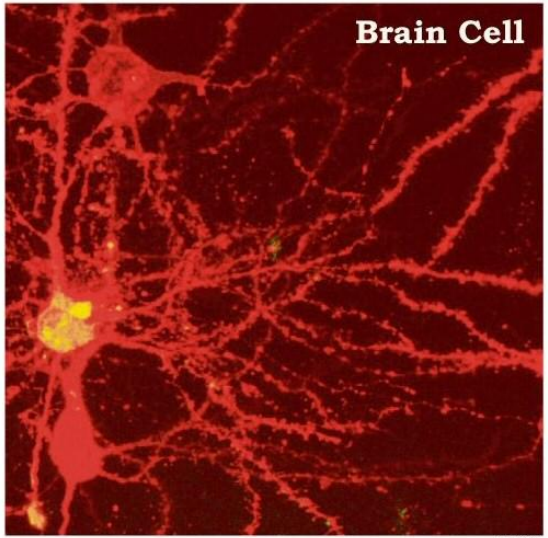
Processor

Micro Processor

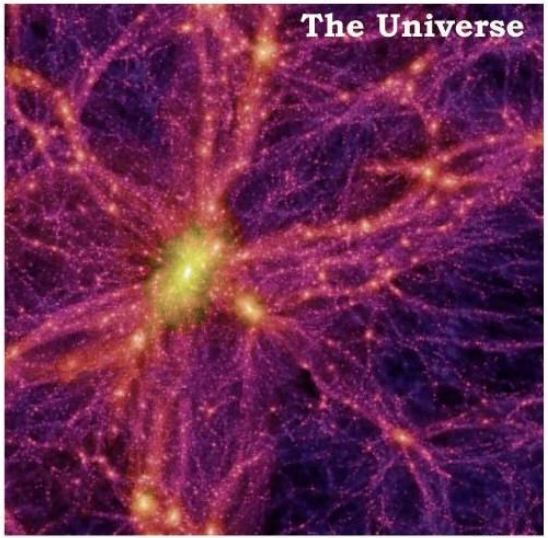
뭐 이런것 까지 알아야 하나?

One is only micrometers wide. The other is billions of light-years across. One shows neurons in a mouse brain. The other is a simulated image of the universe. Together they suggest the surprisingly similar patterns found in vastly different natural phenomena. *DAVID CONSTANTINE*



Mark Miller

Mark Miller, a doctoral student at Brandeis University, is researching how particular types of neurons in the brain are connected to one another. By staining thin slices of a mouse's brain, he can identify the connections visually. The image above shows three neuron cells on the left (two red and one yellow) and their connections.



Virgo Consortium

An international group of astrophysicists used a computer simulation last year to recreate how the universe grew and evolved. The simulation image above is a snapshot of the present universe that features a large cluster of galaxies (bright yellow) surrounded by thousands of stars, galaxies and dark matter (web).

Source: Mark Miller, Brandeis University; Virgo Consortium for Cosmological Supercomputer Simulations; www.visualcomplexity.com

The New York Times

If you enjoy mind-boggling science stories like this, visit **Pickover.Com**

굳이 이유를 들자면

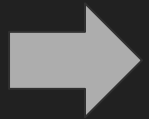
1. 대상은 다르지만 풀려고 하는 문제는 같다
2. The free lunch is over
3. Welcome to the jungle

프로세서를 이해하기 위해 몇 가지

Instruction Set Architecture

CISC vs RISC ~~이제 그만 하자~~

```
struct Vector
{
    int x;
    int y;
    int z;
} data[10];
data[index].y = Ox87654321;
```



;; ARM (좌로부터 문번호; 주소, 기계어 인코딩 값, 기계어, 해석)

```
0001100C ES9D1000 ldr rl, [sp, #4]   ; rl = index
00011010 E3A0300C mov r3, #0xc       ; r3 = sizeof(Vector)
00011014 E0020391 mul r2, rl, r3     ; r2 = index * sizeof(Vector)
00011018 E28D3004 add r3, sp, #8     ; r3 = data
0001101C E0833002 add r3, r3, r2     ; r3 = $data[index]
00011020 E2832004 add r2, r3, #4     ; r2 = $(data[index].y)
00011024 E59F3008 ldr r3, [pc, #8]   ; r3 = 0xOx87654321
00011028 E5823000 str r3. [r2]       ; *(r2) = r3
```

;; x86-64 (문번호: 주소, 기계어 인코팅 값, 기계어)

```
13FA110C8 48 63 44 24 24     movsxd      rax, dword ptr [rsp+24h]
13FA110CD 48 6B CO OC        imul        rax, rax, OCh
13FA110Dl C7 44 04 34 21     mov         dword ptr [rsp+rax+34h], 8787654321h
```

그 외

- uarch
- ALU
- Register
  - x86 8~16개 라고들…
  - 100 개 이상
- Cache
- RAM
- Virtual memory
- 1 HZ란?
- Process & Thread

Dependency or Hazard

1. 데이터
    a. RAW (Read-After-Write)
        i. x = y + 1;
        ii. z = x * 2;
    b. WAR (Write-After-Read)
        i. z = x * 2;
        ii. x = y + 1;
    c. WAW (Write-After-Write)
        i. x = z * 2;
        ii. x = y + 1;
2. 메모리
3. 컨트롤

Instruction 5 steps

1. 명령어 인출(Instruction fetch, IF)
2. 명령어 해독(Instruction decode and register fetch, ID)
3. 피연산자 인출(Operand(s) Fetch, OF)
4. 명령어 실행(Instruction Execution, EX)
5. 결과 저장(Operand Store, OS또는 Write Back, WB)

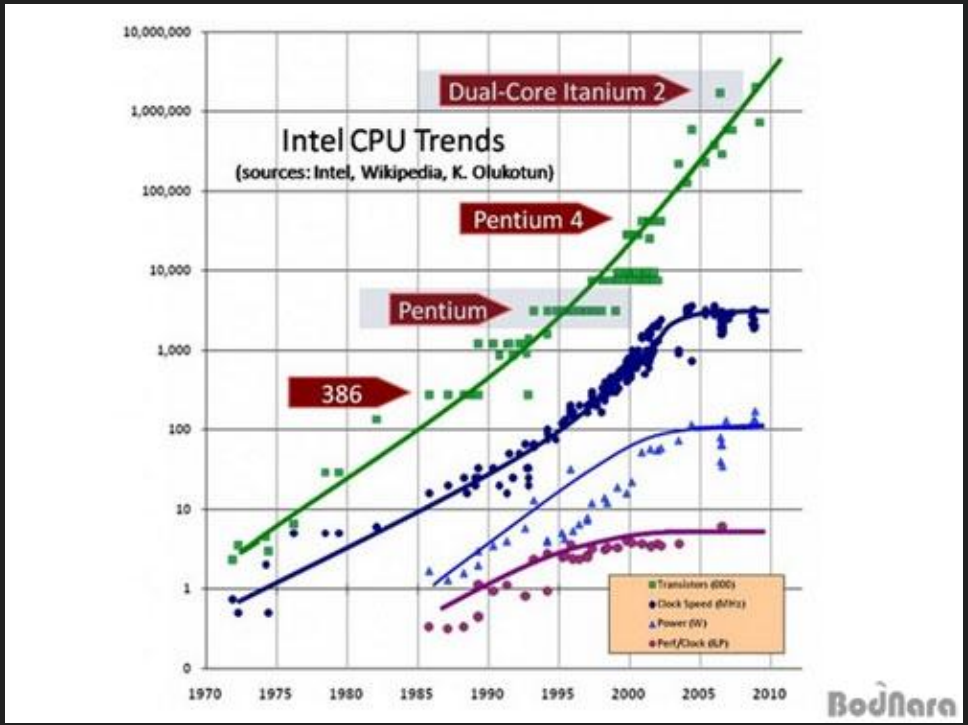
프로세서 발전의 두 축?

1. 반도체 기술 향상
2. 아키텍쳐 향상

무어의 법칙

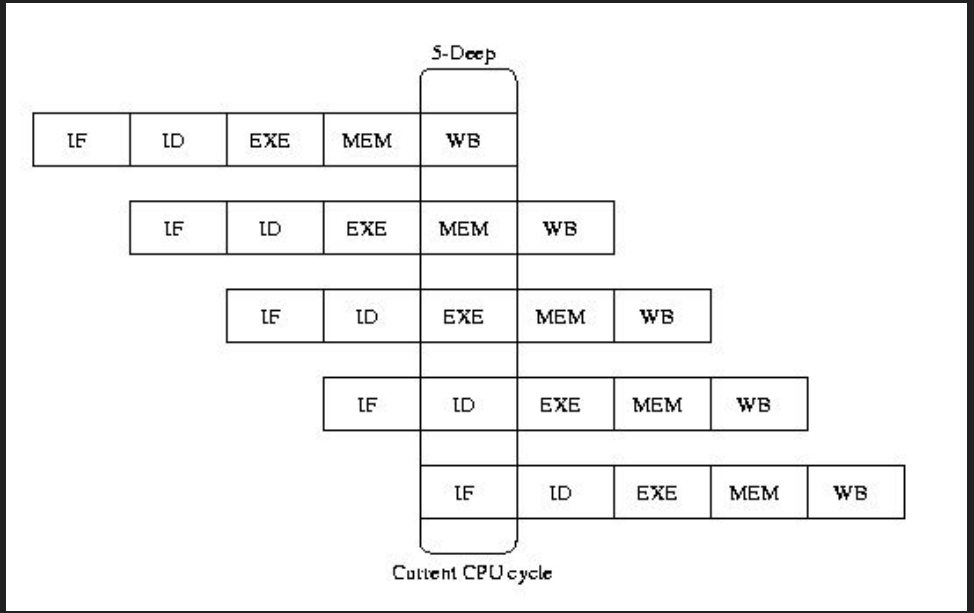
"18개월 혹은 24개월마다 저렴한 비용에 집적될 수 있는 트랜지스터의 개수가 두 배씩 증가"

- 1978년 인텔 8086
  - 29,000개. 5MHz
- 2008 Core i7
  - 731,000,000개. 2,930MHz
- 좀 지겹다



Pipeline

- Instruction수준의 병렬 (ILP)
- Throughput 개선
  - Latency는 개선 못함
- 최초는 1960년대 IBM 7030
- 1980년대 RISC 주요 구현 방식
  - 그럼 CISC는?
- x86 은 1989년 i486 최초
  - 286, 386도 어느 정도는 구현됨

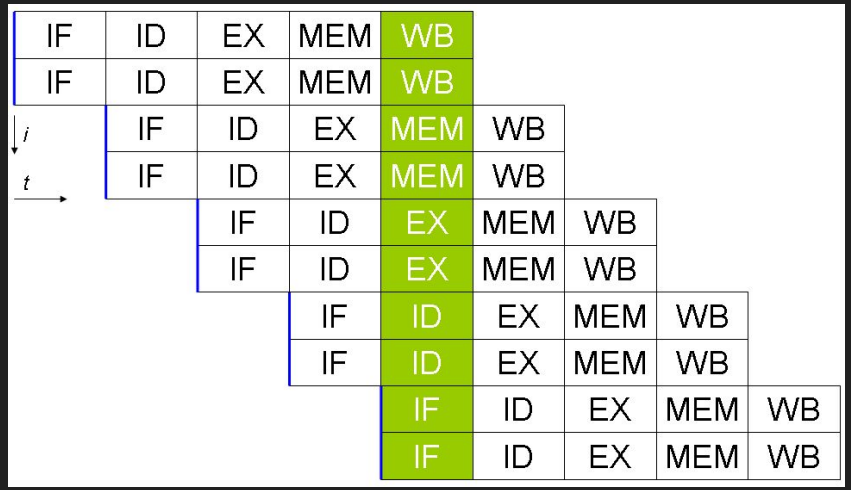


비순차 실행

- In-order VS Out-of-order
- ILP 찾기
  - dependency 조사가 필수
  - 명령어 윈도우
    - 컴파일러는 수천개의 ILP
    - 프로세서에서는 백여개
- 비순차도 좋은 알고리즘이 필요하다
- 명령어 완료의 순차성 여부는 관리한다.
  - ROB(Re-Order Buffer)
- 1964년 CDC 6600 메인프레임에서 시작
  - 마이크로 프로세서는 1990 IBM Power1
- 현대의 모든 프로세서가 비순차는 아니다.
  - Intel Itanium, Atom. IBM Power6

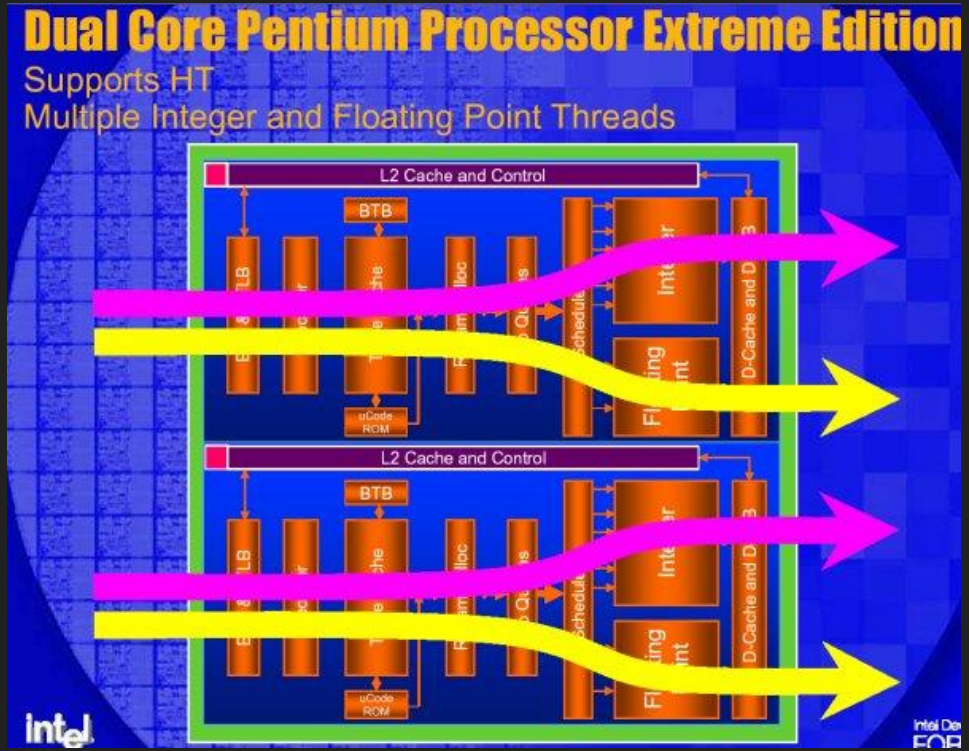
Superscalar

- 하나 이상의 파이프 라인
- N-wide, N-way 라고 표현
- 최초의 펜티엄은 아주 일부라인만 두개
- 인텔 네할렘은 4-wide까지 확장
- 비순차와 슈퍼스칼라는 보통 같이 간다.
  - 하지만 인텔 Atom 2-Wide 슈퍼스칼라 순차



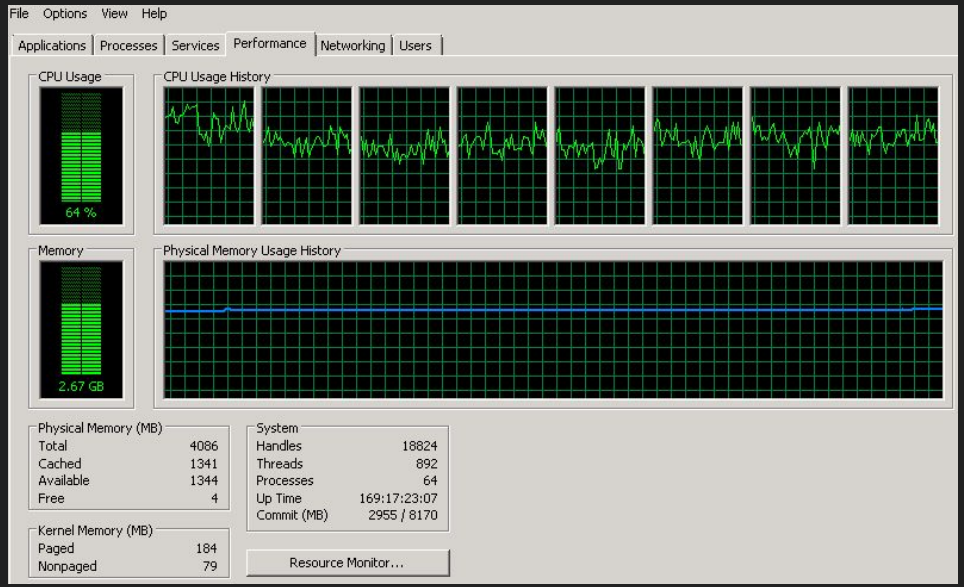
SMT(Simultaneous Multi-Threading)

- 스레드 문맥을 하드웨어 수준에서 복제
- 둘 이상의 논리 프로세서로 보이게 하는 기술
- 스레드 수준 병렬
- 전체 면적중 5~6% 이하가 더 필요
- 파이프라인이나 캐시는 공유



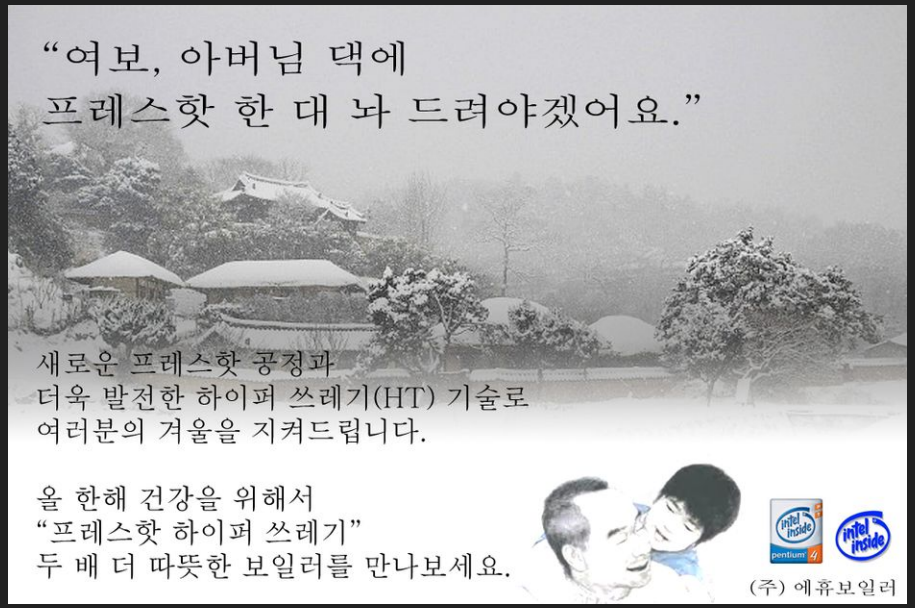
하이퍼 스레딩

- SMT의 마케팅 네임
- 펜티엄4 이후
- 2배향상은 아니지만 작업종류에 따라 7~34% 까지 성능향상
  - 물론 안좋은 작업도 있을 수도...
- OS의 스케쥴링도 중요
  - 현 시점(2017)에서는 거의 문제되지 않음



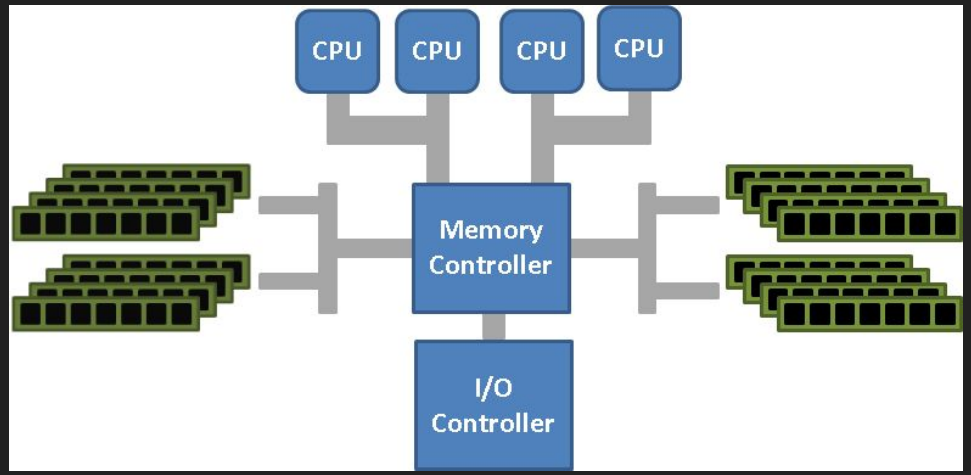
싱글코어의 한계

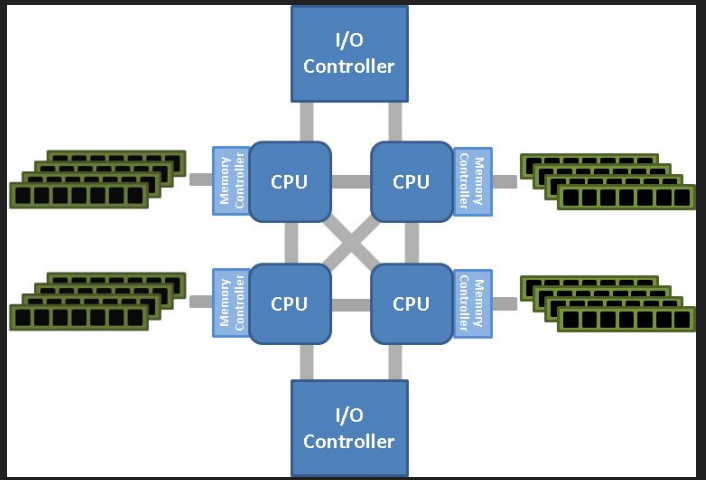
- 한때는 클럭 = 성능
- 클럭수는 가장 좋은 마케팅 수단
- 에너지 장벽(energy wall)
  - 아뜨! 아뜨!
  - 이 추세면 2015년엔 태양표면온도 까지 도달 (2001. ISSCC 인텔 부사장)
- 최적화 기법도 한계



멀티코어 병렬컴퓨터

- Homogeneous core
- TLP 수준 병렬 프로그래밍
- 메모리 구조 변화
    - SMP vs NUMA (Non-Uniform Memory Access)

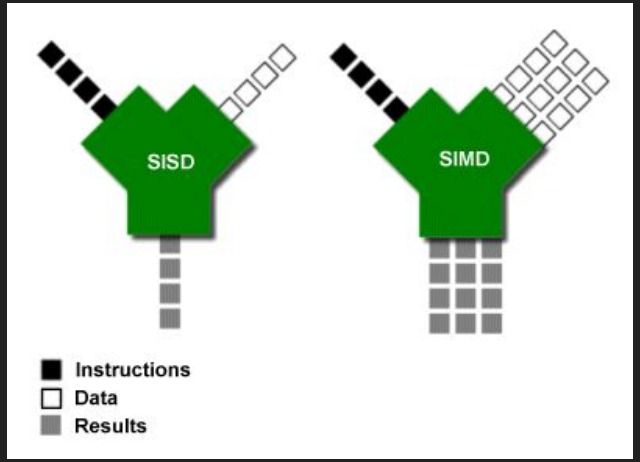




GPU

- 데이터 수준의 병렬성(Data-level parallelism)
- SIMD
  - Single Instruction Multiple Data
- GPU 에서 GPGPU
  - General Purpose Graphic Processing Unit
- 순차 처리
- 수천, 수만개의 스레드 문맥 관리
- 메모리와 대역폭도 아주 높다
- Heterogeneous core도 가능
- 슈~~~~퍼 컴퓨터

```
1.    for (int i = 0; i < 4; ++i)
2.            c[i] = a[i] + b[i];
```



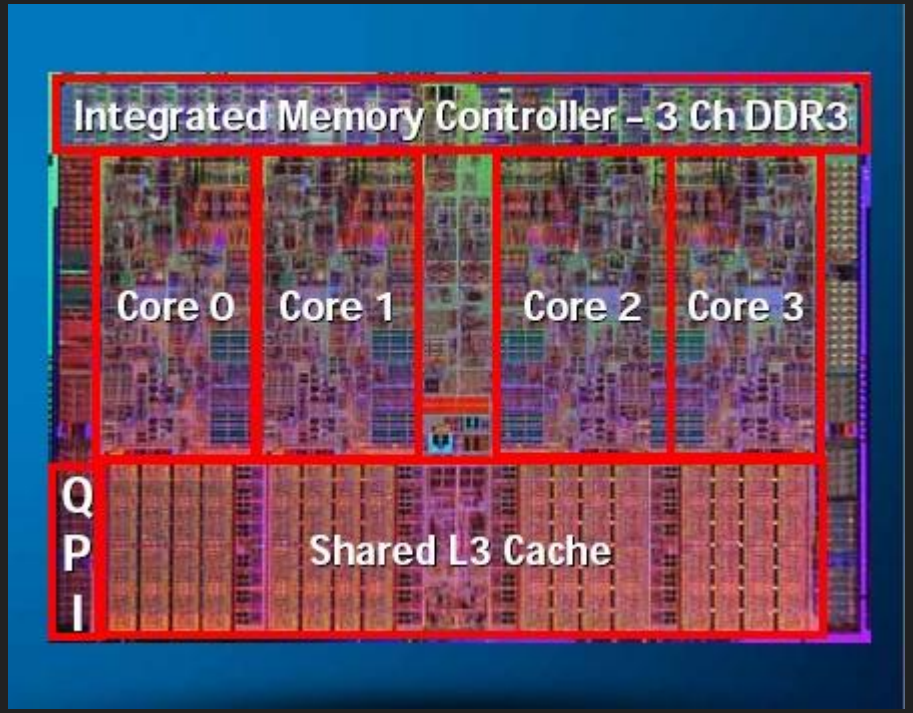
Differences between CPU and GPU

GPU 컴퓨팅 파워


ARM or x86 CPU
Optimized for Few
Serial Tasks
+
GPU Accelerator
Optimized for Many
Parallel Tasks
146X
의학
36X
분자동력학
18X
비디오 변환
50X
매트랩
100X
천체물리학
149X
재무시뮬레이션
47X
선형대수
20X
3D 초음파
130X
양자화학
30X
유전자 배열

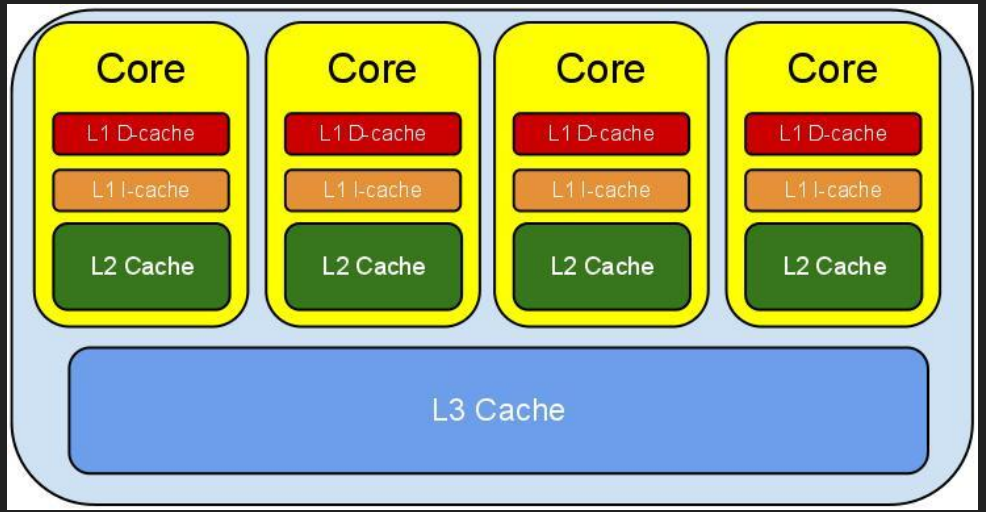
캐시(Cache)

- 속도 차이가 나는 두 계층 사이
  - Pentium 100MHz
    DRAM 66~100MHz (1996)
  - 프로세서 2~3GHz
    메모리 533~800MHz (2009)
- 데이터의 지역성(locality)을 활용
  - 시간적 (temporal)
  - 공간적 (spatial)
- 용어
  - cache hit/miss
  - miss penalty, hit latency
- 캐시 설계 고려 요소
  - 검색, 배치, 교체, 쓰기

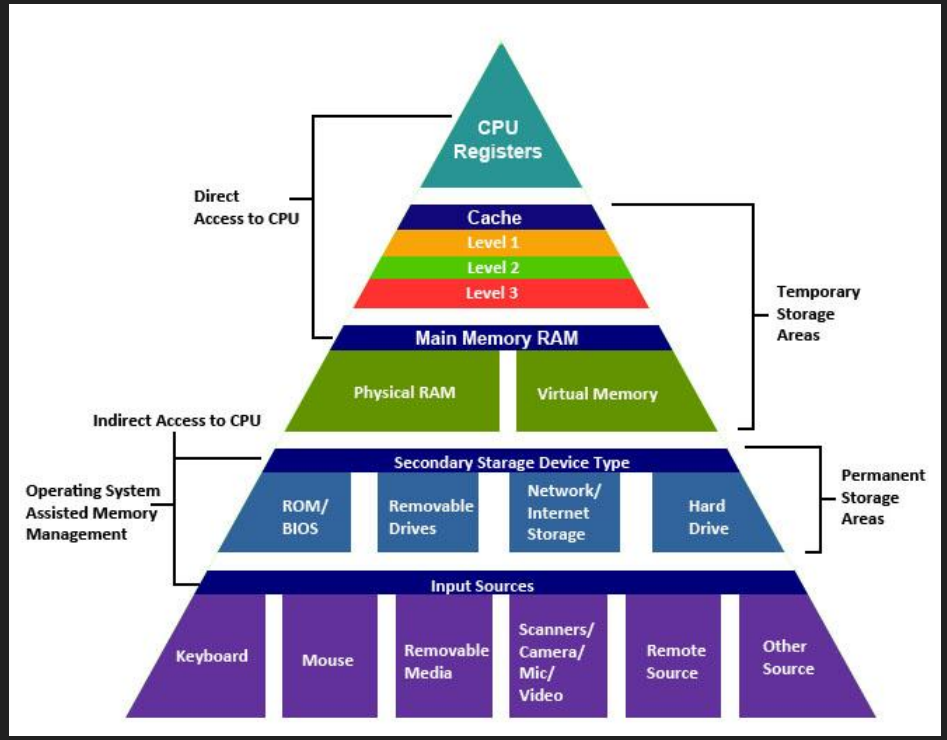


CPU multicore $

- 멀티레벨 구조 L1, L2, L3 (LL)
  - 2008년 nehalem 구조 L1-64KB, L2 or L3 - 8MB, 12MB
  - 왜 이렇게 나뉘어져 있을까?
- Cache line 혹은 cache block
  - 32 ~ 256 byte
- Cache coherenece
  - MSI MESI 프로토콜
  - Write back/through



Memory hierarchy



Latency comparison numbers

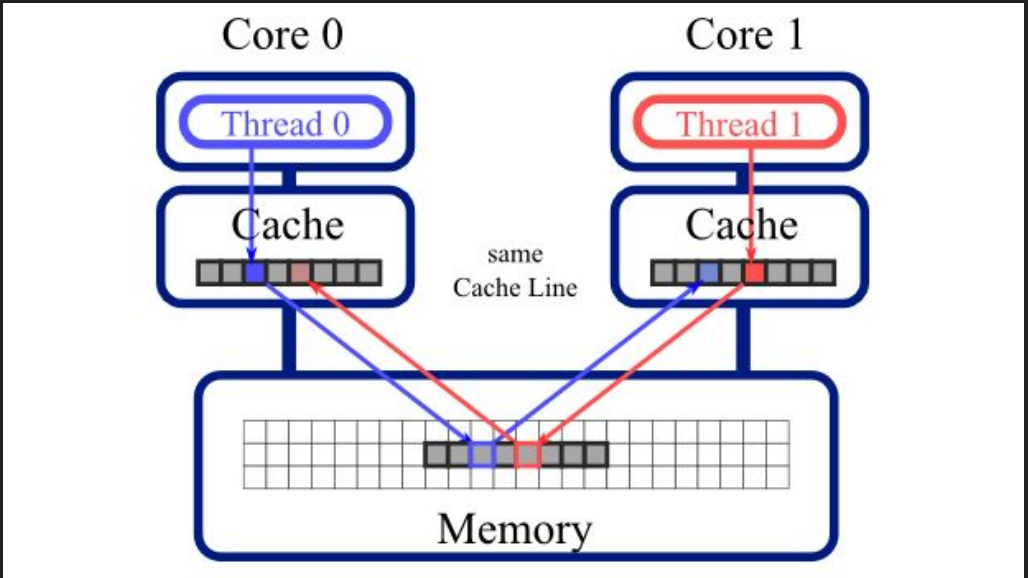
L1 cache reference	0.5	ns			
Branch mispredict	5	ns			
L2 cache reference	7	ns			14x L1 cache
Mutex lock/unlock	25	ns			
Main memory reference	100	ns			20x L2 cache, 200x L1 cache
Compress 1K bytes with Zippy	3,000	ns	3 us		
Send 1K bytes over 1 Gbps network	10,000	ns	10 us		
Read 4K randomly from SSD*	150,000	ns	150 us		~1GB/sec SSD
Read 1 MB sequentially from memory	250,000	ns	250 us		
Round trip within same datacenter	500,000	ns	500 us		
Read 1 MB sequentially from SSD*	1,000,000	ns	1,000 us	1 ms	~1GB/sec SSD, 4X memory
Disk seek	10,000,000	ns	10,000 us	10 ms	20x datacenter roundtrip
Read 1 MB sequentially from disk	20,000,000	ns	20,000 us	20 ms	80x memory, 20X SSD
Send packet CA->Netherlands->CA	150,000,000	ns	150,000 us	150 ms	

https://gist.github.com/jboner/2841832

False sharing

- 팀킬이다!
- 같은 캐시 라인
- 공유캐시가 없으면 200% 까지도 성능저하

```
1.  volatile int data1;
2.  volatile int data2;
```



Branch prediction

- 분기예측
  - 과거 실행에 기반
  - 만약, 브라우저에서 파일을 한번 저장한다면?
- 투기적 실행(Speculative execution)
  - 이길이 아니라면?
  - 백업장치가 필수
- 예측이 맞으면 …
- 아래의 경우라면…..

```
1: if (rand() % 2)
2:    a = 10;
3: else
4:    a = 20;
```

```
if (unlikely(fd < 0))
{
    /* Do something */
}
```

or

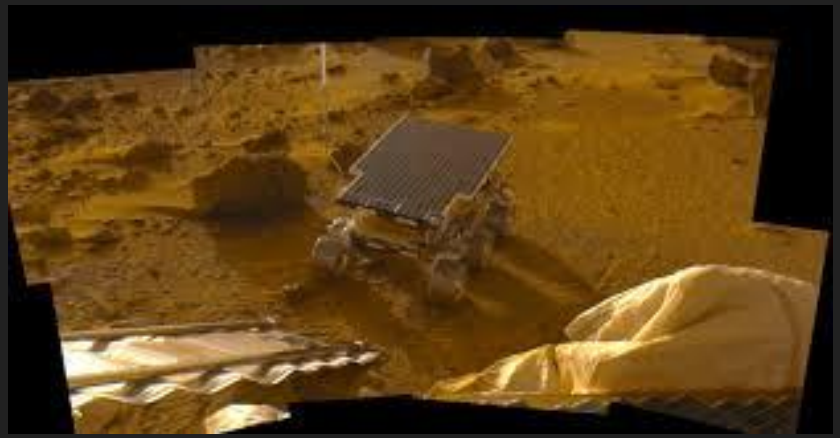
```
if (likely(!err))
{
    /* Do something */
}
```

Prefetcher

- 이번엔 미리 가져와 읽는다.
- Latency 개선
- 잘못 읽으면
  - 자원 낭비 (대역폭, 전력 등)
  - 캐시오염(cache pollution)

병렬 프로그래밍

- 여전히 싱글코어의 속도가 중요한건 맞다
- 하지만 하드웨어는 병렬성을 증대하고 있다
- 그런데 어렵다.
  - 데드락(dead lock)
  - 우선순위 역전현상(priority inversion)
  - 락 콘보잉(lock convoying) - many cores



References

- 프로그래머가 몰랐던 멀티코어 CPU 이야기
- 프로세서를 지탱하는 기술
- The Free Lunch Is Over
- Welcome to the jungle
- CPU·GPU가 뭐야?…인공지능 놓고 칩들의 '두뇌' 싸움
- 수억 킬로미터 밖에서의 디버깅 (2)