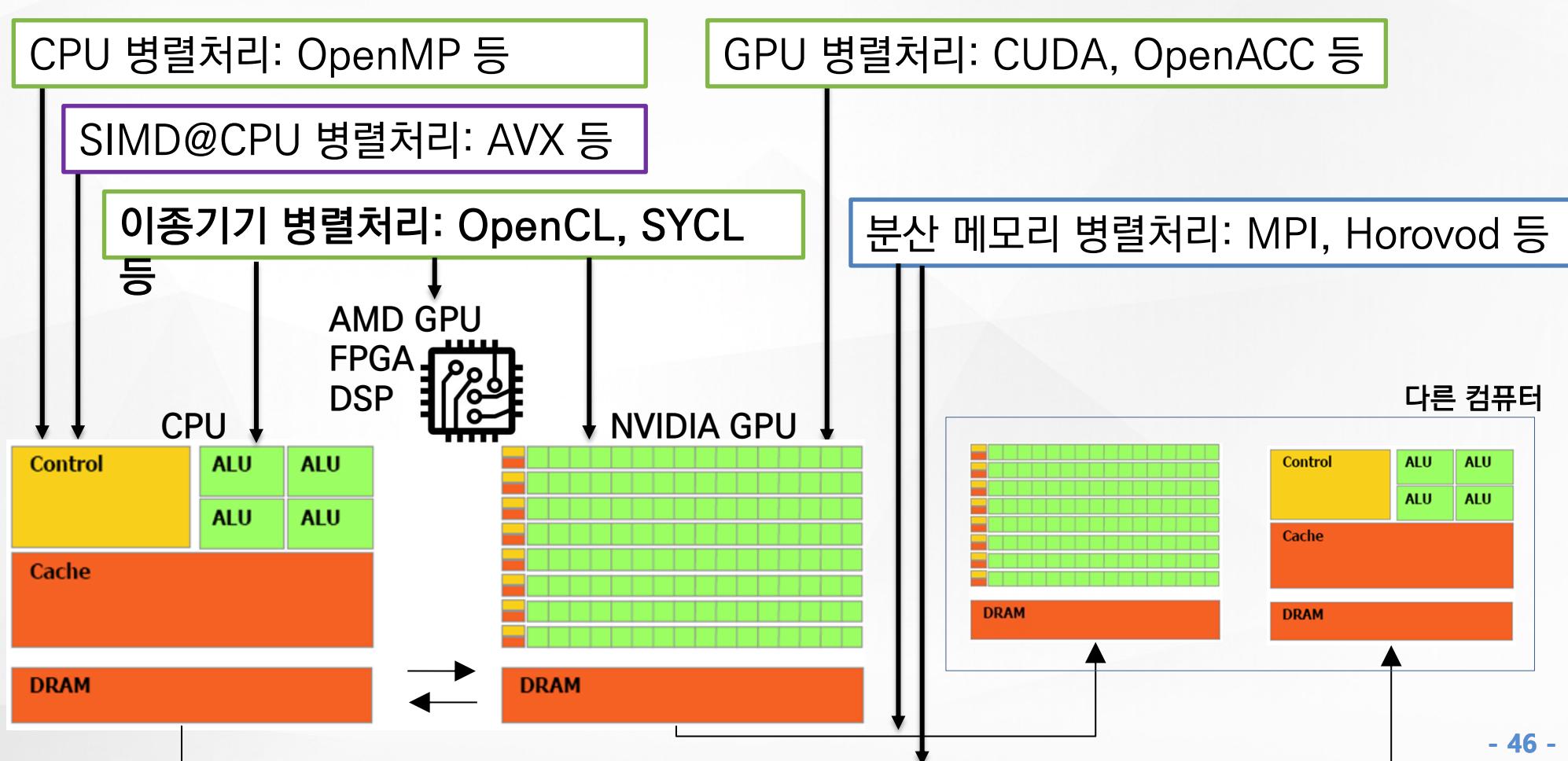
분산 학습 – 병렬 프로그래밍 모델 기반

- Data 분산 샘플링
- Inter-layer 병렬화
- Intra-layer 병렬화

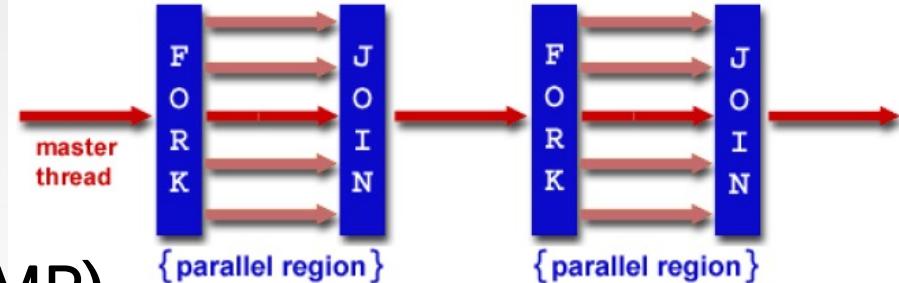


주요 컴퓨팅 가속 프로그래밍 모델

- ◆ 컴퓨터의 각 **컴포넌트**를 사용하여 병렬처리
 - ◆ 공유 메모리 병렬처리 모델
 - ◆ 분산 메모리 병렬처리 모델
- ◆ **하이브리드** 병렬처리 모델



OpenMP



- ◆ Open Multi-Processing (OpenMP)
- ◆ 공유 메모리 병렬처리를 위한 API
- ◆ 병렬코드의 블록만 설명하면 컴파일러가 처리
- ◆ #pragma 명령어를 코드에 추가하여 사용

```
#include <stdio.h>
#include <omp.h>

int main( int argc, char *argv[ ] ) {
    int i;
    #pragma omp parallel for
    for ( i = 0; i<10; i++ ) {
        printf( "[ %d-%d ] Hello World\n", omp_get_thread_num( ), i );
    }
    return 0;
}
```

SSE/AVX/FMA

- ◆ Streaming SIMD Extensions (SSE), Advanced Vector Extensions (AVX), Fused Multiplication Addition (FMA)
- ◆ x86 명령어 집합의 확장 SIMD명령어 집합
- ◆ SIMD 레지스터의 폭이 128비트에서 256비트로 확장
- ◆ AVX-512는 512bit 벡터 연산 가능 함

```
int main() {  
  
    __m256d veca = _mm256_setr_pd(6.0, 6.0, 6.0, 6.0);  
    __m256d vecb = _mm256_setr_pd(2.0, 2.0, 2.0, 2.0);  
    __m256d vecc = _mm256_setr_pd(7.0, 7.0, 7.0, 7.0);  
  
    __m256d result = _mm256_fmaaddsub_pd(vecb, vecc, veca);  
  
    return 0;  
}
```

CUDA

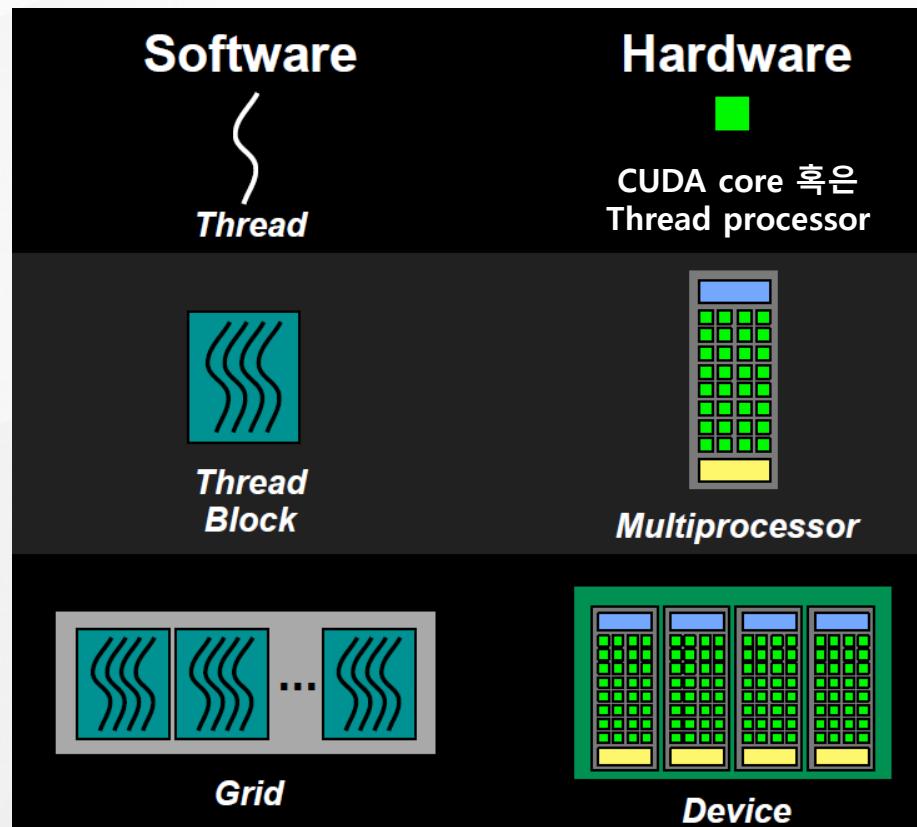
- ◆ NVIDIA GPU를 이용하여 GPGPU 컴퓨팅을 할 수 있게 개발한 기술
- ◆ NVIDIA GPU에서만 사용 가능

```
// CUDA kernel. Each thread takes care of one element of c
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    C[ i ] = A[ i ] + B[ i ];
}

int main()
{
    ...
    // Kernel invocation with blockSize threads
    VecAdd <<< gridSize, blockSize >>>(A, B, C);
    ...
}
```

[참고] GPU(NVIDIA) 프로세서 아키텍처 주요 개념: SM, Grid, Block, Thread, ...

- ◆ Streaming Multi-processor (SM): CPU Core와 동일 수준 단위로 써, 1개의 GPU에는 여러 개의 SM이 있음. Ex) A100의 SM은 108개
- ◆ Kernel: Programmer가 GPU에서 병렬로 실행하게 구현한 함수로 써, 코드를 작성하는 함수 단위



- ◆ **Thread:** 단위 명령을 실행하는 동작 단위로 써, CUDA Core 1개가 1개의 Thread 수행
- ◆ **Block:** 병렬로 독립적으로 수행되는 단위로 써, SM에서 실행되고 여러 개의 Thread와 또한 여러 개의 Warp 단위로 구성
- ◆ **Grid:** 병렬 실행되는 명령의 모음이며, Kernel(=함수) 1개당 1개의 Grid 실행하고 여러 개의 block으로 구성
- ◆ **Warp:** 동시에 동일 명령어로 실행되는 Thread 묶음(32개)으로 써, 스케줄링 되는 최소 단위

32 threads

OpenCL

- ◆ Apple 이 시작하고 AMD, Intel, Nvidia 등과 공동으로 만든 병렬처리 표준 API
- ◆ GPU 뿐만 아니라 CPU, DSP, FPGA 에서도 사용할 수 있도록 설계
- ◆ 크로노스 그룹(Khronos Group)에서 관리하고 있음
- ◆ CUDA와 유사한 프로그래밍 구조

```
__kernel void vecAdd( __global float *A,
                      __global float *B,
                      __global float *C,
                      const unsigned int n)
{
    //Get our global thread ID
    int id = get_global_id(0);

    //Make sure we do not go out of bounds
    if (id < n)
        C[id] = A[id] + B[id];
}
```

SYCL

- ◆ SYCL은 이종 플랫폼 컴퓨팅을 표준 C++로 단일 소스 프로그래밍으로 수행할 수 있음
- ◆ 검증된 OpenCL에서 많은 개념 차용(platform, device, work group, range)

```
#include <sycl/sycl.hpp>

int main(int argc, char* argv[ ]) {
    std::vector<float> a{ 2.3 }, b{ 3.2 }, r{ 7.9 };
    try {
        auto defaultQueue = sycl::queue{};

        auto bufA = sycl::buffer{ a, sycl::range{dataSize} };
        auto bufB = sycl::buffer{ b, sycl::range{dataSize} };
        auto bufR = sycl::buffer{ r, sycl::range{dataSize} };

        defaultQueue
            .submit([&](sycl::handler& cgh) {
                sycl::accessor accA{ bufA, cgh, sycl::read_only };
                sycl::accessor accB{ bufB, cgh, sycl::read_only };
                sycl::accessor accR{ bufR, cgh, sycl::write_only };

                cgh.parallel_for<vector_add>(
                    sycl::range{ dataSize },
                    [=](sycl::id<1> idx) { accR[idx] = accA[idx] + accB[idx]; });
            })
            .wait();

        defaultQueue.throw_asynchronous();
    }
}
```

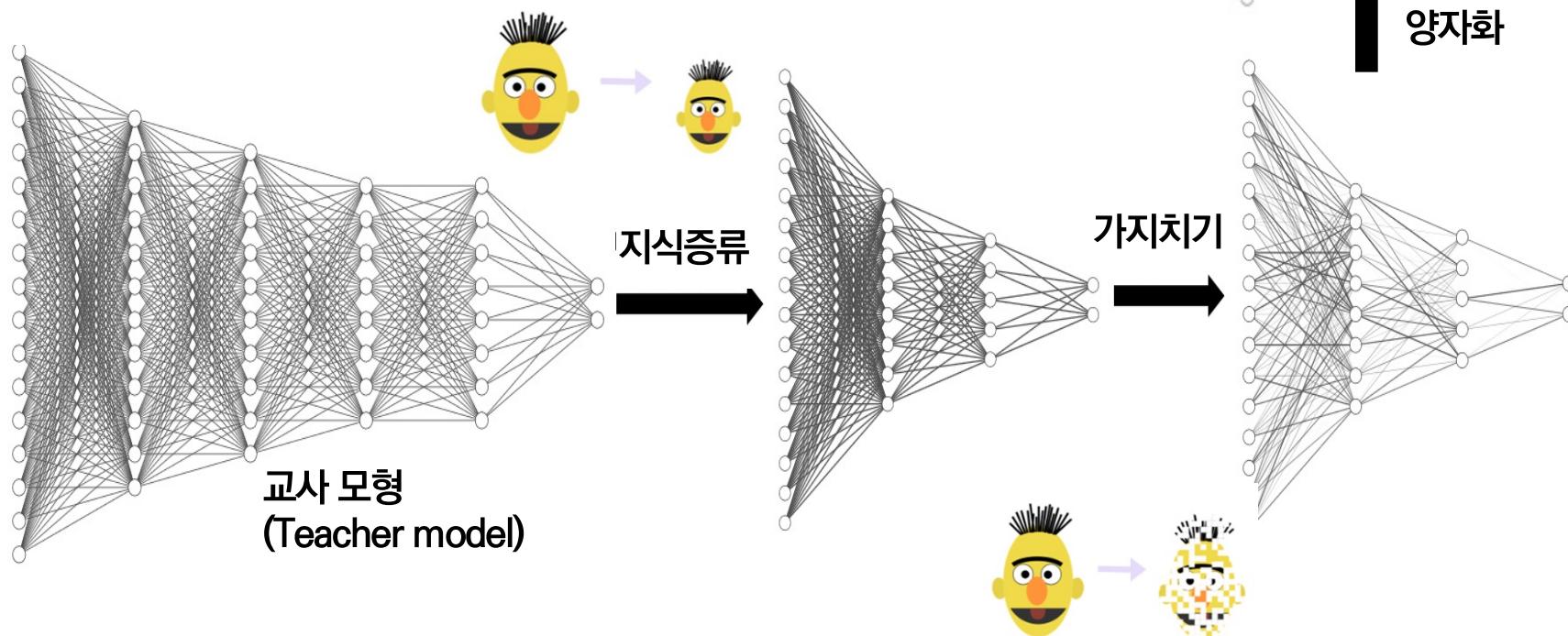
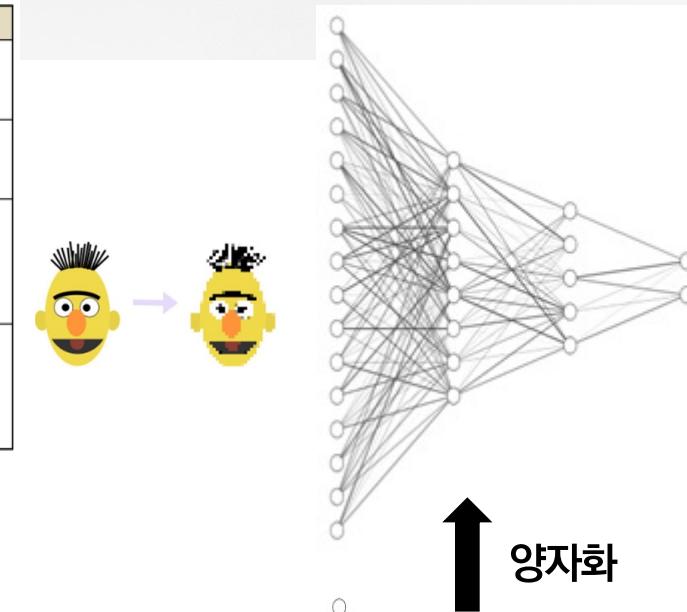
데이터 관리
buffer

작업 단위
Work unit

디바이스 코드
Device code
(Kernel)

모델 경량화 방법론

기술 분류	기술 요약	접근 방식
가지치기 (Pruning)	기준 값 이하의 가중치나 뉴런을 제거하며 반복 학습	기 학습된 모델을 이용
양자화 (Quantization)	가중치, 활성값, 기울기(구배)값을 보다 낮은 비트너비로 표현되는 값으로 대체	기 학습된 모델을 이용하거나, 모델 학습 때 적용
지식 증류 (Knowledge Distillation)	기 학습된 모델을 teacher 모델로 하여 경량화된 student 모델을 학습함	기 학습된 모델을 이용
경량 네트워크 설계 (Compact Network Design)	연산 효율성을 향상을 위한 뉴럴 네트워크 구조를 변경하는 방식	모델 학습 때부터 적용



Key Takeaways

- ✓ 딥러닝 최적화 방법: 분산 훈련, Hot-spot 병렬처리, 모델 경량화
- ✓ 분산 학습: Inter-layer, Intra-layer 등 (torch, tensorflow 프레임워크)
- ✓ 주요 병렬 프로그래밍 모델들: OpenMP, SIMD-AVX, CUDA, SYCL 등
- ✓ 모델 경량화: 가지치기, 지식증류, 양자화



저녁식사 맛있게 하세요.
shandy77@kisti.re.kr