Introduction To Parallel Computing

이 정 근 (Jeong-Gun Lee), 유 동 훈(Dong-Hoon Yoo)

한림대학교 컴퓨터공학과, 임베디드 SoC 연구실
삼성전자
www.onchip.net
Email: **Jeonggun.Lee@hallym.ac.kr**

Contents

- Introduction to Parallel Computing / High Performance Computing (HPC)
- Concepts and terminology
- Parallel programming models
- Parallelizing your programs

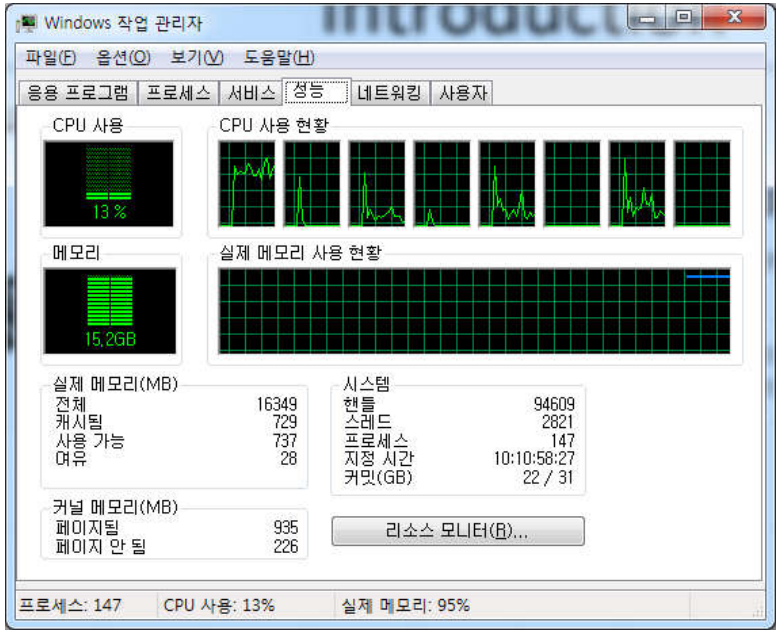
Parallel Computing

- Multicore/Manycore and GPU

인텔 코어i7-7세대 7700 (카비레이크) (정품) 표준PC

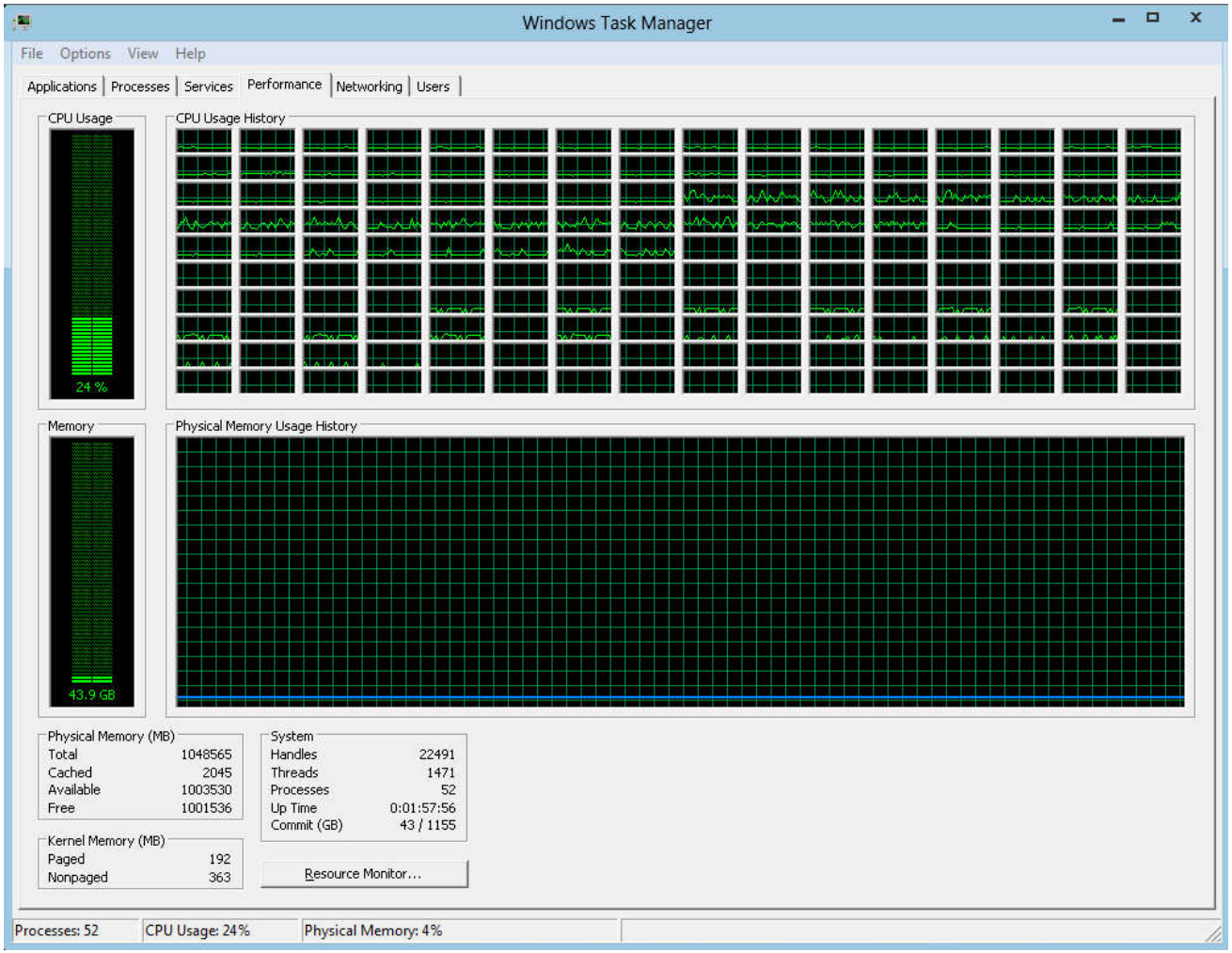
인텔(소켓1151) / 쿼드 코어 / 쓰레드 8개 / 64비트 / 3.6GHz / 8MB / 인텔 HD 630 / 350MHz / 65W / 하이퍼스레딩

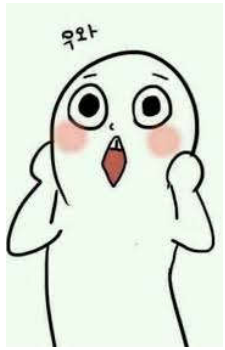
관련기사 인텔 공인대리점, 7세대 코어 프로세서 봄맞이 퀴즈 이벤트
사용기 [똑똑한 리뷰씨] 게임에 강한 프로세서! 7세대 인텔 코어 프로세서



Parallel Computing

- Multicore/Manycore and GPU





Parallel Computing

- **First, Wiki Definition …**

병렬 컴퓨팅

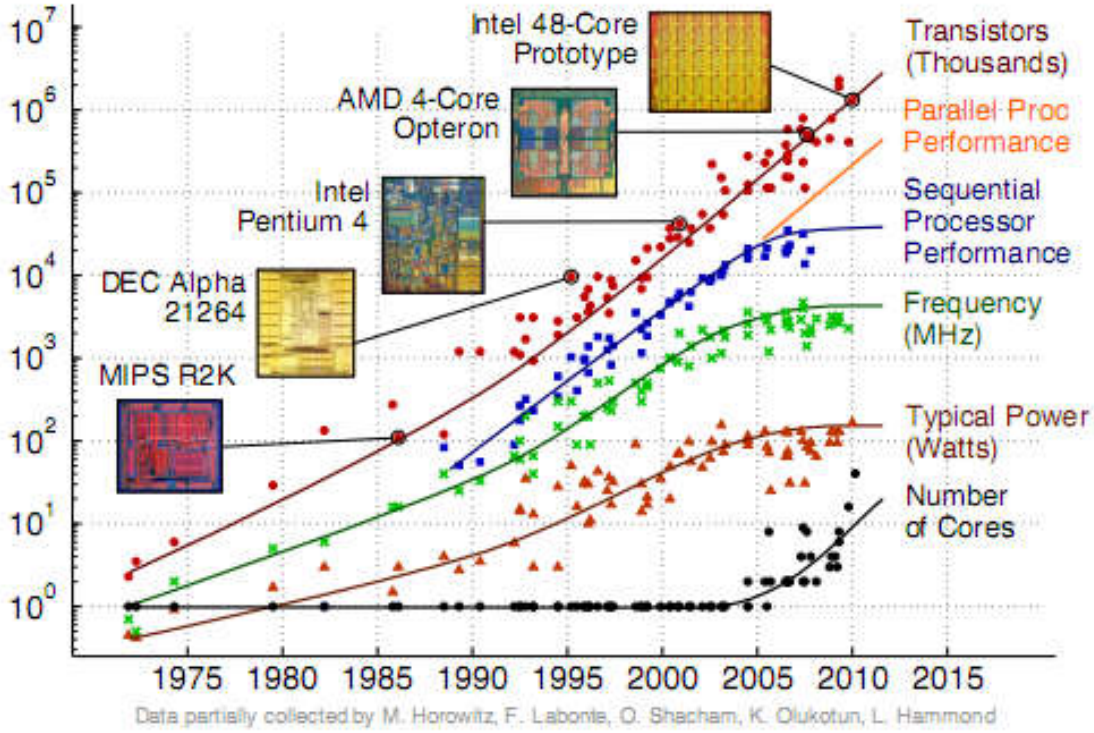
위키백과, 우리 모두의 백과사전.

병렬 컴퓨팅(parallel computing) 또는 **병렬 연산**은 동시에 많은 계산을 하는 연산의 한 방법이다. 크고 복잡한 문제를 작게 나눠 동시에 병렬적으로 해결하는 데에 주로 사용되며[1], 병렬 컴퓨팅에는 여러 방법과 종류가 존재한다. 그 예로, 비트 수준, 명령어 수준, 데이터, 작업 병렬 처리 방식 등이 있다. 병렬 컴퓨팅은 오래전부터 주로 고성능 연산에 이용되어 왔으며, 프로세서 주파수[2] 의 물리적인 한계에 다가가면서 문제 의식이 높아진 이후에 더욱 주목받게 되었다. 최근 컴퓨터 이용에서 발열과 전력 소모에 대한 관심이 높아지는 것과 더불어 멀티 코어 프로세서를 핵심으로 컴퓨터 구조에서 강력한 패러다임으로 주목받게 되었다[3].

프로세서 주파수 의 물리적인 한계에 다가가면서 문제 의식이 높아진 이후에 더욱 주목 받게 되었다. 최근 컴퓨터 이용에서 발열과 전력 소모에 대한 관심이 높아지는 것과 더불어 멀티 코어 프로세서를 핵심으로 컴퓨터 구조에서 강력한 패러다임으로 주목

Parallel Computing: Why ?

프로세서 주파수 의 물리적인 한계에 다가가면서 문제 의식이 높아진 이후에 더욱 주목 받게 되었다. 최근 컴퓨터 이용에서 발열과 전력 소모에 대한 관심이 높아지는 것과 더불어 멀티 코어 프로세서를 핵심으로 컴퓨터 구조에서 강력한 패러다임으로 주목



Prepared by C. Batten - School of Electrical and Computer Engineering - Cornell University - 2005 - retrieved Dec 12 2012 - http://www.csl.cornell.edu/courses/ece5950/handouts/ece5950-overview.pdf

Intel® Xeon Phi™ Processor 7290F

16GB, 1.50 GHz, 72 core

Xeon Phi 7200 Series	sSpec Number	Cores (Threads)	Frequency	Turbo
Xeon Phi 7210[75]	SR2ME (B0) SR2X4 (B0)	64 (256)	1300 MHz	1500 MHz
Xeon Phi 7210F[76]	SR2X5 (B0)	64 (256)	1300 MHz	1500 MHz
Xeon Phi 7230[77]	SR2MF (B0) SR2X3 (B0)	64 (256)	1300 MHz	1500 MHz
Xeon Phi 7230F[78]	SR2X2 (B0)	64 (256)	1300 MHz	1500 MHz
Xeon Phi 7250[79]	SR2MD (B0) SR2X1 (B0)	68 (272)	1400 MHz	1600 MHz
Xeon Phi 7250F[81]	SR2X0 (B0)	68 (272)	1400 MHz	1600 MHz
Xeon Phi 7290[82]	SR2WY (B0)	72 (288)	1500 MHz	1700 MHz
Xeon Phi 7290F[83]	SR2WZ (B0)	72 (288)	1500 MHz	1700 MHz

Each core will have two 512-bit vector units and will support AVX-512 SIMD instructions

Parallel Computing: Why ?

프로세서 주파수 의 물리적인 한계에 다가가면서 문제 의식이 높아진 이후에 더욱 주목 받게 되었다. 최근 컴퓨터 이용에서 발열과 전력 소모에 대한 관심이 높아지는 것과 더불어 멀티 코어 프로세서를 핵심으로 컴퓨터 구조에서 강력한 패러다임으로 주목

- Worse news: Power (normalized to i486) trends

Small is Beautiful !

That does not immediately imply that "smallest is best" !

- **Parallelism** is an **energy-efficient way to achieve performance** [Chandrakasan et al 1992]
- A larger number of smaller processing elements allows a finer-grained ability to perform dynamic voltage scaling and power down

Parallel Computing: Why ?

프로세서 주파수 의 물리적인 한계에 다가가면서 문제 의식이 높아진 이후에 더욱 주목 받게 되었다. 최근 컴퓨터 이용에서 발열과 전력 소모에 대한 관심이 높아지는 것과 더불어 멀티 코어 프로세서를 핵심으로 컴퓨터 구조에서 강력한 패러다임으로 주목

- Multi-core Challenges: The **3 P**'s
  - Performance challenge
  - **Power efficiency challenge**

Processor	Power	Perf.	Power Efficiency
Itanium 2	100W	1	1
RISC*	1/2W	1/8X**	25X

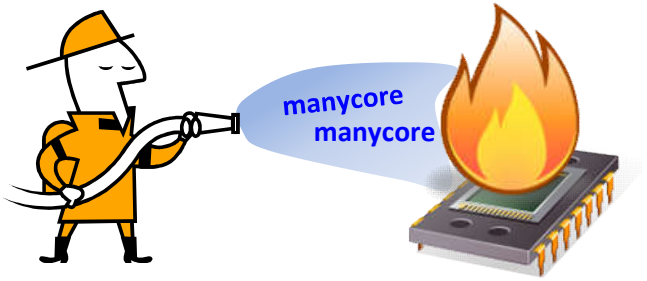
Assuming 130nm
* 90's RISC at 405 MHz
** e.g., Timberwolf (SpecInt)

  - Programming challenge

Anant Agarwal (professor at MIT), "Going Multi-core: Opportunities, Challenges and Dreams" in his keynote talk at MULTI-CORE EXPO, Mar. 2006

Parallel Computing: Why ?

프로세서 주파수 의 물리적인 한계에 다가가면서 문제 의식이 높아진 이후에 더욱 주목 받게 되었다. 최근 컴퓨터 이용에서 발열과 전력 소모에 대한 관심이 높아지는 것과 더불어 멀티 코어 프로세서를 핵심으로 컴퓨터 구조에서 강력한 패러다임으로 주목

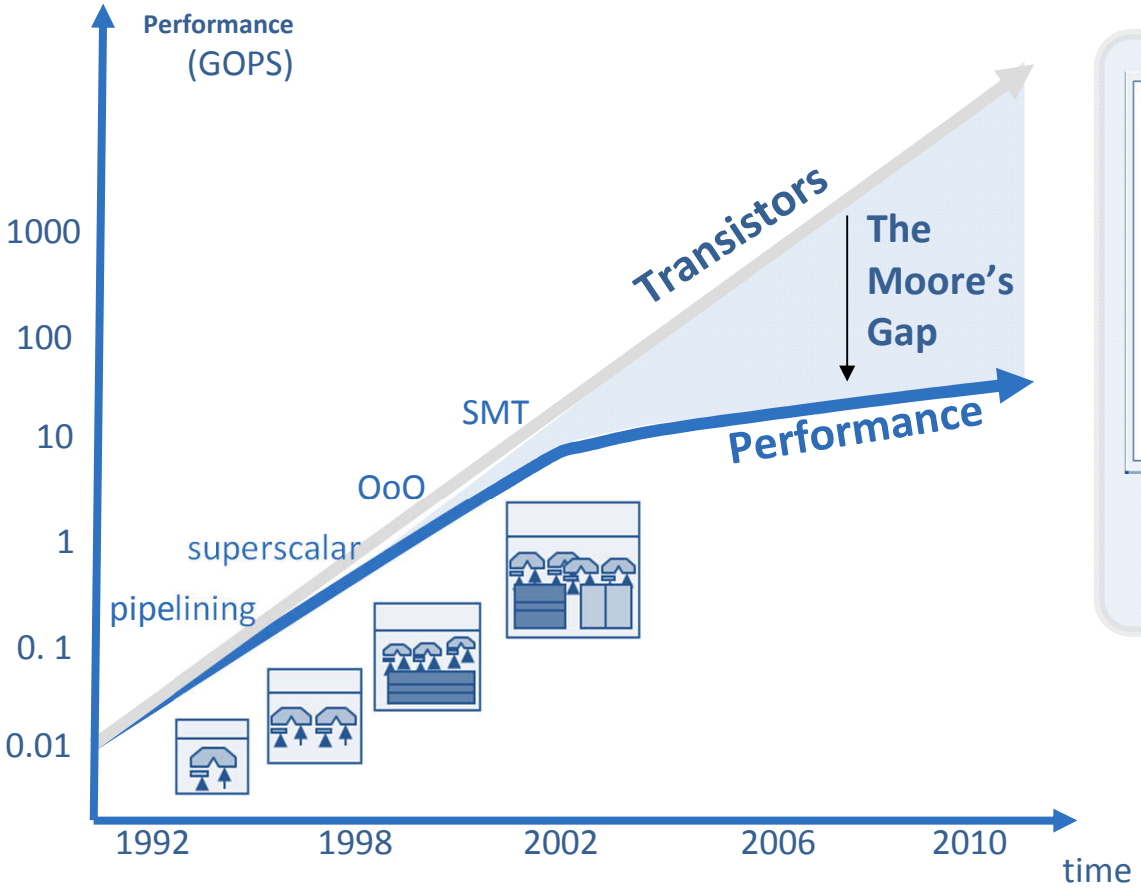


+ Diminishing Return

Parallel Computing: Why ?

프로세서 주파수 상 물리적인 한계에 다가가면서 문제 인식이 높아진 이후에 더욱 주목 받게 되었다. 최근 컴퓨터 [illegible] 대한 [illegible] 것과 더불어 멀티 코어 프로세서를 핵심으로 컴퓨터 구조에서 강력한 패러다임으로 주목

Diminishing Return



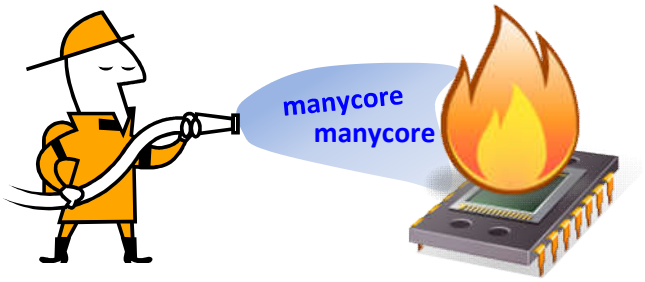
Pentium 3	Pentium 4
- 1 GHz	- 1.4 GHz
- Year 2000	- Year 2000
- 0.18 um	- 0.18 um
- 28M Tr. →	- 42M Tr. 50%↑
343 →	393 15%↑
(SpecInt 2000)	(SpecInt 2000)

1. ***Diminishing returns*** from single CPU

2. ***Wire*** delays
3. ***Power*** envelopes

Parallel Computing: Why ?

프로세서 주파수 의 물리적인 한계에 다가가면서 문제 의식이 높아진 이후에 더욱 주목 받게 되었다. 최근 컴퓨터 이용에서 발열과 전력 소모에 대한 관심이 높아지는 것과 더불어 멀티 코어 프로세서를 핵심으로 컴퓨터 구조에서 강력한 패러다임으로 주목



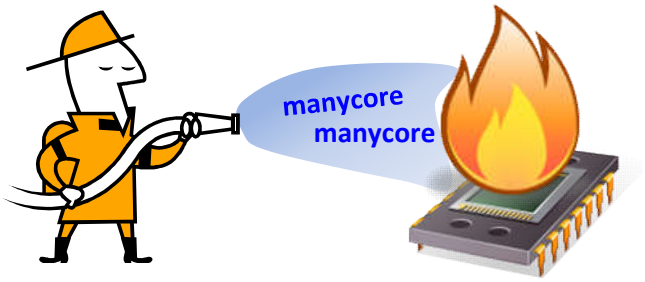
+ Diminishing Return

Manycore

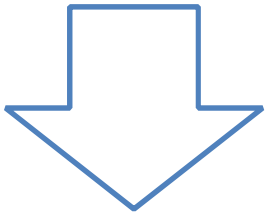
➔ Parallel Computing

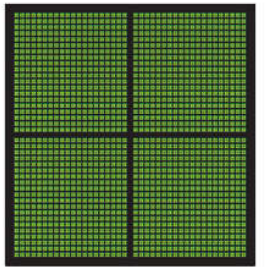
Parallel Computing: Why ?

프로세서 주파수 의 물리적인 한계에 다가가면서 문제 의식이 높아진 이후에 더욱 주목 받게 되었다. 최근 컴퓨터 이용에서 발열과 전력 소모에 대한 관심이 높아지는 것과 더불어 멀티 코어 프로세서를 핵심으로 컴퓨터 구조에서 강력한 패러다임으로 주목



+ Diminishing Return





GPU

Manycore

➔ Parallel Computing

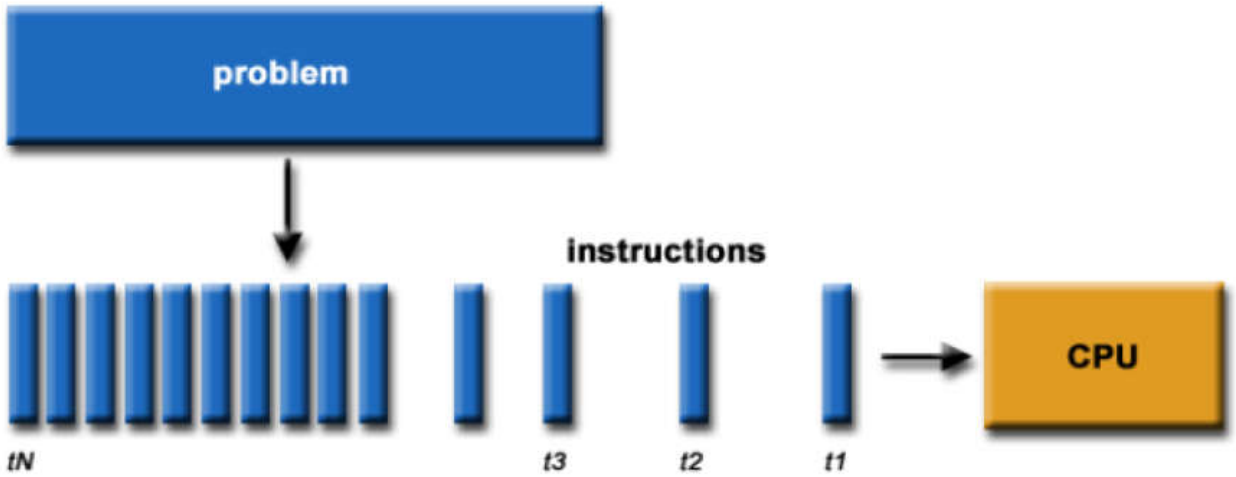
Parallel Computing: Why ?

프로세서 주파수 의 물리
되었다. 최근 컴퓨터 이
코어 프로세
 이후에 더욱 주목 받게
는 것과 더불어 멀티
으로 주목

➔ Parallel Computing

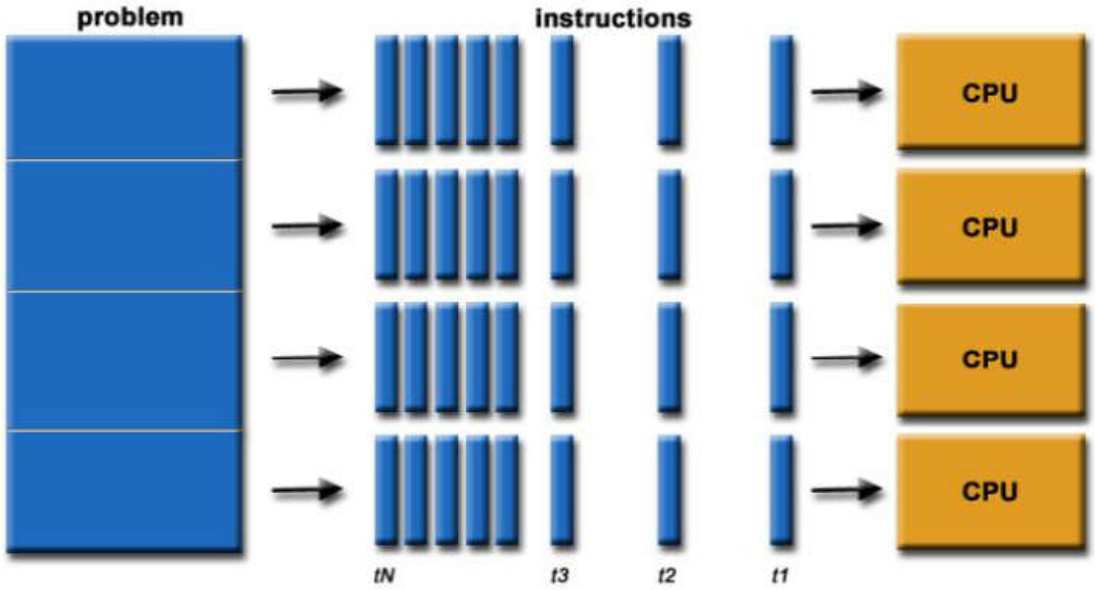
Serial Computation

- Traditionally software has been written for serial computations:
  - To be run on a single computer having a single Central Processing Unit (CPU)
    - A problem is broken into a discrete set of instructions(컴파일)
    - Instructions are executed one after another (순차적으로)
    - Only one instruction can be executed at any moment in time



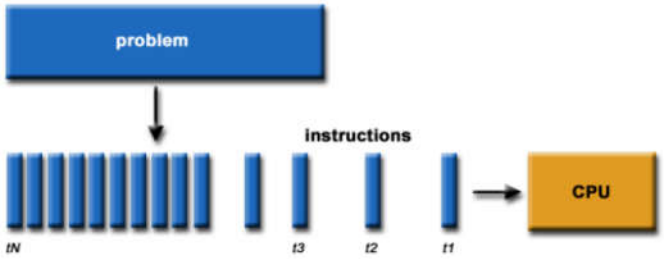
Parallel Computing

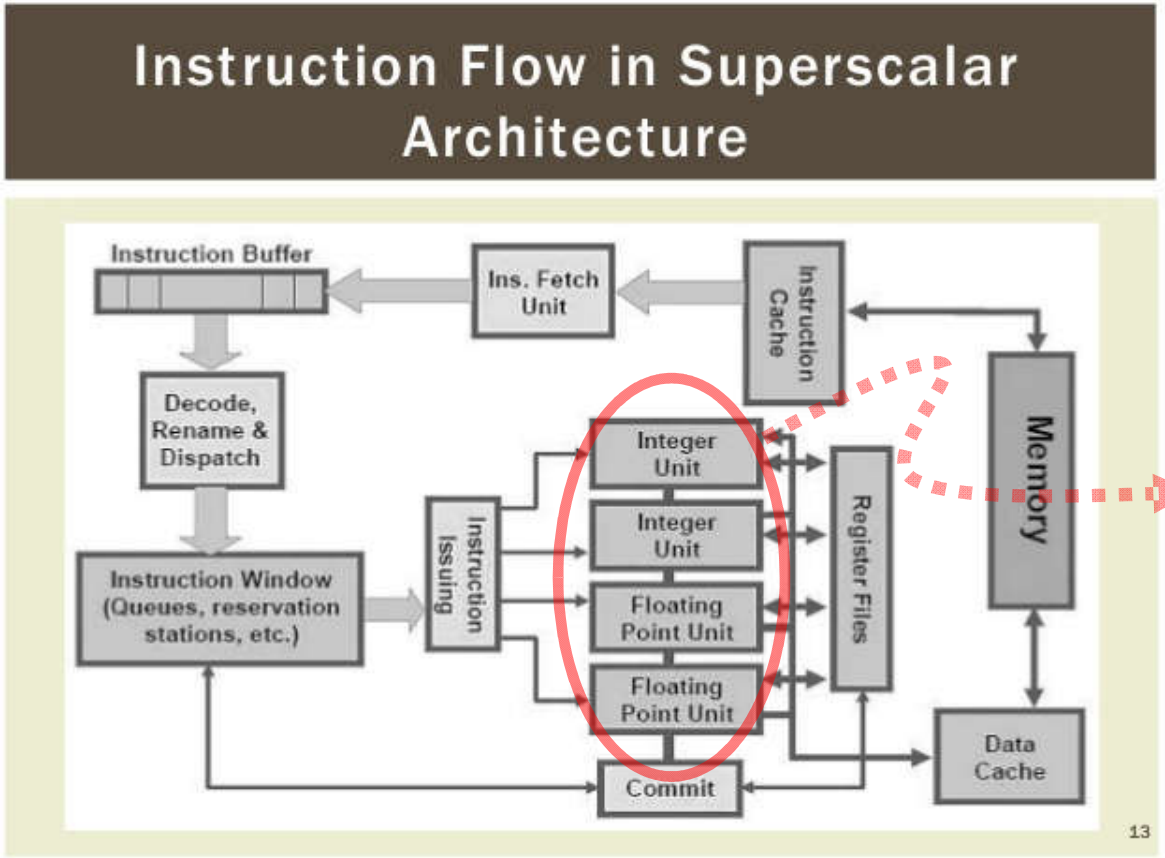
- Simply, parallel computing is the simultaneous use of multiple compute resources to solve a computational problem:
  - To be run using multiple CPUs
    - A problem is broken into discrete parts that can be solved concurrently (Task-Level or Data-Level Parallelism)
    - Each part is further broken down to a series of instructions
    - Instructions from each part execute simultaneously on different CPUs



Single CPU → No Parallelism ?

- If a CPU has multiple functional units ?



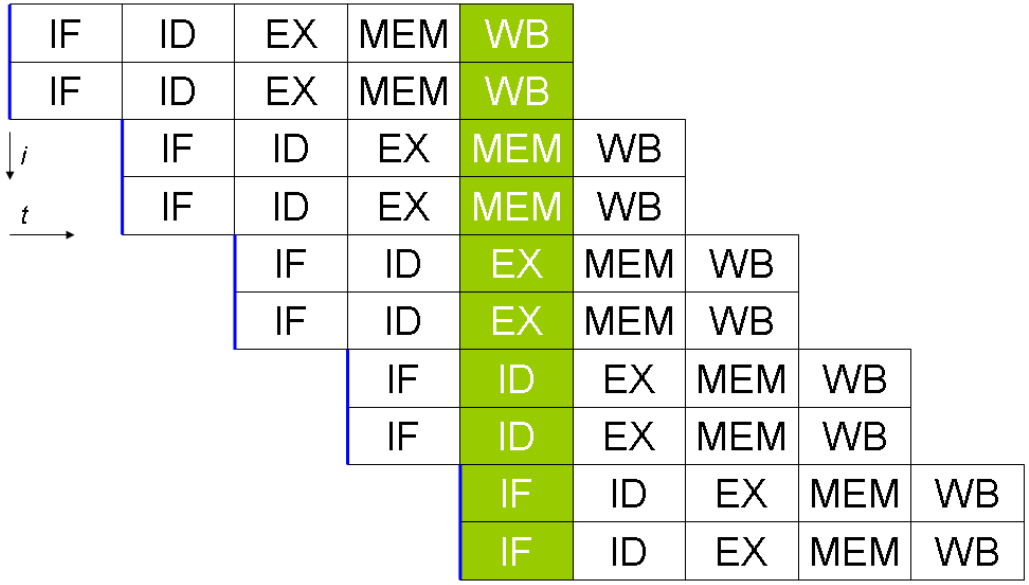


Instruction Level Parallelism (ILP)

Instruction Level Parallelism

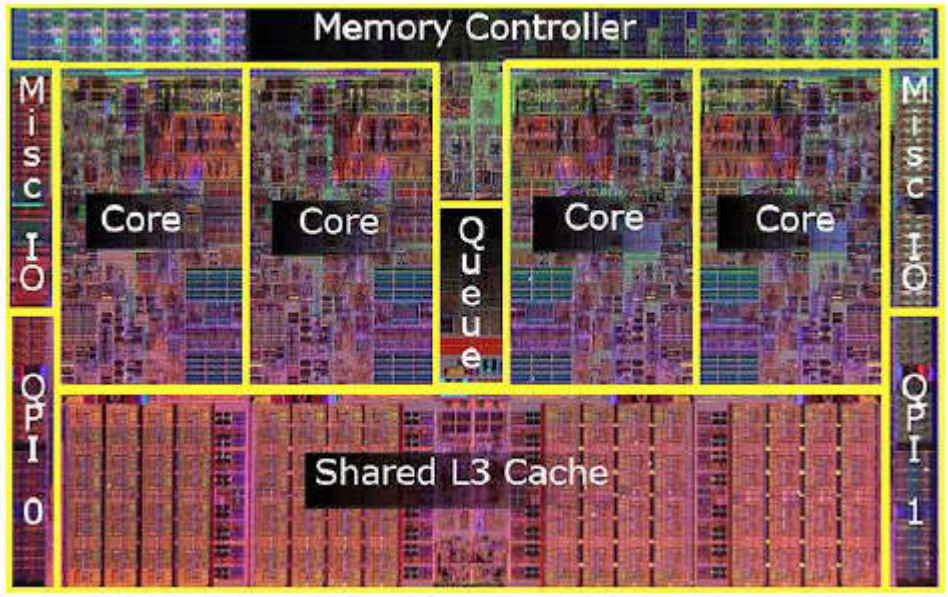
- **ILP** = Instruction Level Parallelism
  - Even in a single CPU, more than one instruction can be executed at a time with advanced microarchitecture techniques

Instr No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7



Parallel Computers

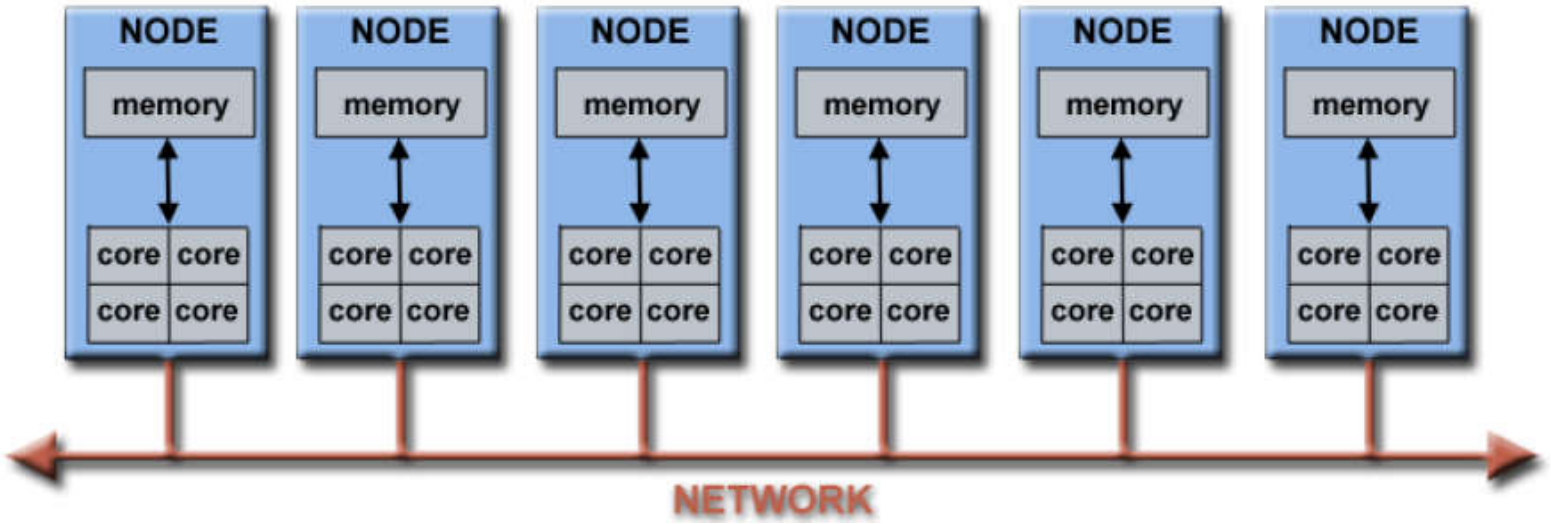
- Virtually all stand-alone computers today are parallel machines from a hardware perspective:
  - Multiple functional units (floating point, integer, GPU, etc.)
  - Multiple execution units / cores
  - Multiple hardware threads



Intel Core i7 CPU and its major components

Parallel Computers

- Networks connect multiple stand-alone computers (nodes) to create larger parallel computer clusters
  - Each compute node is a multi-processor parallel computer in itself
  - Multiple compute nodes are networked together with an InfiniBand network
  - Special purpose nodes, also multi-processor, are used for other purposes



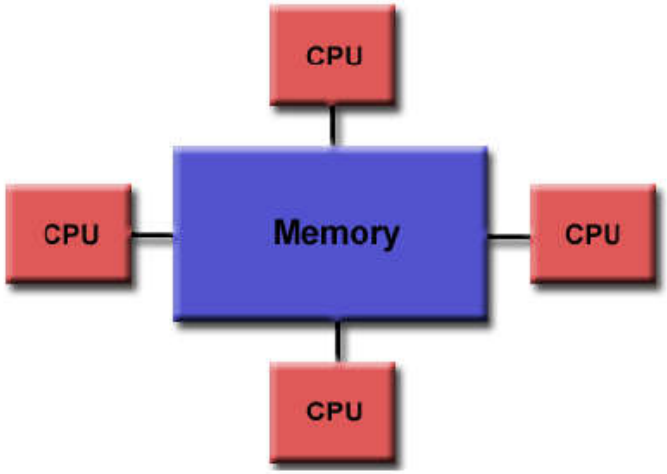
HPC Terminology 1

- **Supercomputing / High-Performance Computing (HPC)**
- **Flop(s)** – Floating point operation(s)
- **Node** – a stand alone computer
- **CPU / Core** – a modern CPU usually has several cores (individual processing units)
- **Task** – a logically discrete section from the computational work
- **Communication** – data exchange between parallel tasks

HPC Terminology 2

- **Speedup** – time of serial execution / time of parallel execution
- **Massively Parallel** – refer to hardware of parallel systems with many processors (“many” = hundreds of thousands)
- **Pleasantly Parallel** – solving many similar but independent tasks simultaneously. Requires very little communication (***Embarrassingly Parallel***)
- **Scalability** - a proportionate increase in parallel speedup with the addition of more processors

Parallel Computer Memory Architectures



- ***Shared Memory (공유메모리):***
  - Multiple processors can operate independently, but share the same memory resources
  - Changes in a memory location caused by one CPU are visible to all processors

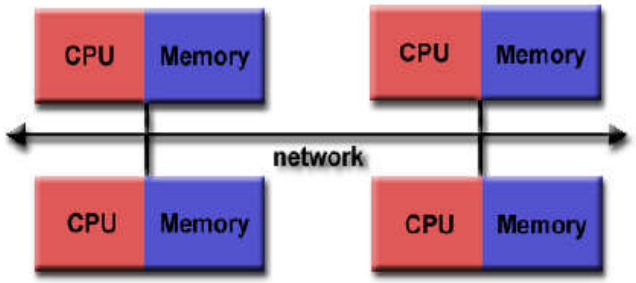
Advantages:

- Global address space provides a user-friendly programming perspective to memory
- Fast and uniform data sharing due to proximity of memory to CPUs

Disadvantages:

- Lack of scalability between memory and CPUs. Adding more CPUs increases traffic on the shared memory-CPU path
- Programmer responsibility for “correct” access to global memory

Parallel Computer Memory Architectures



- ***Distributed Memory(분산메모리):***
  - Requires a communication network to connect inter-processor memory
  - Processors have their own local memory.
  - Changes made by one CPU have no effect on others
  - Requires communication to exchange data among processors

Advantages:

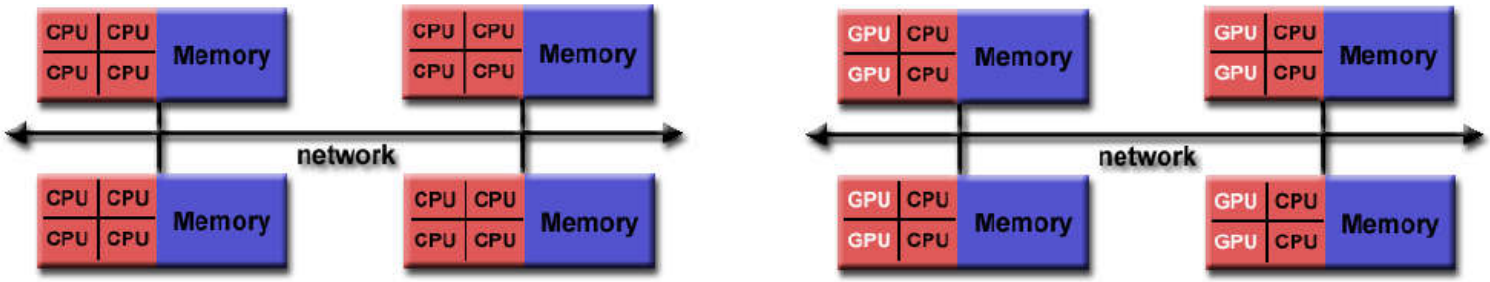
- Memory is scalable with the number of CPUs
- Each CPU can rapidly access its own memory without overhead incurred with trying to maintain global cache coherency

Disadvantages:

- Programmer is responsible for many of the details associated with data communication between processors

Parallel Computer Memory Architectures

- ***Hybrid Distributed-Shared Memory(분산공유메모리)***:
  - The largest and fastest computers in the world today employ both shared and distributed memory architectures.



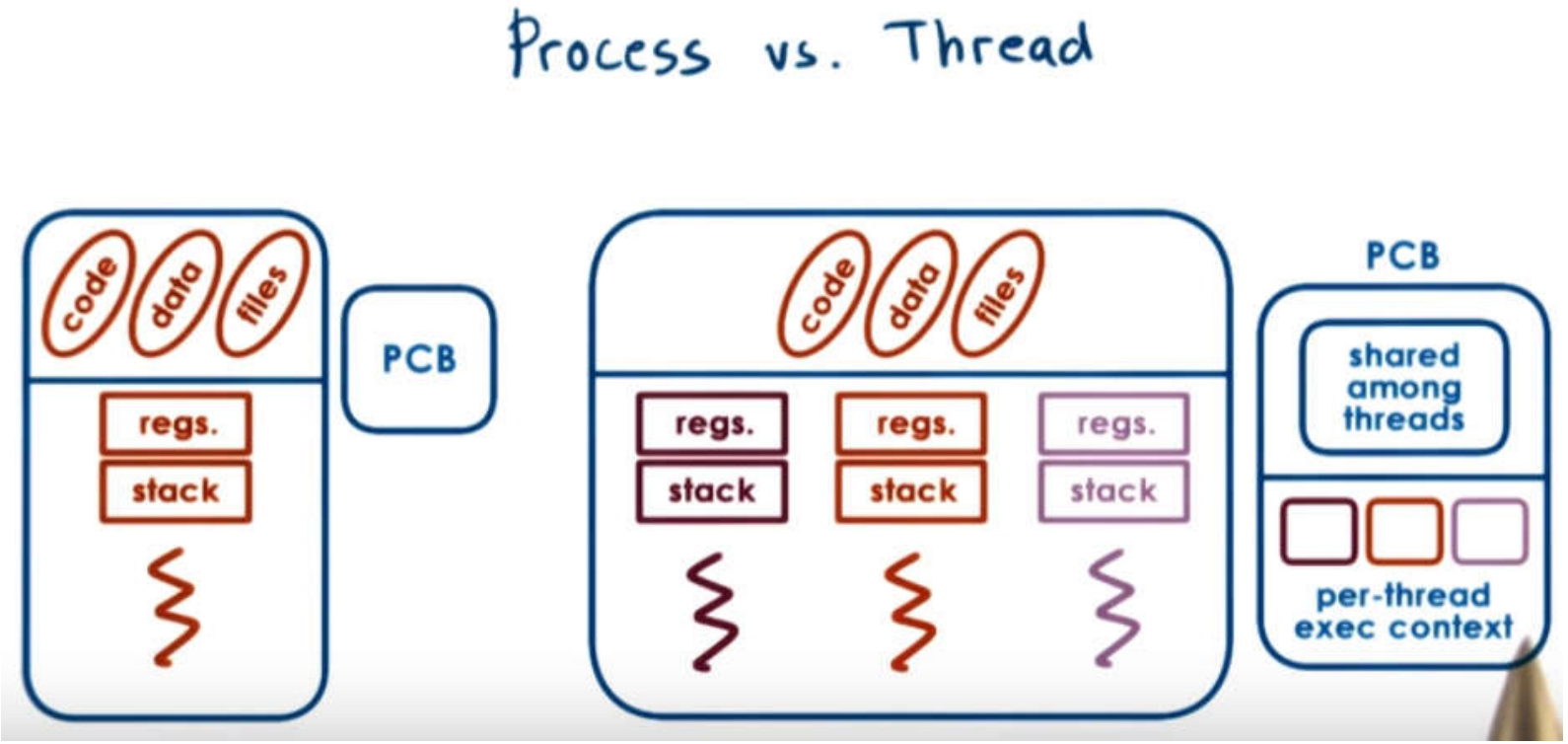
- Shared memory component can be a shared memory machine and/or GPU
- Processors on a compute node share same memory space
- Requires communication to exchange data between compute nodes

Parallel Programming Models

- Parallel Programming Models exist as an abstraction above hardware and memory architectures
  - Shared Threads Models (Pthreads, OpenMP)
  - Distributed Memory / Message Passing (MPI)
  - Single Instruction/Program Multiple Data (SIMD, SPMD)
    - Vector Processing (MMX, SSE, AVX, …)
    - Single Instruction Multiple Thread (SIMT for GPU)
  - …

Shared Threads Models

- **POSIX Threads**



POSIX (**포직스**, /ˈpɒzɪks/)는 **이식 가능 운영 체제 인터페이스**(移植可能運營體制 interface, **p**ortable **o**perating **s**ystem **i**nterface)의 약자로, 서로 다른 UNIX OS의 공통 API를 정리하여 이식성이 높은 유닉스 응용 프로그램을 개발하기 위한 목적으로 IEEE가 책정한 애플리케이션 인터페이스 규격이다.
POSIX의 마지막 글자 X는 유닉스 호환 운영체제에 보통 X가 붙는 것에서 유래한다.

Shared Threads Models

- **POSIX Threads**

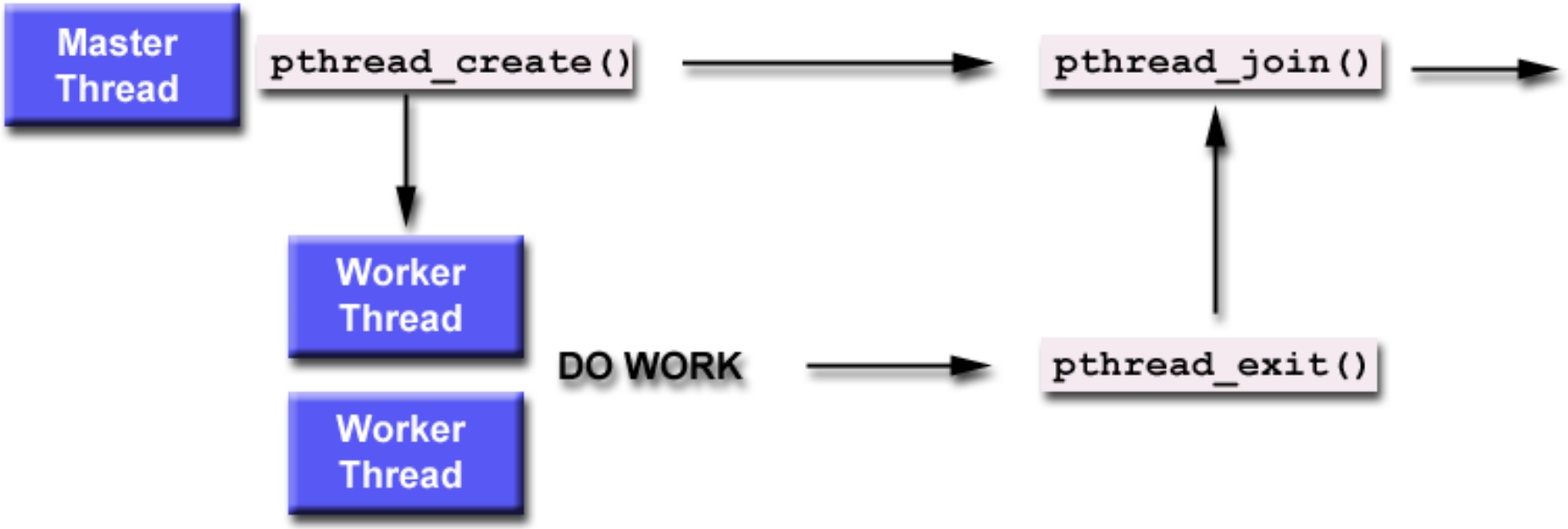
POSIX (**포직스**, /ˈpɒzɪks/)는 **이식 가능 운영 체제 인터페이스**(移植可能運營體制 interface, **p**ortable **o**perating **s**ystem **i**nterface)의 약자로, 서로 다른 UNIX OS의 공통 API를 정리하여 이식성이 높은 유닉스 응용 프로그램을 개발하기 위한 목적으로 IEEE가 책정한 애플리케이션 인터페이스 규격이다. POSIX의 마지막 글자 X는 유닉스 호환 운영체제에 보통 X가 붙는 것에서 유래한다.

- Library based; requires explicit parallel coding
- C Language only; Interfaces for Perl, Python and others exist
- Commonly referred to as **Pthreads**
- Most hardware vendors now offer Pthreads in addition to their proprietary threads implementations
- Very explicit parallelism; requires significant programmer attention to detail

Shared Threads Models

- **POSIX Threads**

POSIX (**포직스**, /ˈpɒzɪks/)는 **이식 가능 운영 체제 인터페이스**(移植可能運營體制 interface, **p**ortable **o**perating **s**ystem **i**nterface)의 약자로, 서로 다른 UNIX OS의 공통 API를 정리하여 이식성이 높은 유닉스 응용 프로그램을 개발하기 위한 목적으로 IEEE가 책정한 애플리케이션 인터페이스 규격이다. POSIX의 마지막 글자 X는 유닉스 호환 운영체제에 보통 X가 붙는 것에서 유래한다.



Let's Dive in- pth_hello.c

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS     5

void *PrintHello(void *threadid)
{
   long tid;
   long* tidPtr = (long*) threaded;
   tid = *tidPtr;
   printf("Hello World! It's me, thread #%ld!\n", tid);
   pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
   pthread_t threads[NUM_THREADS];
   int rc;
   long t;
   for(t=0;t<NUM_THREADS;t++){
     printf("In main: creating thread %ld\n", t);
     rc = pthread_create(&threads[t], NULL, PrintHello, (void *)&t);
     if (rc){
       printf("ERROR; return code from pthread_create() is %d\n", rc);
       exit(-1);
       }
     }

   /* Last thing that main() should do */
   pthread_exit(NULL);
}
```

210.115.230.177

Output

```
jeong-gun@jeonggun-desktop:~/PTHREAD$ ./pth_hello
In main: creating thread 0
In main: creating thread 1
Hello World! It's me, thread #1!
In main: creating thread 2
Hello World! It's me, thread #2!
In main: creating thread 3
Hello World! It's me, thread #3!
In main: creating thread 4
Hello World! It's me, thread #4!
Hello World! It's me, thread #5!
```

Some disorder is possible

Shared Threads Models

- **OpenMP**
  - Compiler directive based; can use serial code
  - Portable / multi-platform, including Unix and Windows platforms
  - Available in C/C++ and Fortran implementations
  - Can be very easy and simple to use - provides for "**incremental parallelism**"

Shared Threads Models

- **OpenMP**
  - Compiler directive based; can use serial code

Sequential Program

```
void main()
{
  int i, k, N=1000;
  double A[N], B[N], C[N];
  for (i=0; i<N; i++) {
    A[i] = B[i] + k*C[i]
  }
}
```

Parallel Program

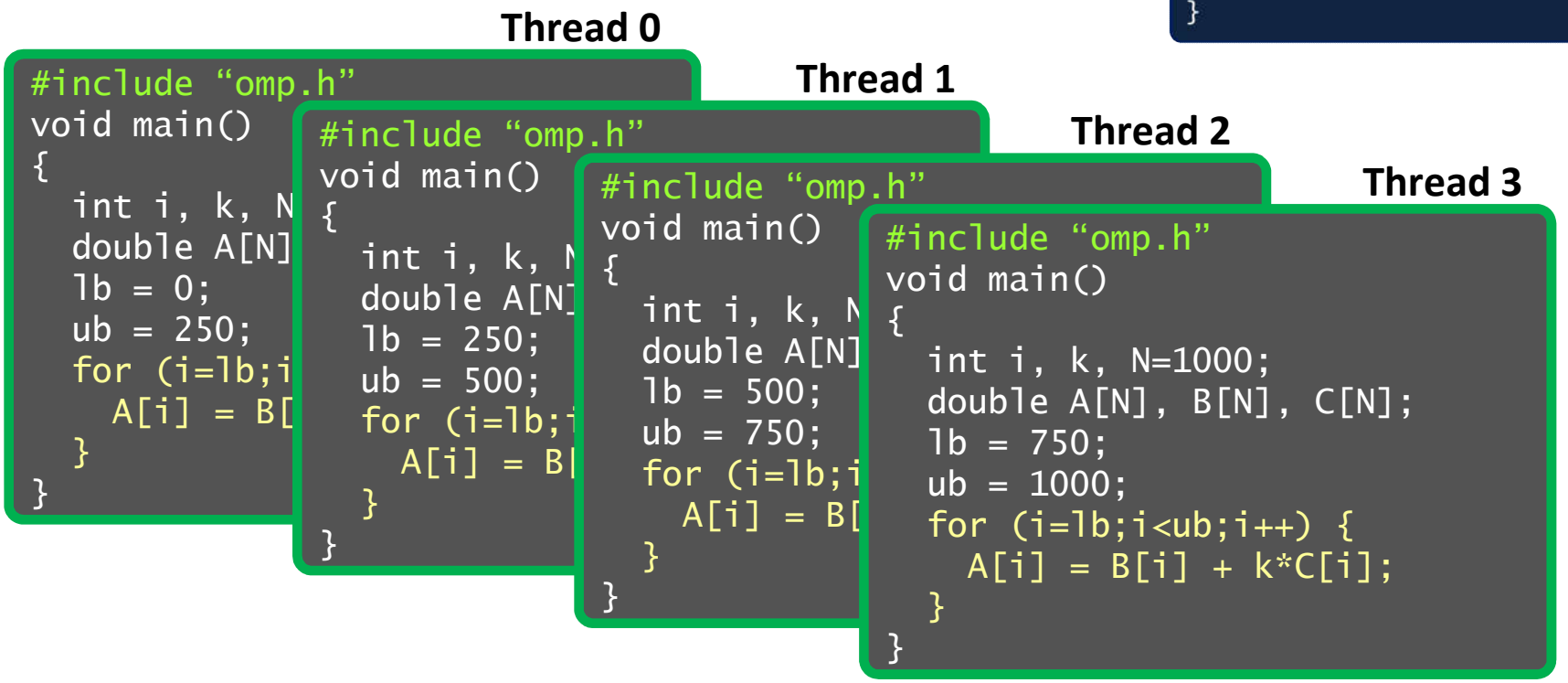
```
#include "omp.h"
void main()
{
  int i, k, N=1000;
  double A[N], B[N], C[N];
#pragma omp parallel for
  for (i=0; i<N; i++) {
    A[i] = B[i] + k*C[i];
  }
}
```

Shared Threads Models

- **OpenMP**
  - Single Program Multiple Data (SPMD)

Parallel Program

```
#include "omp.h"
void main()
{
  int i, k, N=1000;
  double A[N], B[N], C[N];
#pragma omp parallel for
  for (i=0; i<N; i++) {
    A[i] = B[i] + k*C[i];
  }
}
```



OpenMP Fork-and-Join model

```
printf("program begin\n");
N = 1000;

#pragma omp parallel for
for (i=0; i<N; i++)
    A[i] = B[i] + C[i];

M = 500;

#pragma omp parallel for
for (j=0; j<M; j++)
    p[j] = q[j] - r[j];

printf("program done\n");
```

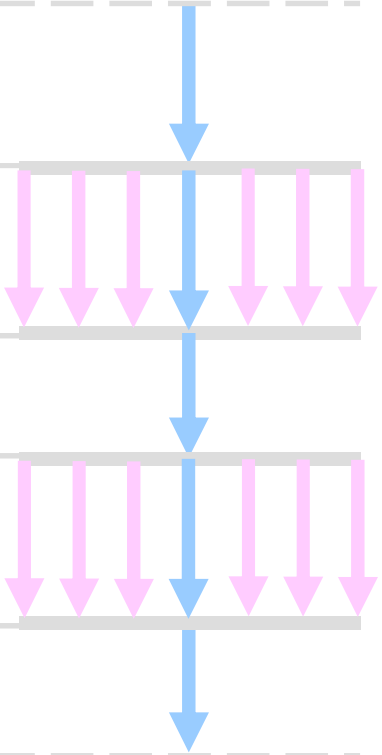
Serial

Parallel

Serial

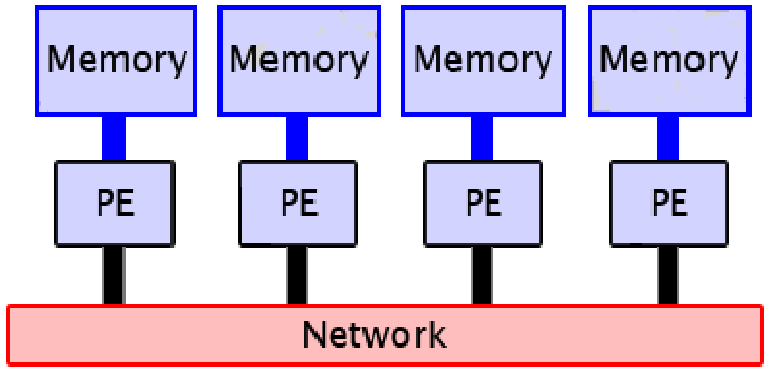
Parallel

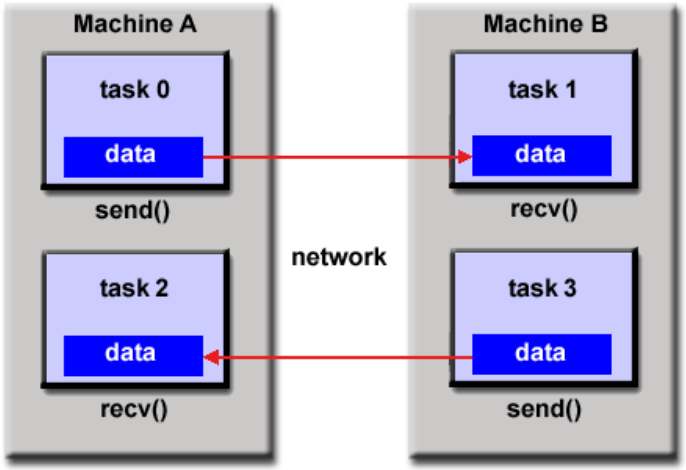
Serial



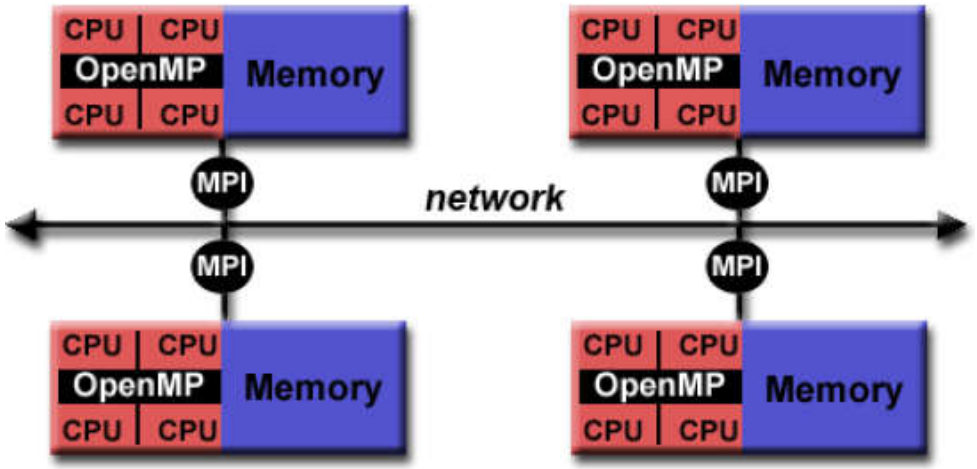
Distributed Memory / Message Passing Models

- Multiple tasks can reside on the same physical machine and/or across an arbitrary number of machines
- Tasks exchange data through communications by sending and receiving messages



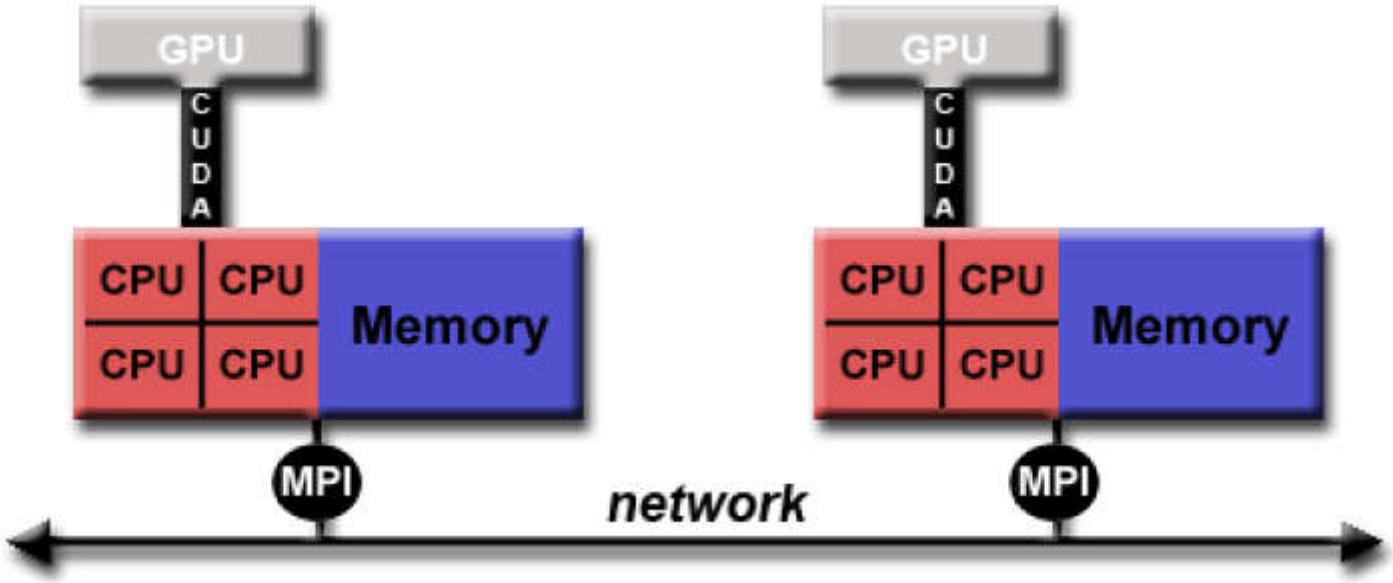


Hybrid Parallel Programming Models



- Currently, a common example of a hybrid model is the combination of the message passing model (MPI) with the threads model (OpenMP)
  - Threads perform computationally intensive kernels using local, on-node data
  - Communications between processes on different nodes occurs over the network using MPI
- This hybrid model lends itself well to the increasingly common hardware environment of clustered multi/many-core machines

Hybrid Parallel Programming Models



- Another similar and increasingly popular example of a hybrid model is using MPI with GPU (Graphics Processing Unit) programming
  - GPUs perform computationally intensive kernels using local, on-node data
  - Communications between processes on different nodes occurs over the network using MPI

Can my code be parallelized?

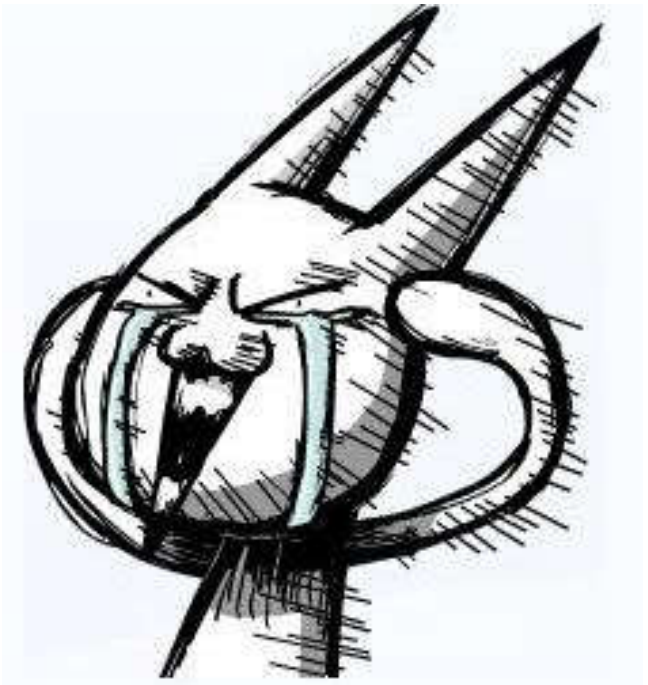
Interesting Quotes about Parallel Programming

1. "There are 3 rules to follow when parallelizing large codes. Unfortunately, no one knows what these rules are." ~ W. Somerset Maugham, Gary Montry

2. "The wall is there. We probably won't have any more products without multicore processors [but] we see a lot of problems in parallel programming." ~ Alex Bachmutsky

3. "We can solve [the software crisis in parallel computing], but only if we work from the algorithm down to the hardware — not the traditional hardware-first mentality." ~ Tim Mattson

4. "[The processor industry is adding] more and more cores, but nobody knows how to program those things. I mean, two, yeah; four, not really; eight, forget it." ~ Steve Jobs



Can my code be parallelized?

- Does it have large loops that repeat the same operations?
- Does your code do multiple tasks that are not dependent on one another? If not, is the dependency weak?
- Is the amount of communications small?
- Do multiple tasks depend on the same data?
- Does the order of operations matter?