CA 1장

하드웨어/소프트웨어가 성능에 미치는 영향

알고리즘 –> 소스 프로그램 문자수와 입출력 작업 수를 결정

프로그램밍 언어, 컴파일러, 컴퓨터 구조 –> 각 소스 프로그램 문장에 해당하는 기계어 명령어 수 결정

프로세서와 메모리 시스템 -> 명령어의 실행 속도 결정

입출력 시스템(HW/OS) – 입출력 작업의 실행 속도 결정

6 Great Ideas in Computer Architecture

1. Layers of Representation/Interpretation
2. Moore's Law
3. Principle of Locality/Memory Hierarchy
4. Parallelism
5. Performance Measurement & Improvement
6. Dependability via Redundancy(여유분을 이용한 신용도 개선)

시스템 소프트웨어

- 어플리케이션 소프트웨어와 하드웨어를 연결해줌
- OS, 컴파일러, 로더 어셈블러가 속함
- OS : 기본적 입출력 작업의 처리, 보조기억장치 및 메모리 할당, 여러 응용들 간의 컴퓨터 공유방법 제공
- 컴파일러 : 상위 수준 언어를 하드웨어 명령어로 번역
- 어셈블러 : 어셈블리 언어를 이진수로 바꿈

컴퓨터의 고전적 구성 요소 다섯가지 -> 입력, 출력, 메모리, datapath, control유닛

datapath 와 control은 processor라고 부르기도 한다.

1.5 프로세서와 메모리 생산 기술

트렌지스터 -> 전기로 제어되는 온/오프 스위치. 집적 회로는 수십, 수백 개의 트렌지스터를 칩 하나에 집적시킨 것이다. Moore가 자원수가 2배가 된다는 것은 칩 안에 집적되는 트렌지스터의 수가 그렇게 될 것이라고 예상, 조건에 따라 도체가 되기도 하고 절연체가 되기도 하는 물질(스위치)이다.

VLSI(초대규모집접회로) : 수백, 수백만개의 트렌지스터를 포함하는 장치

집적회로 제조과정 -> 실리콘 결정괴 -> 얇게 잘라서 웨이퍼 제작 -> 화확물질 첨가하여 트렌지스터, 도체, 절연체로 바꿈

Wafer 에는 **defect**(결함)때문에 완벽하게 만들기 어렵다.

불완전성 대처하기 위해 한 웨이퍼에 독립적인 컴포넌트를 여러 개 만든다. 그런 다음 웨이퍼를 컴포넌트 별로 자르는데 이것을 **die** 또는 **칩**이라고 함

Yield : 웨이퍼 상의 전체 다이 중 정상 다이의 비율, PE Area(Size)값이 높아지면 die값이 높아지고, yield값은 작아진다.

$$\text{Cost per die} = \frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{Yield}}$$

$$\text{Dies per wafer} \approx \text{Wafer area} / \text{Die area}$$

$$\text{Yield} = \frac{1}{(1 + (\text{Defects per area} \times \text{Die area}/2))^2}$$

면적 및 결함 비율에 대한 비선형 관계

1. **Wafer cost and area are fixed**
2. **제조과정에서 정해지는 Defect rate**
3. **설계 및 회로 설계에서 결정되는 Die area**

1.6 Performance

Latency(response time or execution time) -> 컴퓨터가 태스크를 완료하기까지의 총 소유시간

L= ***람다 W =>*** *average interarrival rate (l) x average service time (W)*

Bandwidth(throughput) -> 단위시간당 완료하는 태스크 수

프로세서를 더 빠른 버전으로 바꾼다 -> 처리량과 응답시간 모두 개선

Faster verion? 질문 -> faster version뜻은 clock의 속도가 점점 높아지는것, 더많은 트렌지스터가 들어간것, 프로세스를 병렬계로 써서 구성

Adding more processors -> 특정 태스크의 실행시간이 단축되는 것이 아니므로 처리량만 개선, 하지만 처리에 대한 요구가 처리량보다 커지면 일부는 큐에서 기다릴것이다. 따라서 이럴 경우에는 처리량이 많아지면 응답시간 역시 줄어든다.

경과시간(elapse time) -> 한 작업을 끝내는 데 필요한 전체 시간, 디스크 접근, 메모리 접근, 입출력 작업, 운영체제 오버헤드 등 모든시간을 다 던한것이다, 시스템성능에 영향끼침.

CPU execution time -> 특정 작업의 실행을 위해 CPU가 소모하는 실제 시간, 입출력에 걸린 시간이나 다른 프로그램을 실행하는데 걸린시간 포함x, but 사용자가 느끼는 응답시간은 CPU시간이 아니고 경과시간(elapse time)이다.

성능 = 1/실행시간

User CPU time -> 프로그램 자체에 소비된 CPU시간

System CPU time -> 프로그램의 수행을 위해서 운영체제가 소비한 CPU시간

Elapse time으로 구분한 성능은 시스템성능, CPU시간으로 구분한 성능은 CPU성능이라고함

컴퓨터 설계자는 하드웨어가 기본 함수를 얼마나 빨리 처리할 수 있는지와 관련된 성능척도 필요

거의 모든 컴퓨터는 하드웨어 이벤트가 발생하는 시점을 결정하는 클락을 이용하여 만들어진다.

Clock cycle-> 클럭의 시간 간격, **Clock period** -> 한 클럭에 걸리는 시간,

Clock frequency(rate)-> 단위시간당 클럭수, 핸드폰 저전력 모드는 CPU clock cyle(cpu 클럭 사이클 수)를 느리게 하는 원리, Clock Cycle Time과 역수관계 , 단위도 외우자(예시 GHZ=10^9HZ),

$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$

$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

클럭사이클의 길이(주기)를 줄이거나 프로그램 실행에 필요한 클럭 사이클 수를 줄이면 성능 개선, 그러나 이 중 하나를 감소하면 다른하나가 증가하는 경우가 자주 발생

명령어 개수도 실행시간과 연관이 있다.

$$\text{Clock Cycles} = \text{Instruction Count} \times \text{Cycles per Instruction}$$

$$\text{CPU Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$$

$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

명령어당 클럭 사이클 수는 **CPI**로 줄여서 말한다. 명령어마다 실행시간이 다르므로 CPI는 프로그램이 실행한 모든 명령어데 대한 평균한 값을 사용한다.

$$\text{Clock Cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{Instruction Count}_i)$$

$$\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^{n} \left(\text{CPI}_i \times \frac{\text{Instruction Count}_i}{\text{Instruction Count}} \right)$$

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

진동수와 연관

컴퓨터 성능에 대한 완벽하고 믿을만한 척도가 실행시간밖에 없다. 예를 들어 명령어 개수를 줄이기 위해 명령어 집합을 바꾸었을 때 클럭 속도가 느려지거나 CPI가 커져서 오히려 성능이 더 나빠질 수 있다.

프로그램 성능에 영향가는 것 – 알고리즘, 프로그래밍 언어, 컴파일러, 명령어 집합 구조에 영향

- ⇨ **Algorithm :** IC(명령어 수), CPI, 알고리즘이 빠른(느린) 명령어를 선호하냐 따라 CPI에 영향 미칠 수 있음.

⇨ **Programming language :** IC, CPI

⇨ **Compiler :** IC, CPI

⇨ **Instruction set architecture :** IC, CPI, Tc(프로세서의 클럭 속도)

1.7 전력장벽

Power Trends -> 클럭속도와 소비 전력은 서로 연관. 둘이 같이 증가.

CMOS -> 에너지가 소비하는 주원인은 동적 에너지. 동적에너리란 트렌지스터가 0에서 1로 혹은 그 반대로 스위칭하는 동안에 소비되는 에너지를 말한다.

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

⇨ power(전력)가 30배 증가하는 동안 클럭속도가 어떻게 1000배가 빨라질 수 있는가? 새로운 공정기술이 나올 때마다 전압은 낮아졌고, 전력은 전압의 제곱에 비례하기 때문에 소비전력이 낮아질 수 있었다. 20년동안 전압은 5V에서 1V로 줄어 들었는데 이것이 전력이 고작 30배밖에 증가하지 않은 이유이다..

Reducing Power -> 전압을 더 이상 낮추면 트렌지스터 누설 전류가 너무 커진다. 칩을 냉각시키기 위해 더 비싼 방법을 사용하면 너무 비싸다. 동적 에너지가 CMOS 에너지 소모의 주원인인지만 트렌지스터가 꺼질 때 누설전류인 정적에너지 소모도 40%이다. 전력은 두가지 이유로 집적회로의 골치 아픈 문제다. 첫째 전력이 칩 전체에 전달되어야 한다. 둘째, 전력이 열로 낭비되는데 열을 제거해야 한다. 어떻게 해결할까? 멀티 프로세서

단일 프로세서의 성능 개선 둔화 이유 -> 전력의 제약, 가용한 명령어 수준 병렬성의 제약, 메모리 지연시간(Constrained by power, instruction-level parallelism, memory latency)

프로그래머가 명시적 병렬 프로그램 작성 어려운 3가지 이유

- 성능을 위해 프로그램밍 하기 어려움
- 부하를 공평하게 분배 -> Load Balancing
- 통신 및 동기화 오버헤드를 줄이기 위해 주의해야한다. -> optimizing Communication and synchronization

1.9 Intel Core i7 벤치마킹

작업부하 : 실행시키는 프로그램들의 집합, 두 컴퓨터 시스템 평가시 두 텀퓨터에서 같은 작업부하의 실행시간만 비교

벤치마크 : 성능을 측정하기 위해 선택된 프로그램들의 집합

SPEC CPU 벤치마크 -> 성능 측정을 위한 프로그램

SPEC CPU2006 – 프로그램 선택을 실행하는데 걸리는 시간, 기준 기계에 대한 표준화, 성능 비율의 기하 평균으로 요약

SPECCratio -> 기준 프로세서의 실행시간을 측정하려는 컴퓨터의 실행시간으로 나누어 정규화, 클수록 성능 더 좋음 , 밑에식의 기호는 다 곱하는 것이다.

$$\sqrt[n]{\prod_{i=1}^{n} \text{Execution time ratio}_i}$$

SPEC Power Banchmark 전력 측정 -> 일정 시간동안 작업부하를 10%식 증가시키면서 서버의 전력소모 측정

$$\text{Overall ssj\_ops per Watt} = \left(\sum_{i=0}^{10} \text{ssj\_ops}_i\right) \Big/ \left(\sum_{i=0}^{10} \text{power}_i\right)$$

$\text{Overall ssj\_ops per Watt}$: CPU 벤치마크에서와 같이 컴퓨터 마케팅을 간단히 하기위한 값 **성능** : ssj_ops/sec(처리량 :초당 수행하는 비즈니스 연산), **power(i)**: 각 성능 수준에서 소비되는 전력, **ssj_ops**(i) :작업부하를 10%씩 증가될 때마다의 성능,

1.10 오류 및 함정

함정: 컴퓨터의 한 부분만 개선하고 그 개선된 양에 비례해 전체 성능이 좋아지라고 기대하는 것

Amdahl's Law

$$T_{improved} = \frac{T_{affected}}{\text{improvement factor}} + T_{unaffected}$$ (improvement factor : 개선의 크기, T(affected) : 개선에 영향을 받는 실행 시간, T(unaffected) : 영향을 받지 않은 실행 시간)

오류 : 이용률이 낮은 컴퓨터는 전력 소모가 작다.

오류 : 성능에 초점을 둔 설계와 에너지 효율에 초점을 둔 설계는 서로 무관하다.

->에너지는 Power(전력)을 시간에 대해서 적분한 것으로, 어떤 하드웨어나 소프트웨어 최적화 기

술이 에너지를 더 소비하더라도 실행시간을줄여서 전체 에너지를 절약하기도 한다.

함정 : 성능식의 일부분으로 성능의 척도로 사용하는 것

-> 클럭속도, 명령어 개수, CPI에서 하나만 또는 두개만 사용하여 성능을 비교하는 것은 옳지 않다.

MIPS -> 명령어 실행 속도, 실행시간 대신에 쓸수 있는 척도, million instructions per second(프로그램의 실행 속도를 백만개의 명령어 단위로 나타내는 척도)

$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$
$$= \frac{\text{Instruction count}}{\frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}$$

-> 빠른 컴퓨터일수록 높은 MIPS값을 가짐

하지만 컴퓨터 성능을 비교하는 기준으로 MIPS를 사용하는 데는 세가지 문제가 있다.

->MIPS는 단순히 명령어를 실행하는 속도를 나타낼 뿐이지, 그 명령어 하나가 얼마나 많은 일을 수행하는지는 반영x

->같은 컴퓨터에서도 어떤 프로그램을 실행하느냐에 따라 MIPS값이 달라진다.

-> 많은 명령어를 실행하지만 빠른 명령어를 사용하는 프로그램으로 꾸미는 경우 컴퓨터 성능과는 반대로 MIPS값은 작아진다.(스스로점검하기 p54)

=> 실행 시간만이 흠잡을 데이 없이 유효한 성능 척도이다.