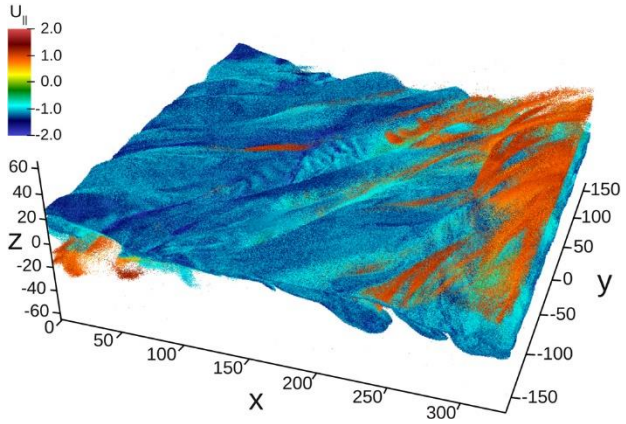
Lecture. 1

왜 병렬 컴퓨팅 인가?

Why parallel computing?

병렬 컴퓨팅(Parallel computing)란?

- 하나의 문제를 여러 개의 컴퓨팅 자원을 활용해서 해결하는 것
  - 여러 개의 코어 또는 연산 노드 등



- Parallel computing
  vs Concurrent computing

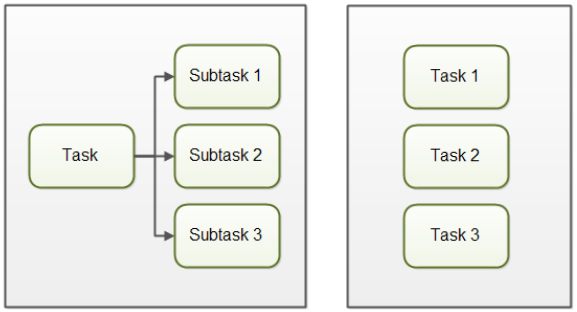


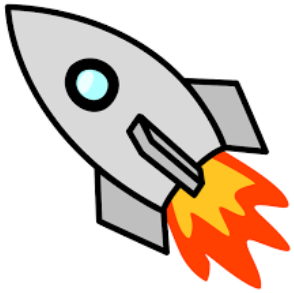
Image from http://tutorials.jenkov.com/java-concurrency/concurrency-vs-parallelism.html

왜 병렬처리 인가?

- 사용자**(application)**의 요구

- 아키텍처**(architecture)** 발전 경향

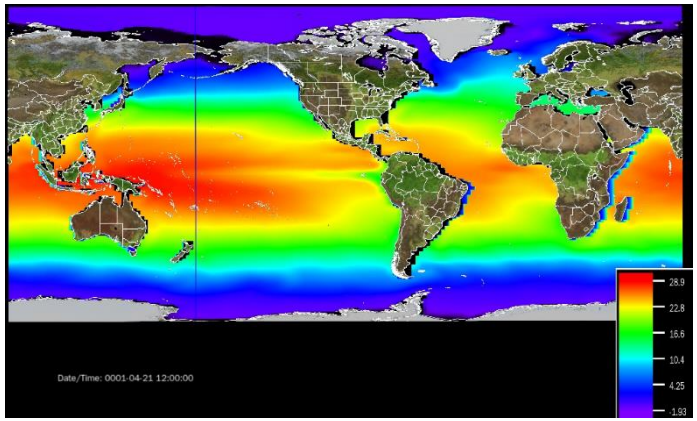
사용자의 요구

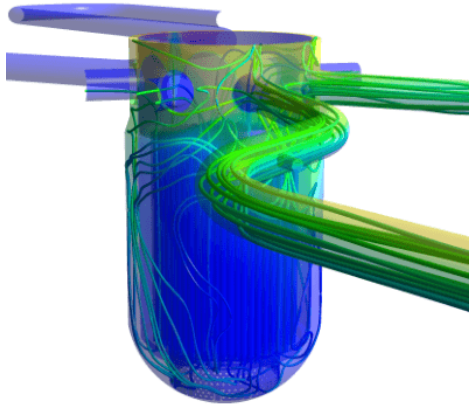


빠르고

정확하게

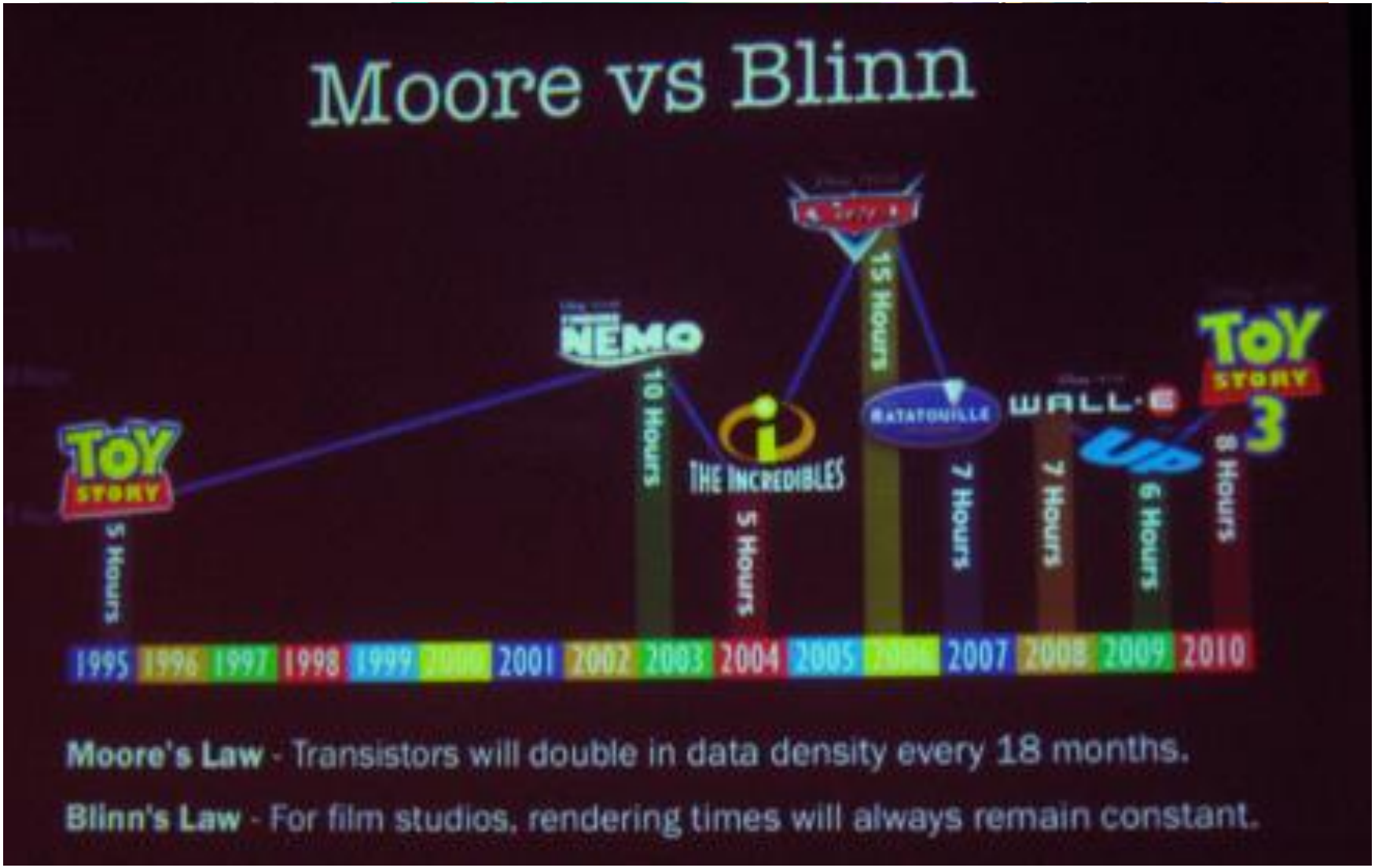
사용자의 요구





- 과거
  - 특정 분야에 한정적으로 고성능 컴퓨팅이 요구 됨
  - 주로 CPU clock frequency를 높이는 방법으로 성능 달성
- 현재
  - 고성능 컴퓨팅을 요구하는 모든 분야에서 필요

사용자의 요구


Moore vs Blinn
TOY STORY
5 Hours
NEMO
10 Hours
THE INCREDIBLES
5 Hours
15 Hours
RATATOUILLE
7 Hours
WALL·E
7 Hours
UP
6 Hours
TOY STORY 3
8 Hours
1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010
Moore's Law - Transistors will double in data density every 18 months.
Blinn's Law - For film studios, rendering times will always remain constant.

사용자의 요구

- 병렬 컴퓨팅의 사용 목적

$$Speedup = \frac{Performance\ (p\ processors)}{Performance\ (1\ processors)}$$

왜 병렬처리 인가?

- 사용자(application)의 요구

- **아키텍처(architecture) 발전 경향**

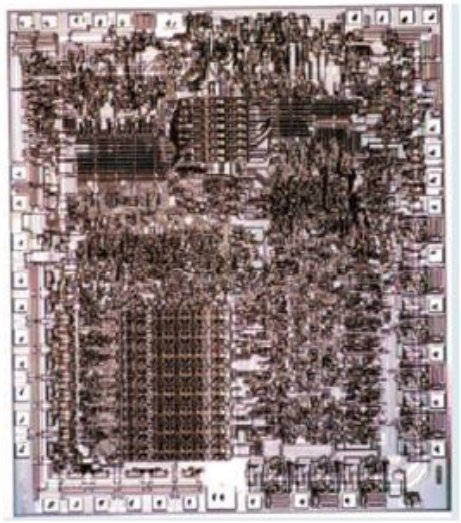
아키텍처 발전 경향

Image from: https://www.minecraftforum.net/forums/minecraft-java-edition/redstone-discussion-and/340404-my-alu-cpu-computer-progress-thread-video-and

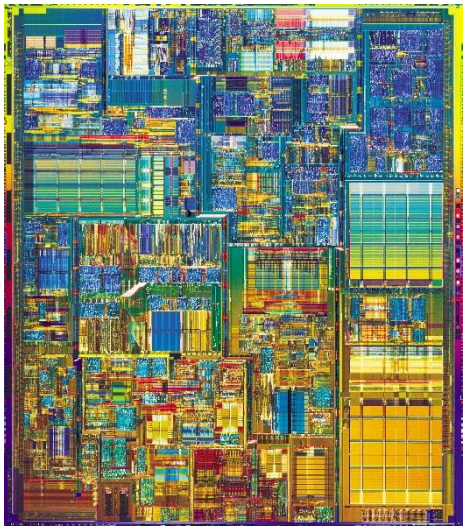
아키텍처 발전 경향

- 고집적

= 고성능
= 고전력
= 고발열
= 연산오류

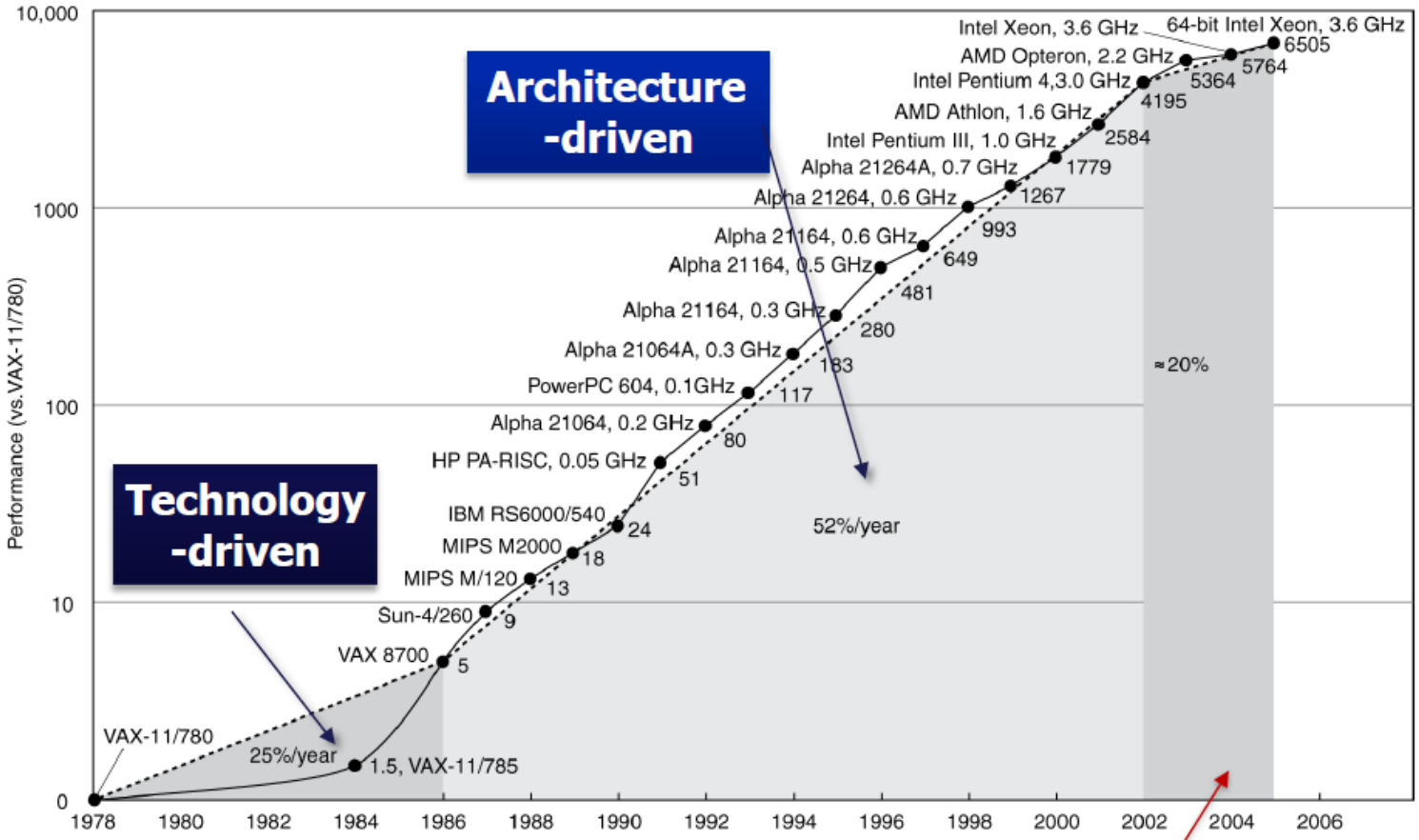


Intel 8080, 1978
29K transistors



Intel Pentium 4, 1999
28M transistors

아키텍처 발전 경향

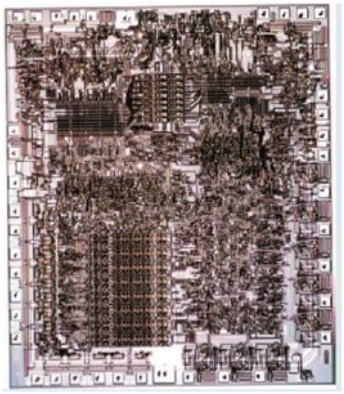


The rate of single-thread performance improvement has decreased

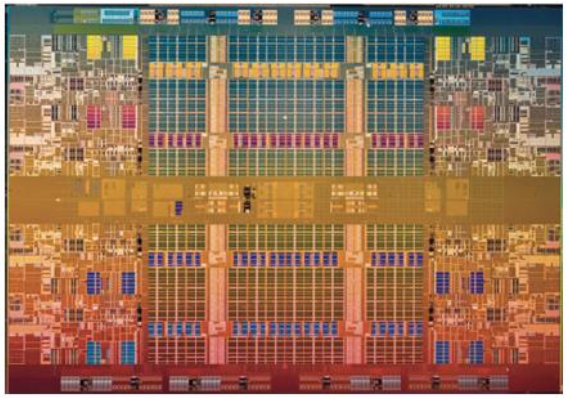
[Computer Architectures – Hennessy & Patterson]

아키텍처 발전 경향

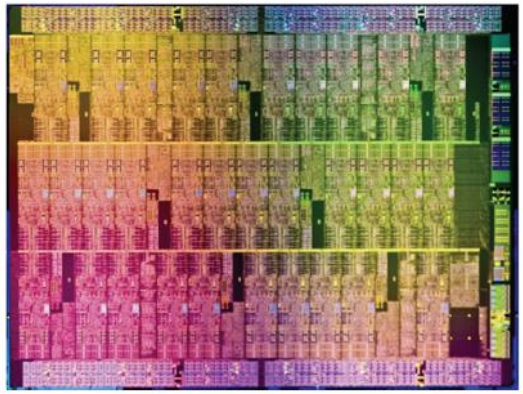
- 하나의 빠른 마이크로 프로세스 대신
- 여러 개의 프로세스를 하나의 칩에 담기 시작



Intel 8088, 1978
1 core, no cache
29K transistors



Intel Nehalem-EX, 2009
8 cores, 24MB cache
2.3B transistors

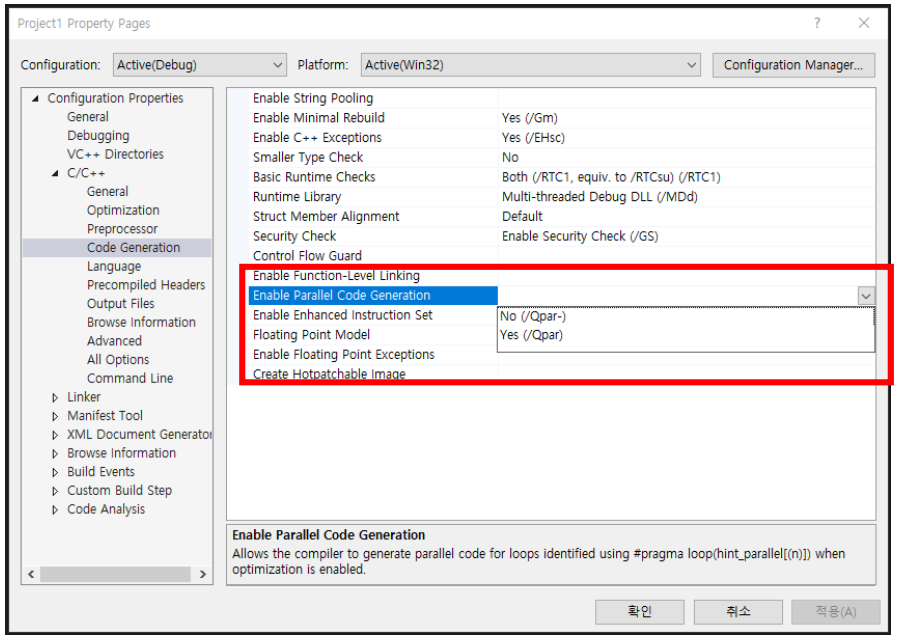


Intel knight landing, 2016(?)
72 cores, 16GB DDR3 LLC(?)
7.1B transistors

병렬 프로그래밍이 필요한 이유

- 고성능에 대한 사용자의 요구를
  하드웨어 발전 경향에 맞추어
  병렬처리 장치를 사용하여 해결하기 위해

- 자동 병렬화?
  - 활용 가능 코드가 제한적
  - 멀티 코어의 효율적
    활용이 보장되지 않음



Example – Serial algorithm

- **n**개의 값을 계산하고, 그 합을 구하는 프로그램

```
int sum = 0;
for (int i = 0; i < n; i++)
{
        int x = ComputeNextValue();
        sum += x;
}
```

Example – Naïve parallel algorithm

- **p**개의 코어, **p**는 **n**보다 매우 작음
- 각각의 코어가 **n/p**개의 값을 계산 **(**병렬처리**)**

```
int ComputeMySum(int tid){
    int my_sum[tid] = 0;
    for (int i = my_first ; i < my_end ; i++)
    {
              int my_x = ComputeNextValue();
              my_sum[tid] += my_x;
    }
}
```

Example – Naïve parallel algorithm

- 모든 코어의 연산이 끝난 후
- 마스터 코어(스레드)가 결과를 취합

```
ComputeMySum(tid);
if (tid == 1 )      // master thread
{
        sum = my_x;
        for (int i = 1; i < p ; i++)
        {
                sum += receive(i);
        }
} else {      // slave threads
        sendMysum();
}
```

Example – Naïve parallel algorithm

- 예, **|core| = 8, n = 24**
  - 1,4,3, 9,2,8, 5,1,1, 5,2,7, 2,5,0, 4,1,8, 6,5,1, 2,3,9

Core	0	1	2	3	4	5	6	7
my_sum	8	19	7	15	7	13	12	14

Global sum

8 + 19 + 7 + 15 + 7 + 13 + 12 + 14 = 95

Core	0	1	2	3	4	5	6	7
my_sum	95	19	7	15	7	13	12	14

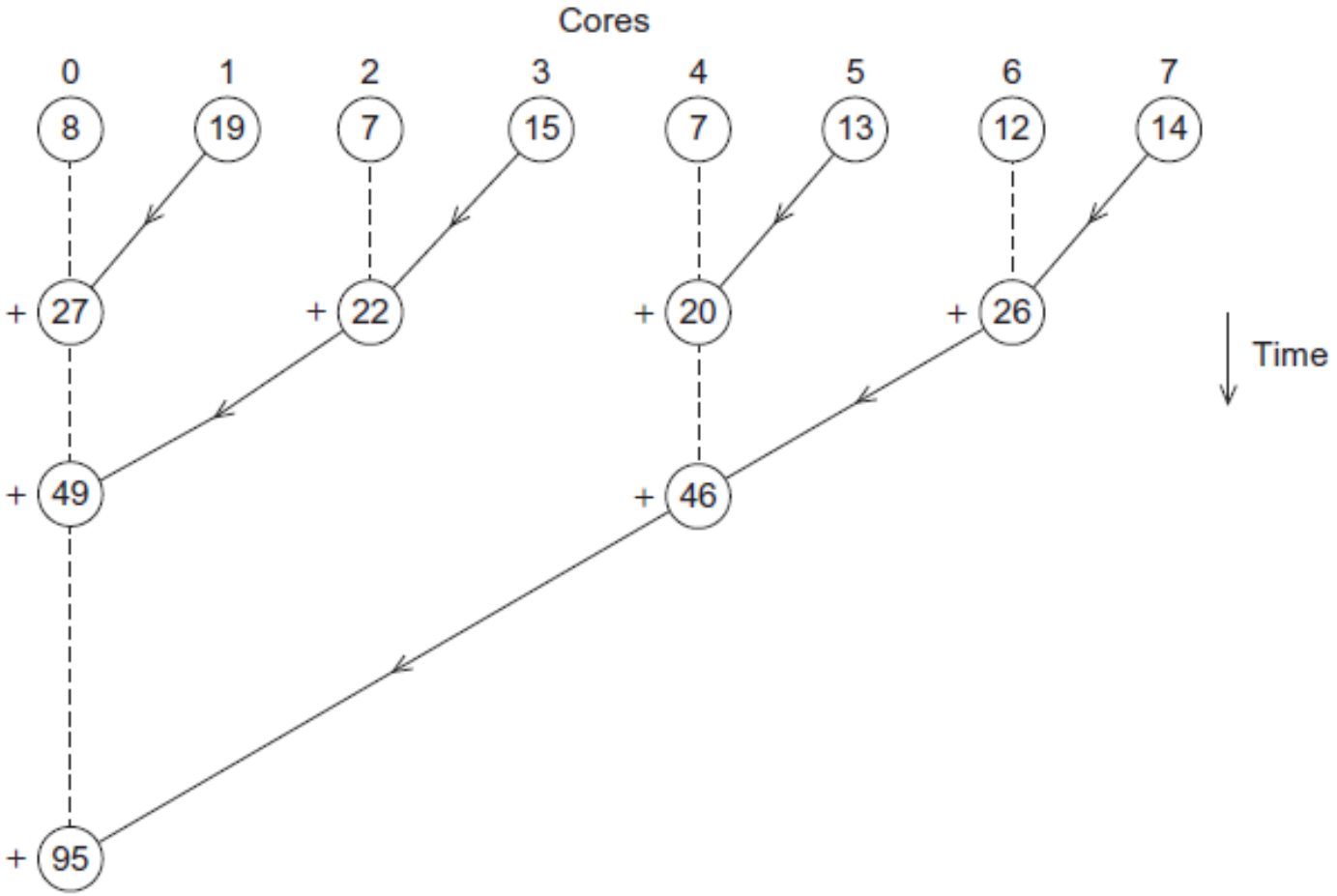
Example – Naïve parallel algorithm

- 효율적인 병렬처리 알고리즘인가?

```
ComputeMySum(tid); // 3
if (tid == 1 )      // master thread
{
        sum = my_x;
        for (int i = 1; i < p ; i++)
        {
                sum += receive(i); // 7
        }
} else {      // slave threads
        sendMysum();
}

// Total computing time = ?
```

Example – Improved Algorithm




```
// Total computing time = ?
```

Performance Analysis

- 3072개의 값을 계산하고 전체 합을 구할 때
- Naïve algorithm

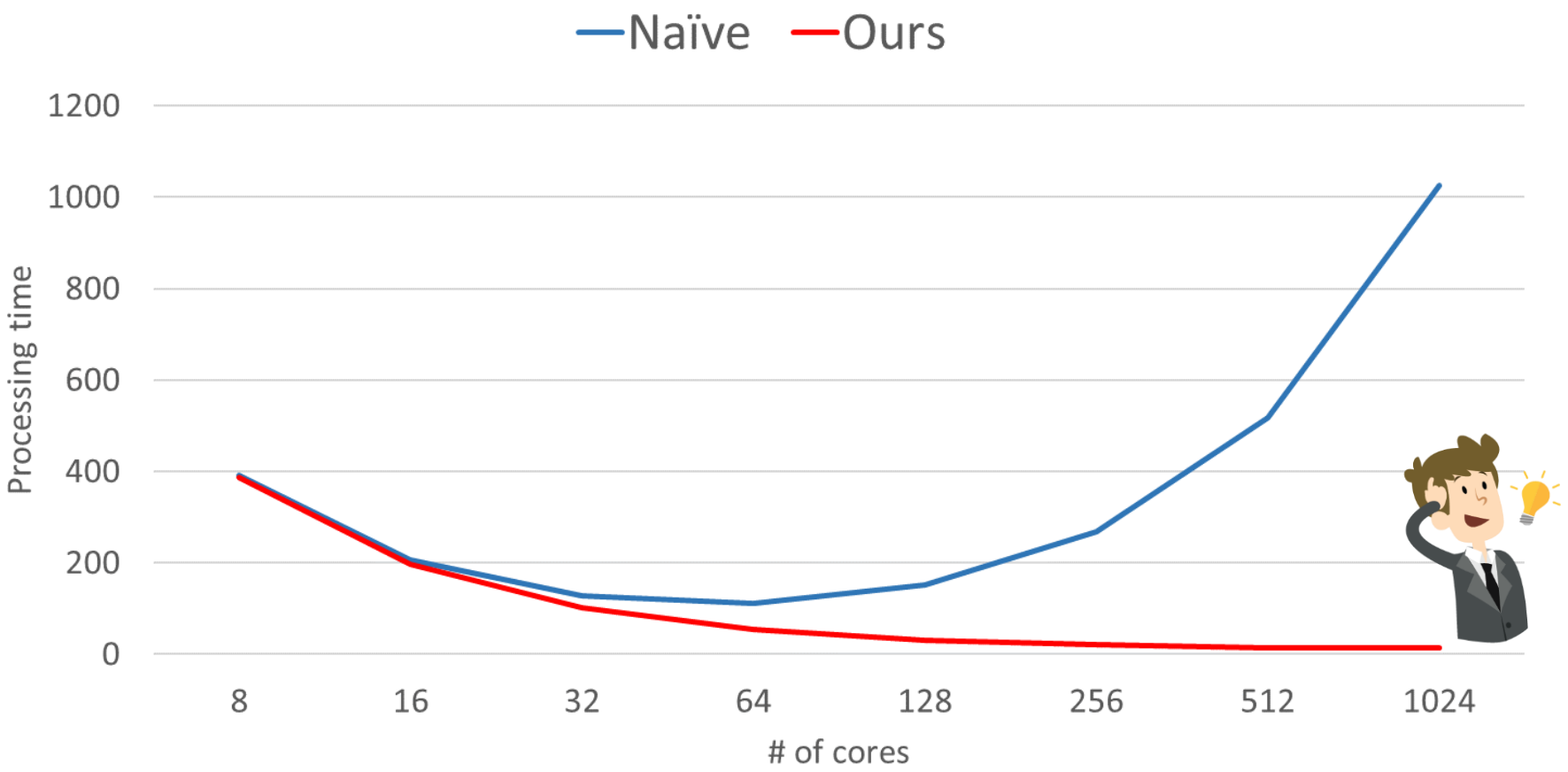
# of cores	8	16	32	64	128	256	512	1024
값 계산 시간	384	192	96	48	24	12	6	3
합 계산 시간	7	15	31	63	127	255	511	1023
총 시간	391	207	127	111	151	267	517	1026

- Improved algorithm

# of cores	8	16	32	64	128	256	512	1024
값 계산 시간	384	192	96	48	24	12	6	3
합 계산 시간	3	4	5	6	7	8	9	10
총 시간	387	196	101	54	31	20	15	13

Performance Analysis

- 3072개의 값을 계산하고 전체 합을 구할 때



요약

- 병렬 컴퓨팅이란?

- 왜 병렬 처리가 필요한가?
  - 사용자의 요구
  - 아키텍처의 발전 경향

- 병렬 프로그래밍이 필요한 이유?