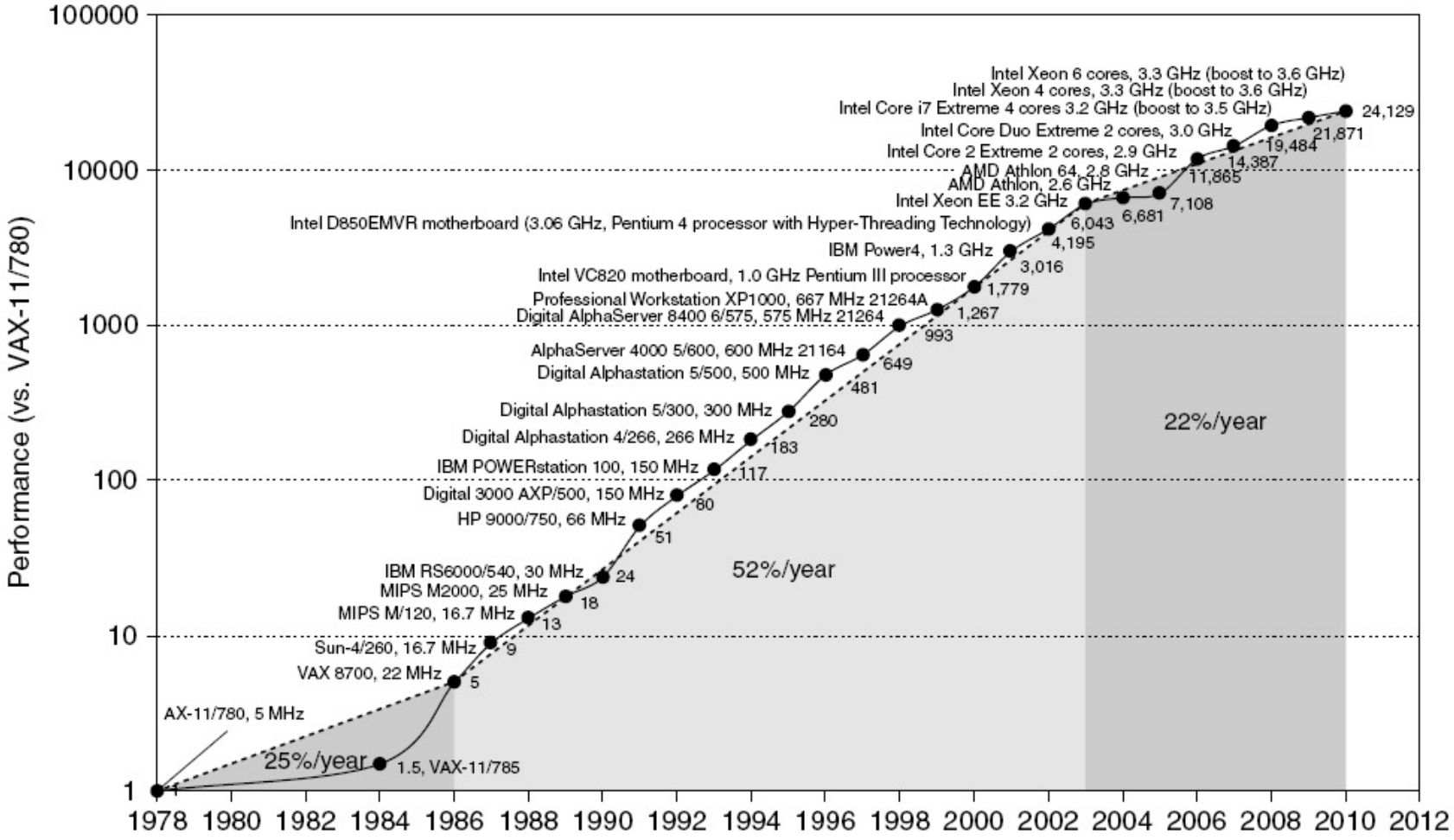
# Introduction to Parallel Computing
# ***Why Parallelism?***

Prof. Seokin Hong

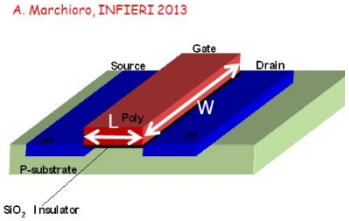
# Incredible progress in computer technology



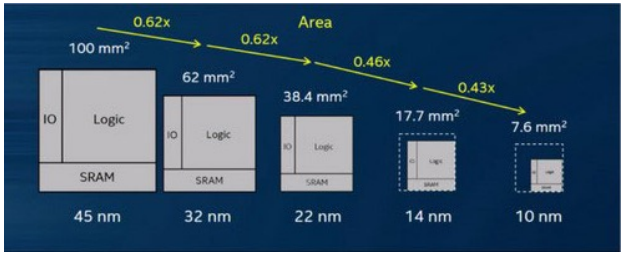
# Incredible progress in computer technology (Cont'd)

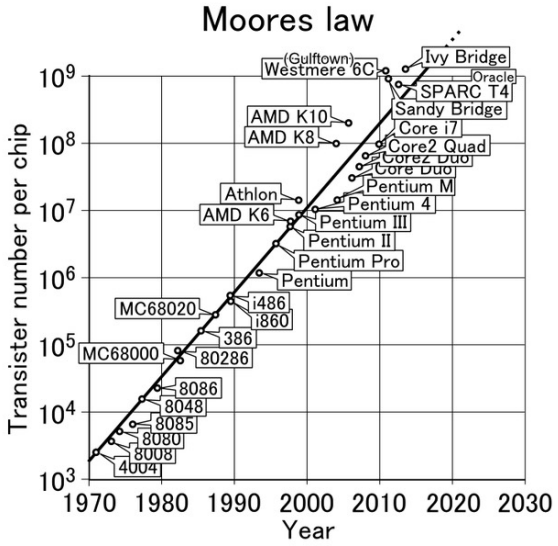
- Performance improvements are led by
  - **Technology Scaling**
    - **Feature size reduction** in CMOS transistor technology
      - Smaller transistors → More transistors
      - Fast transistors → More performance (Higher clock rate)
      - Consume less power → Low power

**Transistor**




W, L = gate width and length



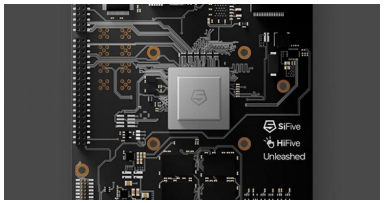

**Moore's Law:** the number of transistors in a dense integrated circuit doubles about every two years, 1965

# Incredible progress in computer technology (Cont'd)

- Performance improvements are led by
  - **Improvements in computer architectures**
    - Enabled by
      - **Advanced Compiler** → Elimination of assembly language programming
      - **Standardized and vendor-independent operating systems** (e.g., LINUX)
    - These two changes lowered the cost of bringing out a new architecture
    - Lead to innovative CPU architectures

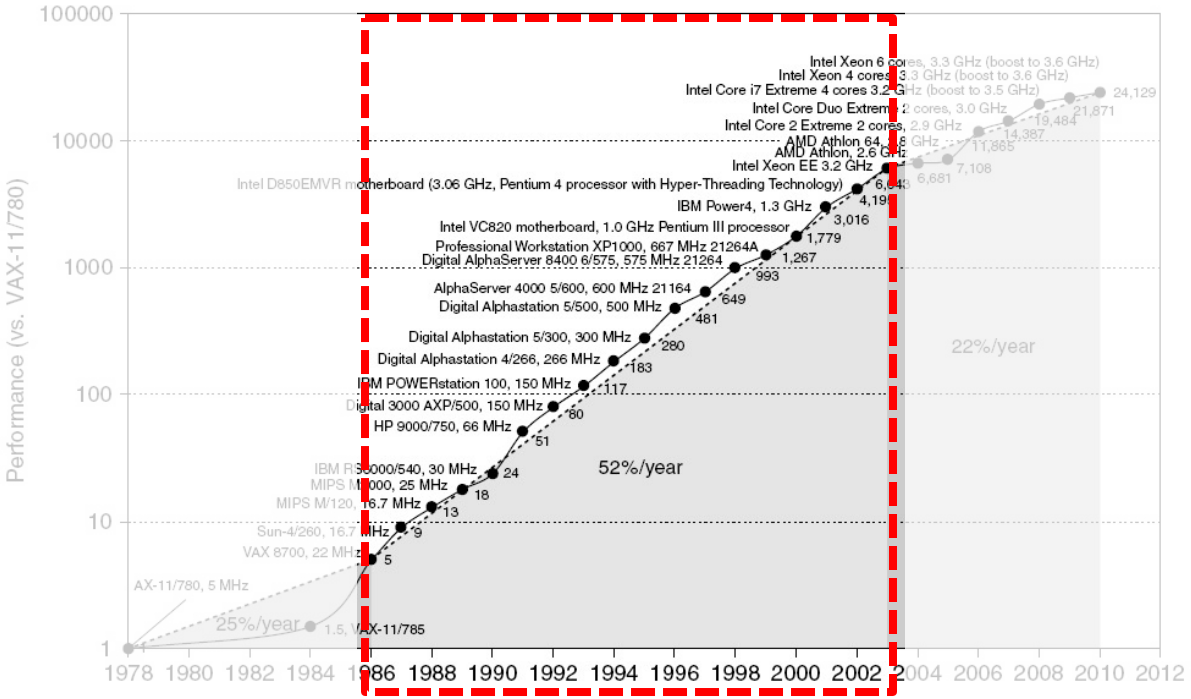






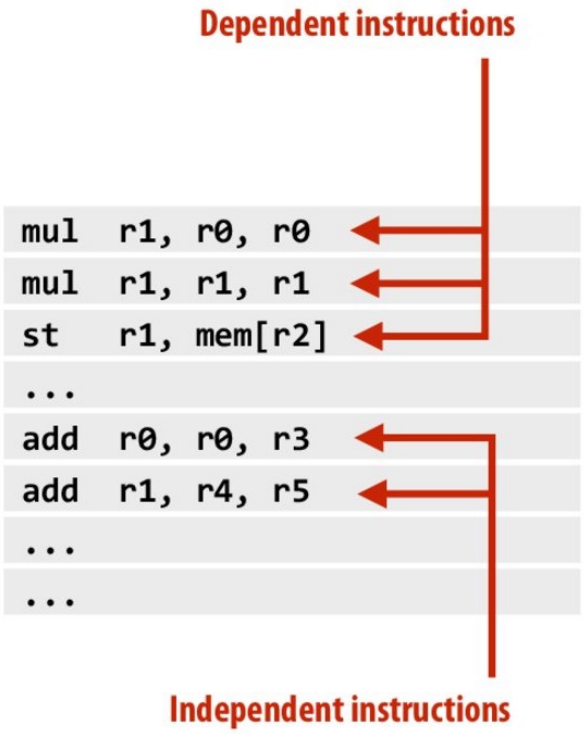
# Why wasn't parallel processing required?

- Single-threaded CPU performance doubling every 18 months
  - Since H/W performance increased, S/W performance *automatically* increased without any change!!
  - Working to parallelize program code was often not worth the time



# Two driving forces of performance improvement until 2003, and their limitation

1. **Exploiting instruction-level parallelism (ILP)**
   - Execute independent instructions simultaneously
2. **Increasing clock frequency**
   - Technology scaling → fast transistor → higher clock frequency



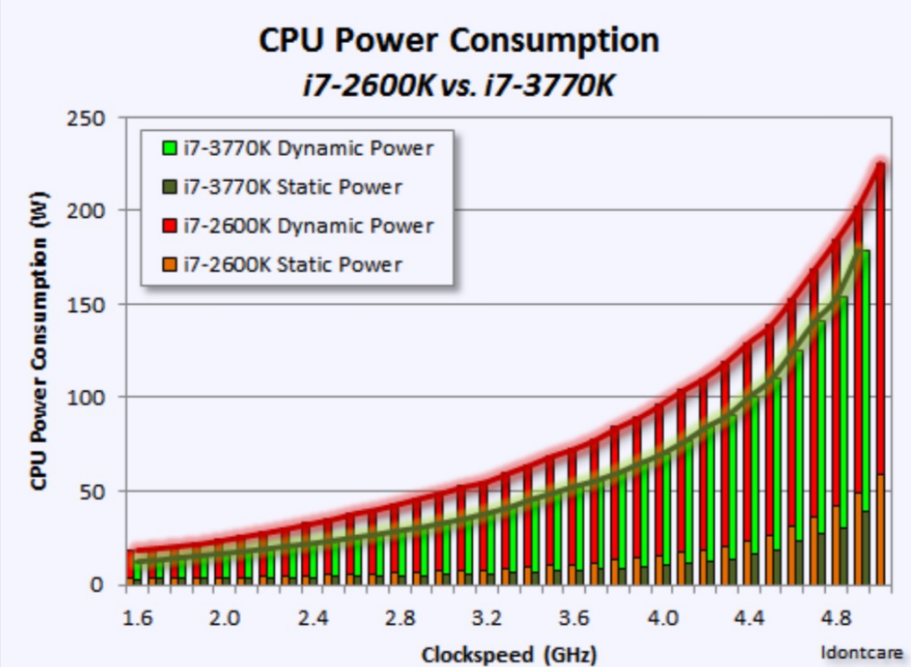

- **Single processor performance improvement ended in 2003**
  - Cannot continue to leverage Instruction-Level parallelism (ILP)
  - Cannot increase the clock frequency further due to power

# Two driving forces of performance improvement until 2003, and their limitation

- The "Power wall"
  - Power consumption is proportional to frequency

$$\text{Power} = \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$

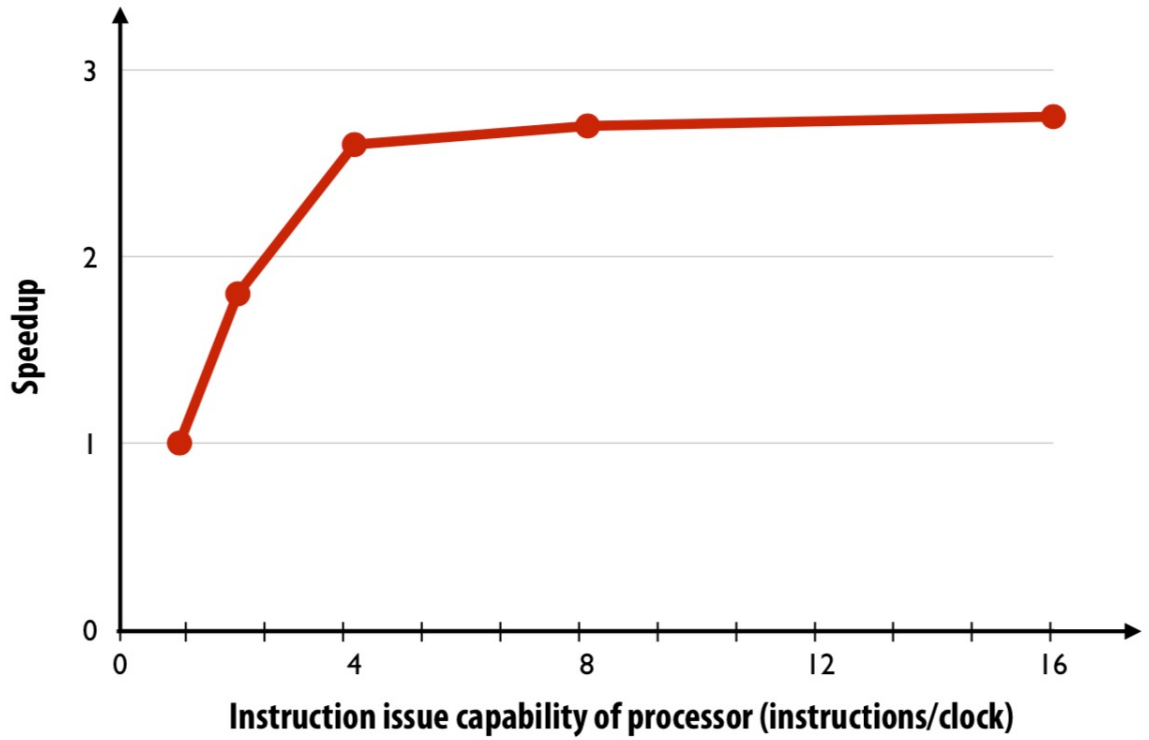
  - High power consumption ➔ high temperature




"Idontcare": posted at: http://forums.anandtech.com/showthread.php?t=2281195

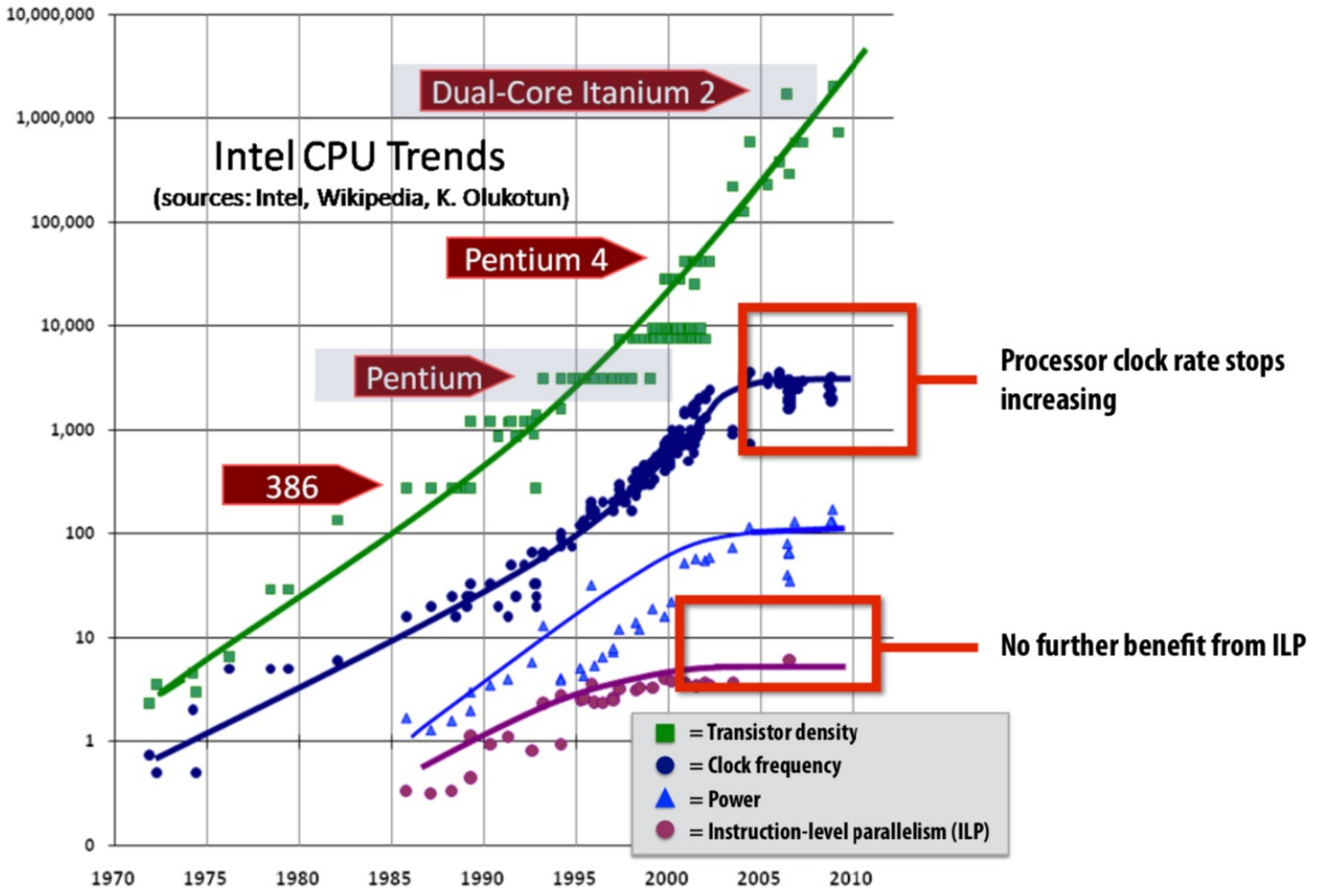
# Two driving forces of performance improvement until 2003, and their limitation

- Diminishing gain with ILP
  - Little performance benefit from building a processor that can issue more




Culler & Singh (data from Johnson 1991)

# Two driving forces of performance improvement until 2003, and **their limitation**




"The free Lunch is Over" by Herb Sutter, Dr. Dobbs 2005

# Why is parallel processing required?

- **Parallel processing is the primary way for continuous performance improvement of processor**

Intel's Big Shift After Hitting Technical Wall

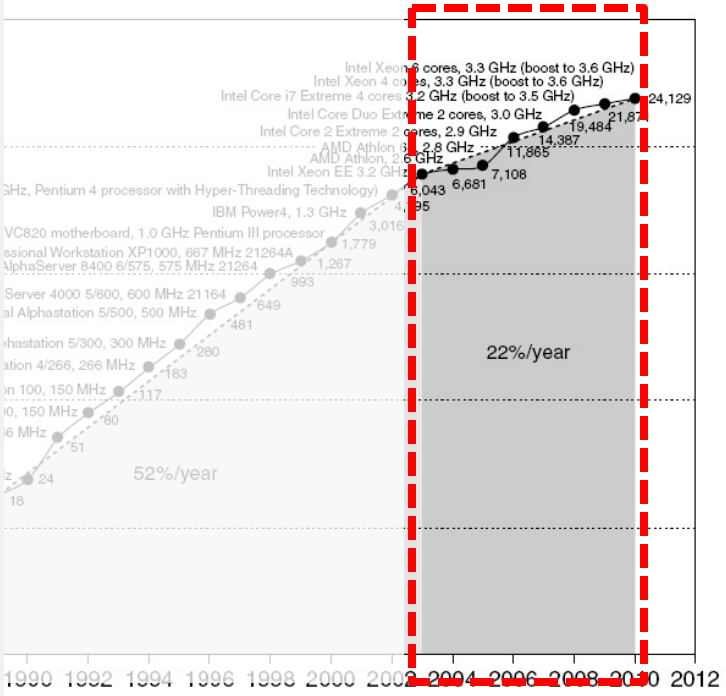
.....

Then two weeks ago, Intel, the world's largest chip maker, publicly acknowledged that it had hit a "thermal wall" on its microprocessor line. As a result, the company is changing its product strategy and disbanding one of its most advanced design groups. Intel also said that it would abandon two advanced chip development projects, code-named Tejas and Jayhawk.

Now, Intel is embarked on a course already adopted by some of its major rivals: obtaining more computing power by stamping multiple processors on a single chip rather than straining to increase the speed of a single processor.

...

**John Markoff, New York Times, May 17, 2004**



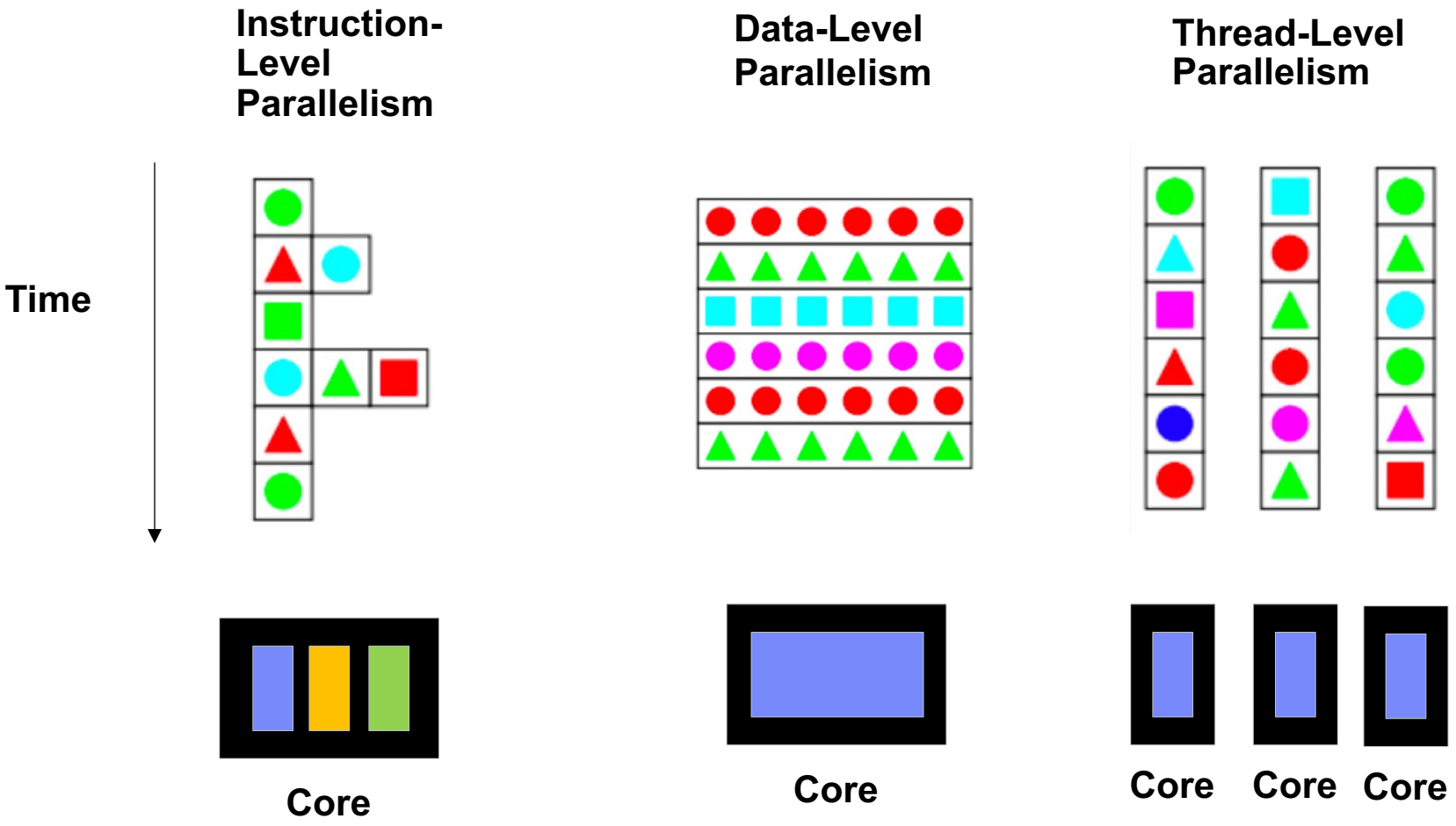
# Types of Parallelism

- **Instruction-Level Parallelism (ILP)**
  - Parallel execution of a sequence of instructions belonging to a specific thread
  - Superscalar, VLIW

- **Data-Level Parallelism (DLP)**
  - Applying a single instruction to a collection of data in parallel
  - SIMD instructions, GPU

- **Thread-Level Parallelism (TLP)**
  - Running tasks (threads) at the same time
  - Multi-core

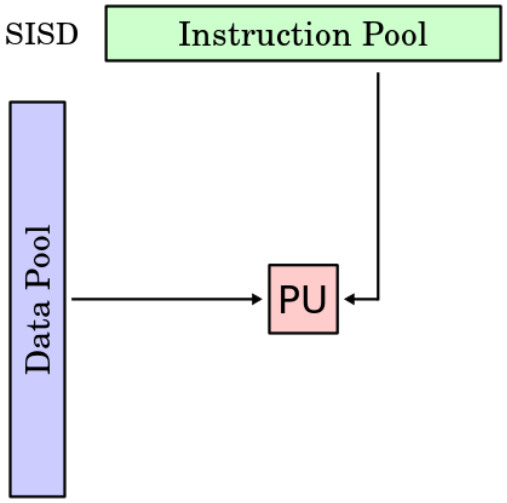
# Types of Parallelism (Cont'd)

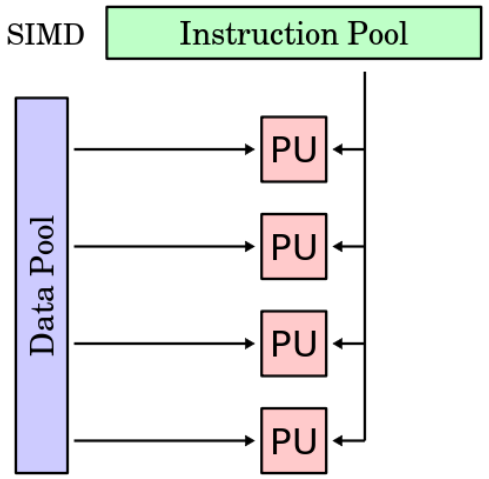


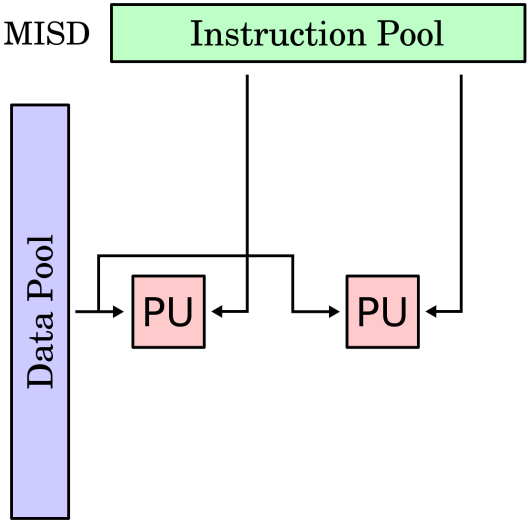
# Flynn's Classification

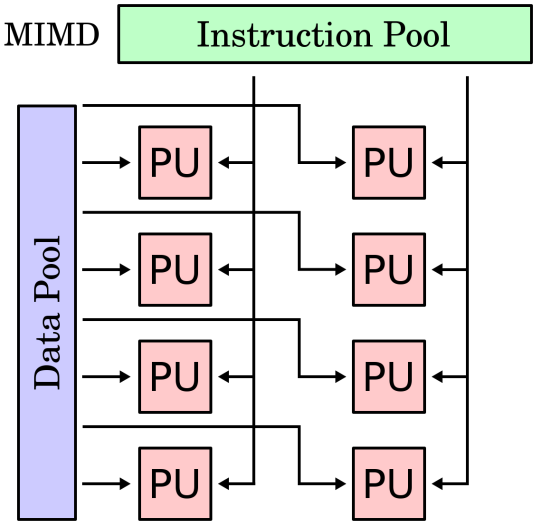
- **SISD** : Single Instruction Single Data Stream
  - Pipelining
  - Out-of-order Execution
  - Superscalar Processor
  - VLIW Processor
- **SIMD** : Single Instruction Multiple Data Stream
  - Array Processor
  - Vector Processor → GPU
- **MISD** : Multiple Instruction Single Data Stream
  - No commercial implementation
- **MIMD** : Multiple Instruction Multiple Data Stream
  - Shared-memory multiprocessor (e.g., Multi-Core)
  - Distributed-memory multiprocessors

# Flynn's Classification



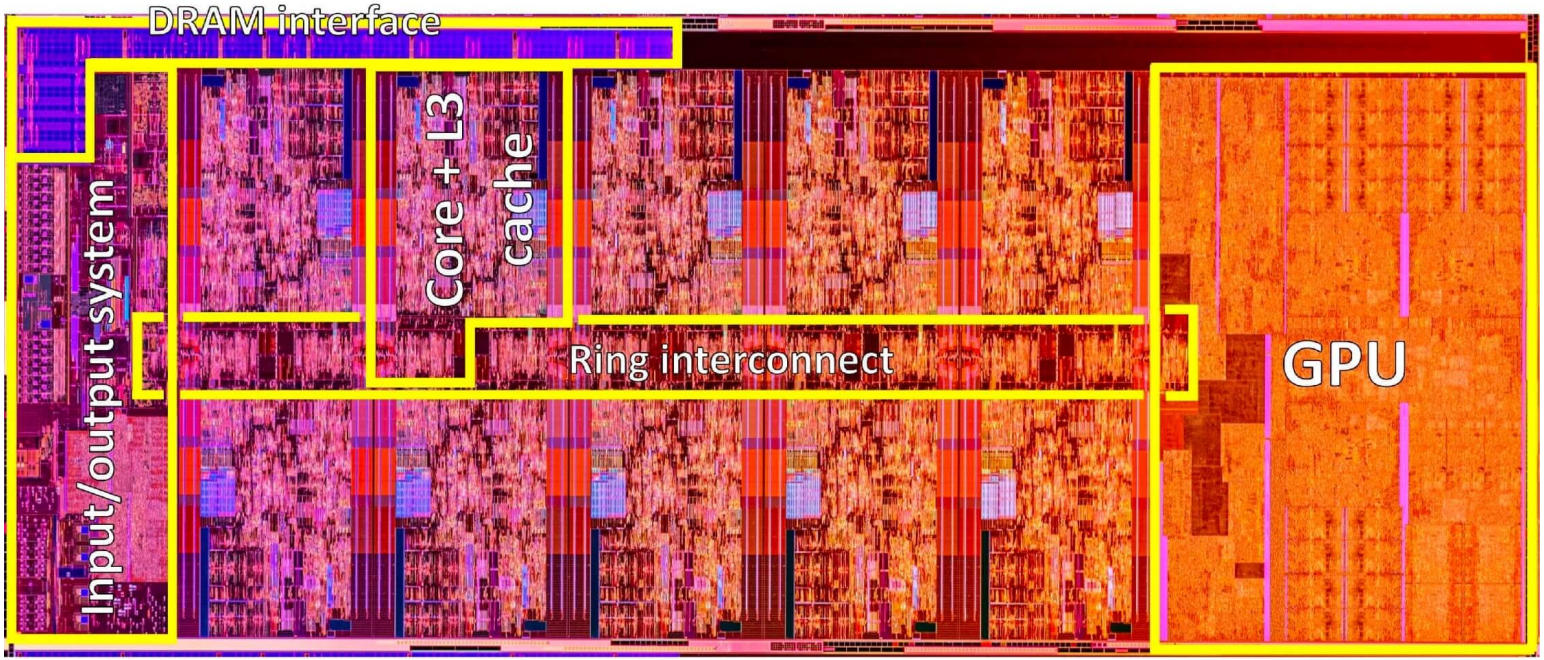



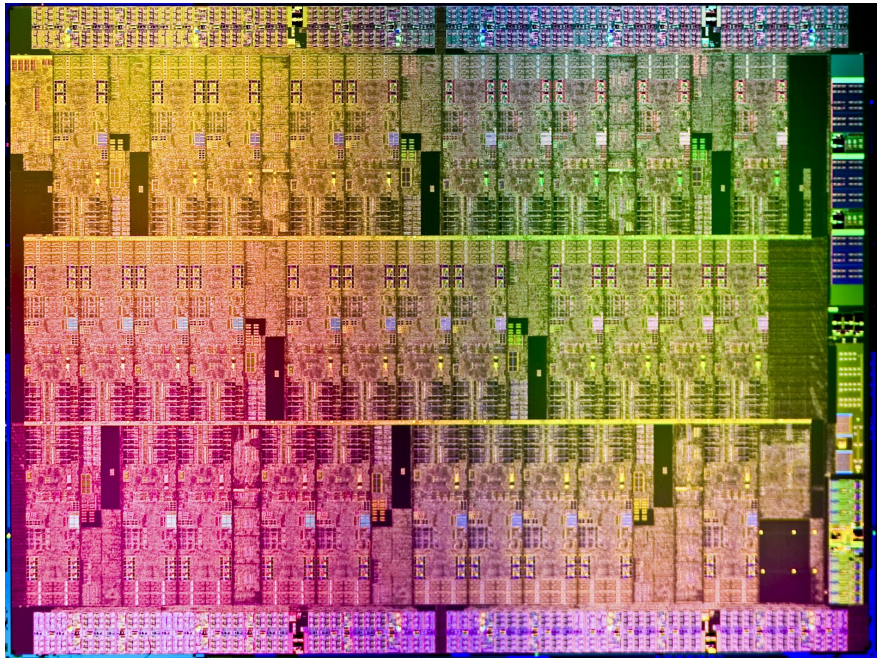
# Parallel Processor


- **Intel *Comet Lake* Core i9**
  - 10 Cores (20 threads), 3.7 GHz
  - GPU : UHD630, 1.2 GHz
  - ILP + DLP + TLP



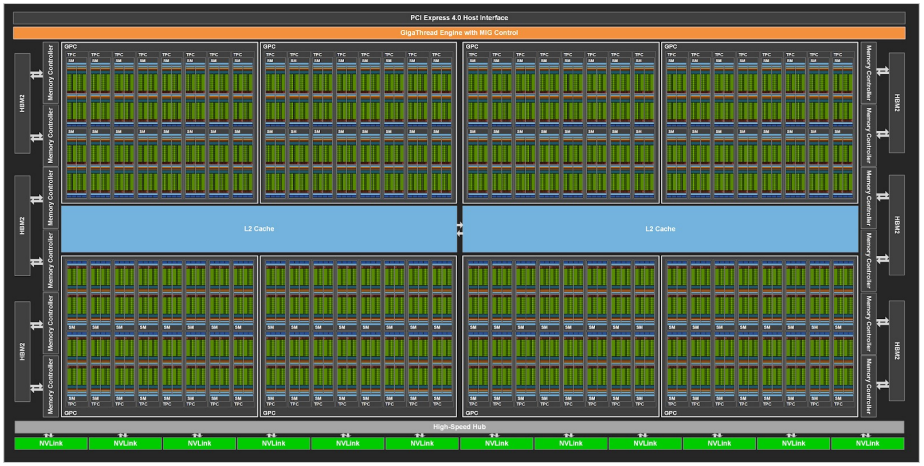
# Parallel Processor

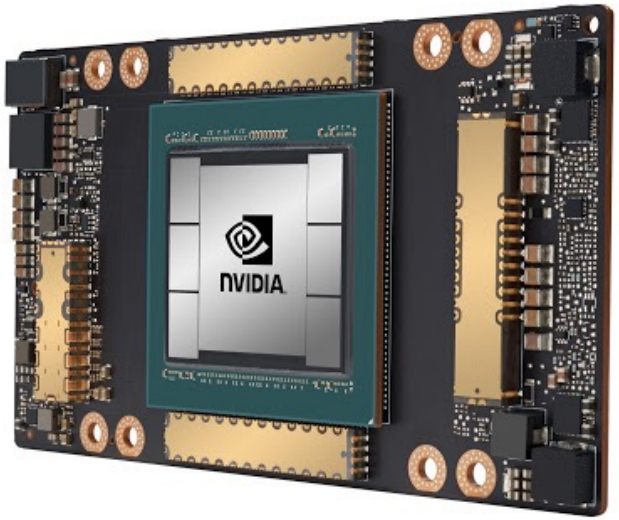
- Intel Xeon Phi 7290 coprocessor
- 72 cores, 1.7 GHz
- ILP + DLP + TLP

# Parallel Processor

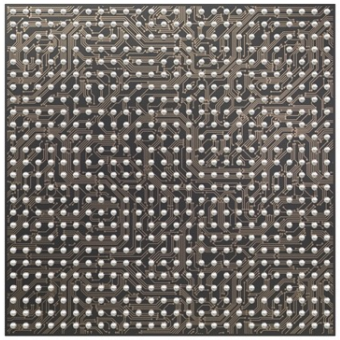
- NVIDIA Ampere A100
- 6912 Cores, 1.4GHz
- DLP+TLP

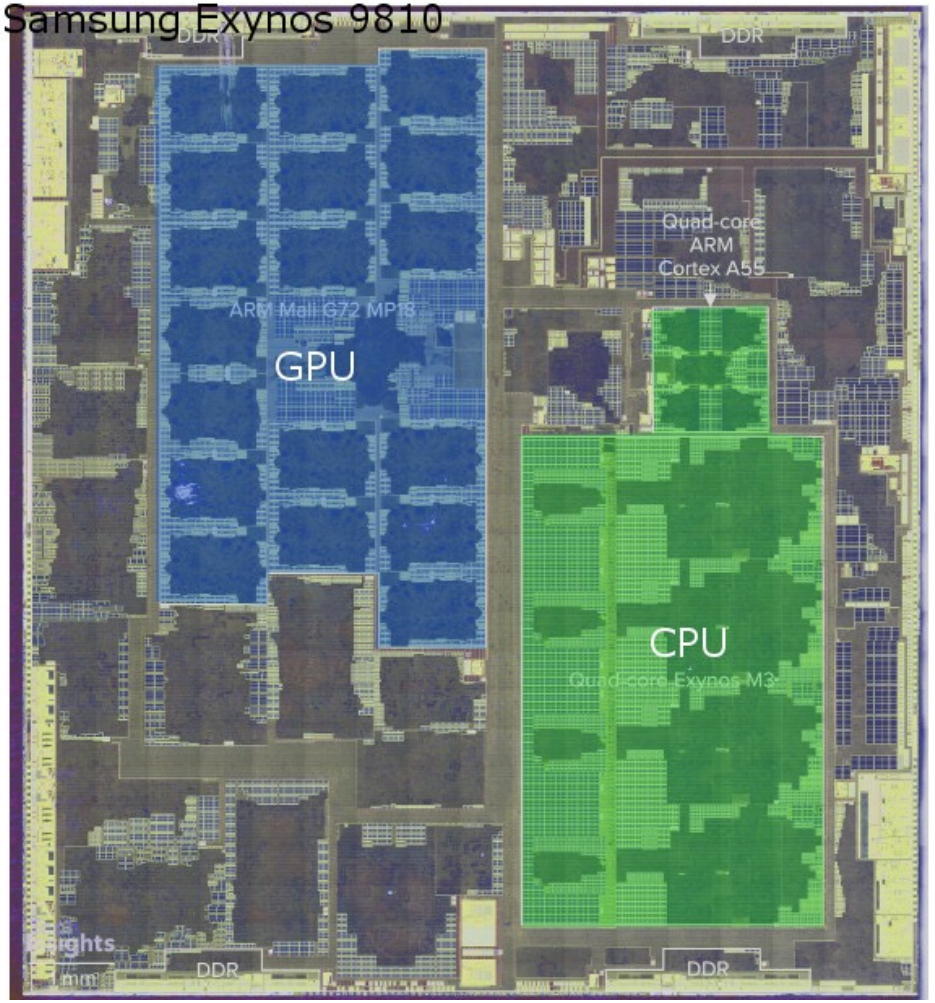



# Parallel Processor

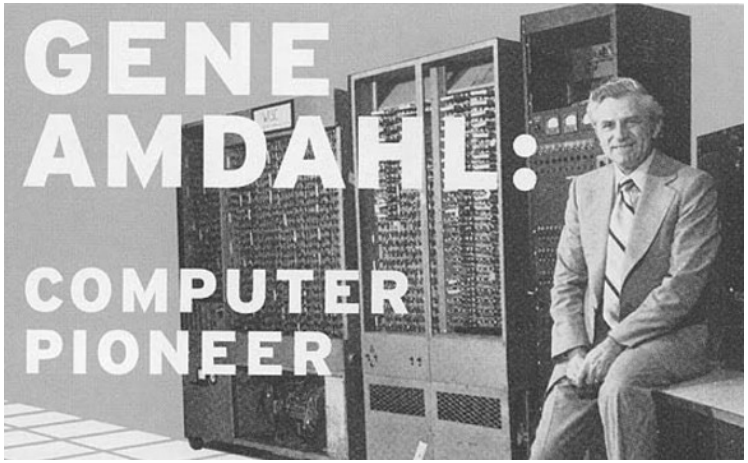
- 8 ARM Cores, Mali GPU, NPU
- ILP + DLP + TLP



# Amdahl's Law (I)

- **Gene Amdahl**, chief architect of IBM's first mainframe series found that there were some fairly stringent restrictions on how much of a speedup one could get for a given parallelized task. These observations were wrapped up in ***Amdahl's Law***

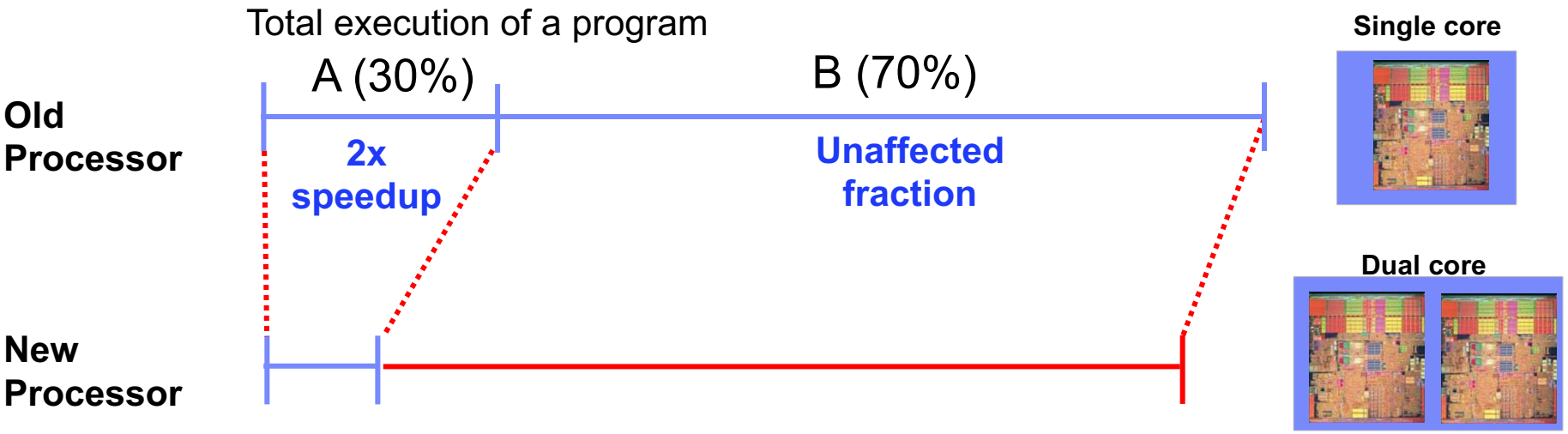



- often used in parallel computing to predict the theoretical maximum speedup using multiple processors
- The speedup of a program using multiple processors in parallel computing is **limited by the time needed for the sequential fraction of the program.**

# Amdahl's Law (II)

$$T_{improved} = \frac{T_{affected}}{improvement\ factor} + T_{unaffected}$$

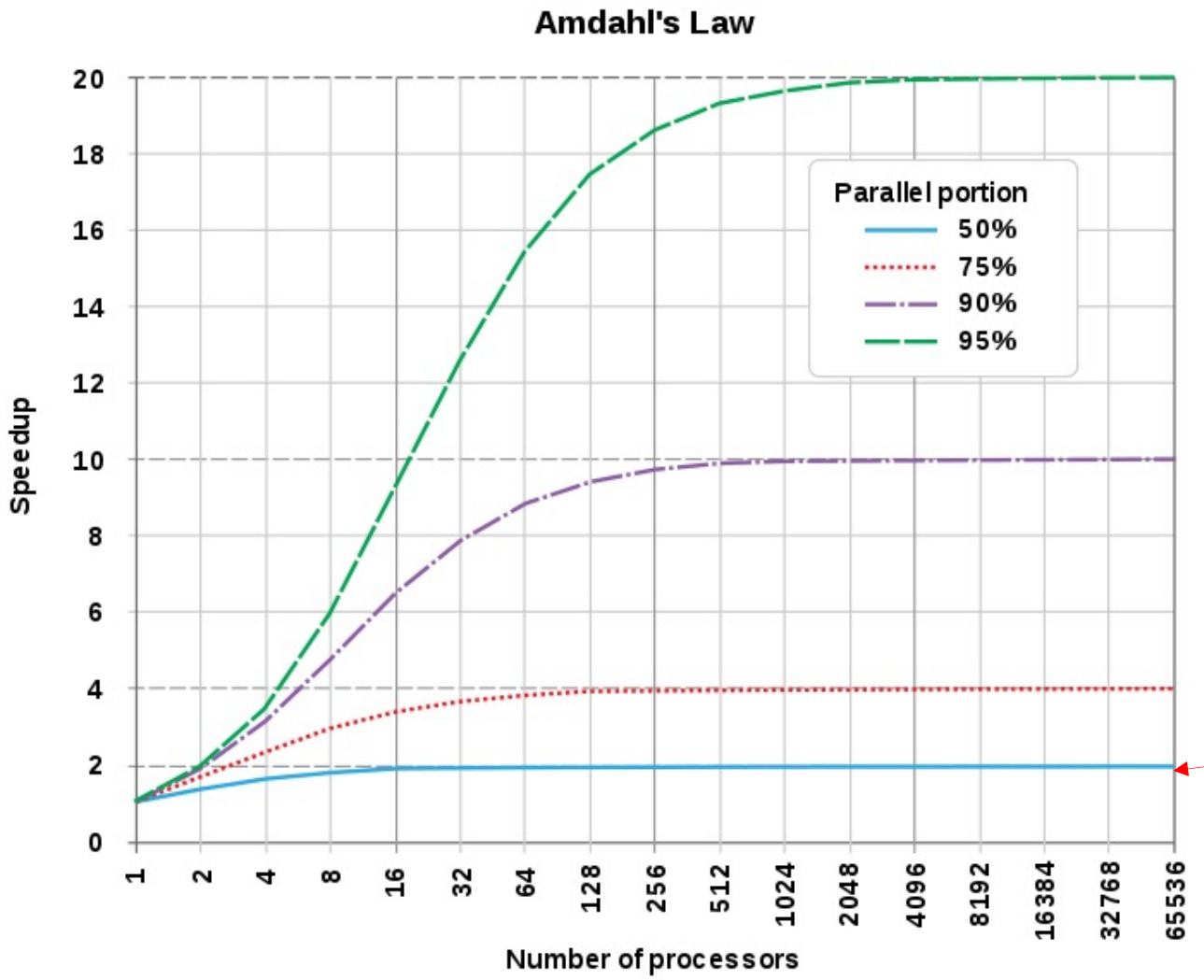
- **Example1**




**Execution time with new processor** = 0.3 T / 2 + 0.7 T

**Speedup** = T/(0.85 T) = 1.176

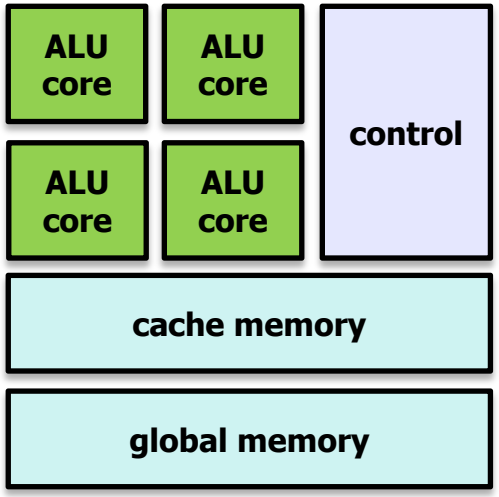
# Amdahl's Law (III)




2x speedup, regardless of the number of processors if the parallel portion is 50%

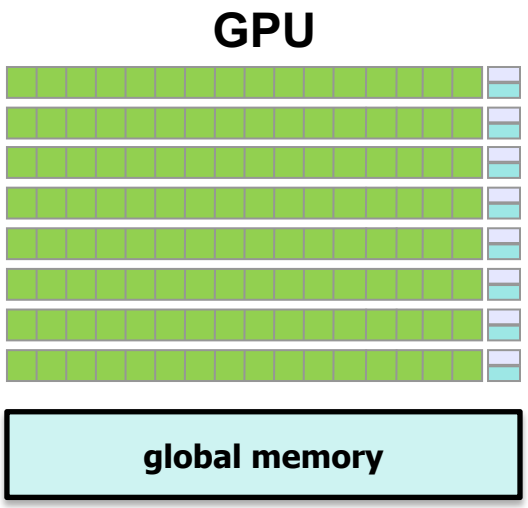
# Heterogeneous Parallel Computing (I)

- **CPUs : Latency Oriented Design**
  - designed to minimize the execution latency of a single thread
    - **Large caches**
      - Convert long latency memory accesses to short latency cache accesses
    - **Large control unit**
      - Branch prediction for reduced branch latency
      - Data forwarding for reduced data latency
    - **Powerful ALU**
      - Reduced operation latency
  - good for programs that have one or very few threads



# Heterogeneous Parallel Computing (II)

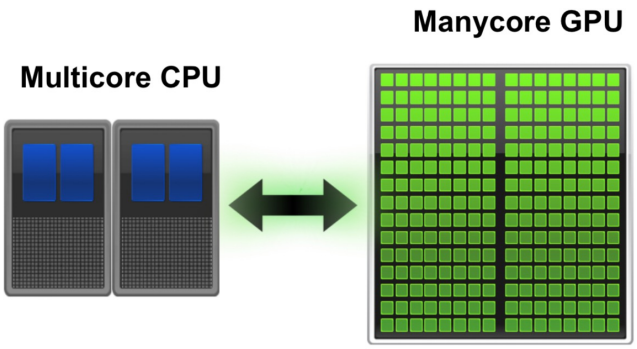
- **GPUs : Throughput Oriented Design**
  - **thread pool**
    - threads are pending when they need memory fetches
    - execute when they completed those fetch operations
  - **Small caches**
    - To boost memory throughput
  - **Simple control**
    - No branch prediction
    - No data forwarding
  - **Energy efficient ALUs**
    - Many, long latency but heavily pipelined for high throughput
  - Require massive number of threads to tolerate latencies

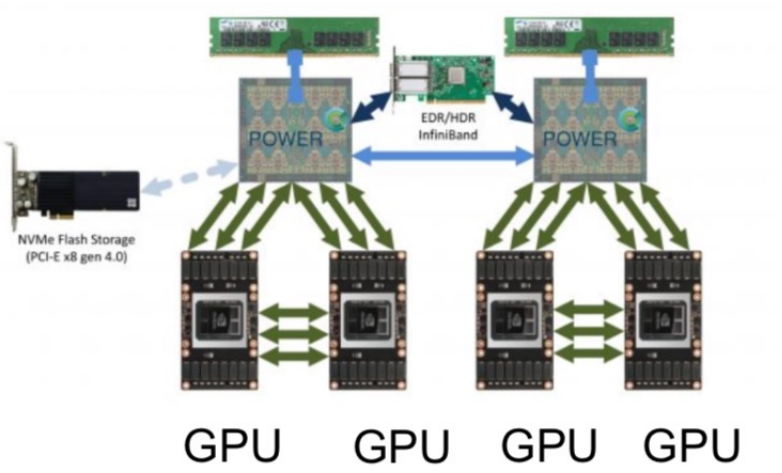


# Heterogeneous Parallel Computing (III)

- **CPUs** for sequential parts where latency matters
  - CPUs can be 10+ times faster than GPUs for sequential code

- **GPUs** for parallel parts where throughput wins
  - GPUs can be 10+ times faster than CPUs for parallel code



**The free lunch is over.. Now it's up to the programmers.** Adding more processors doesn't help much if programmers don't know how to use them

**Next..**

- Multicore Architecture