[강좌] CPU의 고속이라는 것. CPU 구조 -3부

CPU의 고속이라는 것. CPU 구조 -3부

여기까지 2회에 걸쳐서, 가정부의 도움을 빌리면서(?) CPU와 CPU의 주변이 어떻게 되어 있는지를 해설해 왔다. 이번에는, 굳이 가정부의 행차를 바라지 않고, 고속 CPU 에 대해 정리해 보자.

CPU 의 일을 재확인

그런데, 제 1회 와 제 2회를 읽어 주신 것을 전제로서 되돌아 보면, 요컨데 CPU의 일이라고 하는 것은, 데이터의 가공과 처리이다. 그것은 아래의 (1)~(3)이라 할 수 있다. CPU는「고속으로 되어 있다」라고 말할 수 있는 것이다.

(1) 데이터를 고속으로 읽어들인다
(2) 읽어들인 데이터를 고속으로 처리한다
(3) 처리가 끝난 데이터를 고속으로 써 되돌린다

더해 말하면, CPU는 처리를 실시하기 위해서는, 게임이나 Web 브라우저 등 무엇을 처리할 것인가 하는 「프로그램」이 필요하게 된다. 따라서, 아래의 2가지 점이 고속으로 처리할 수 있을지 CPU에 요구되고 있다.

(4) 프로그램을 고속으로 읽어들인다
(5) 읽어들인 프로그램(=명령)을 고속으로 해석한다

이 중, (1)(3)(4)는, 2장에서 해설한 것처럼, CPU 그것이라고 하는 것보다는 외적인 환경(=CPU 주변의 환경)에 의해서 정해지는 부분이다. 즉,「얼마나 메모리와 고속으로 데이터의 교환을 할까」가 중요하다. 이 때문에, CPU는 버스 속도를 올릴 방향으로 항상 진화해 왔다. 표 1은 주요 CPU의 버스에 대해 집계한 것으로, Pentium II시대에 66MHz 정도의 동작 클락이었던 버스는, 2005년 현재 10 배이상 고속으로 되어 있어 데이터를 1초간에 어느 정도 전송 할 수 있는지를 나타내는「대역폭」도, 역시10 배가 되고 있는 것을 알 수 있다.

製品	インターフェース形式	バス速度	バス帯域
Pentium Ⅱ	Slot1	66MHz→100MHz	528MB/s→800MB/s
Pentium Ⅲ	Slot1→Socket370	100→133MHz	800MB/s→1.06GB/s
Pentium 4	Socket423 →Socket478 →LGA775	400MHz →533MHz →800MHz	3.2GB/s →4.3GB/s →6.4GB/s
Athlon	SlotA→SocketA	200MHz→266MHz	1.6GB/s→2.1GB/s
Athlon XP	SocketA	266MHz →333MHz →400MHz	2.1GB/s →2.7GB/s →3.2GB/s
Athlon 64	Socket 754/Socket 939	800MHz→1GHz	6.4GB/s→8GB/s

표1

그러한 이유로, 순수하게 CPU"만"고속를 말하는 경우, 주목 해야 할 것은 (2)와 (5)가 된다. 거기서 우선은, (5)의「읽어들인 프로그램(=명령)을 고속으로 해석한다」에 대해서, 좀 더 파고 들어 보자.
CPU의 명령 처리 성능은, 일반적으로 아래의 식에서 구할 수 있다.

명령 처리 성능 = 1 사이클 사이에 처리할 수 있는 명령수×동작 클럭

Performance = IPC (Instructions Per Cycle) × Frequency

이것은 가정부……가 아니고, 자동차의 엔진으로 상상하면 알기 쉬울지도 모른다. 자동차에는, 트럭과 같이「무거운 짐을 옮기는 타입」이 있고, 레이싱 카와 같이「스피드를 내는 타입」이 있다. 또한, 최고 속도는 높지 않지만 무거운 짐을 실어도 어느 정도의 속도를 유지할 수 있는 타입(토르크형)과 무거운 짐을 실으면 성능이 떨어지지만 어느정도 스피드를 내기 쉬운 타입(고회전형)이 있다. 그래서 승용차는 이 중간에 위치한다. 승용차라고 하는 것은, 말해 보면 스피드가"적당히"에 낼 수 있고, 한편 짐도"적당히"쌓을 수 있다고 하는, 밸런스를 중시한 차종인 것이다.

그런데 성능 경쟁이 격렬해지면, 종래와 같은 발상으로 성능을 전체적으로 끌어올려 가는 것이 어려워지게 되며, 게다가 약간의 향상은 되려 어드밴티지가 없어지게 된다. 이와 반대로, 짐을 쌓는 것은 단호히 포기하는 대신에「대단히 스피드가 나옵니다」또는, 반대로 스피드는 제외하는 대신에「대단히 가득 짐을 쌓을 수 있습니다」라는 신축성을 가진 엔진이, 오히려 이전의 방식보다 더 많은 장점을 가지게 될 수 있을 것이다.

현재의 CPU는 확실히 이러한 상황에 노여 있다. CPU성능은 한계점 도달이 되어 있어, 1 사이클 당으로 처리할 수 있는 명령수(IPC)를 올리는가, 동작 클럭을 올리는가 하는 선택이 필요하게 되었던 것이다.

「그럼, 어느 쪽을 취할까?」이것이, 다음에 오는 문제이다.
IPC를 선택하면, 동시에 다수의 명령을 처리할 수 있는 구조가 필요하게 된다. 여기서 도움이 되는 것은, 앞선 1장에서 설명한 슈퍼 스칼라이다. 슈퍼 스칼라의 경우, 복수의 명령을 동시에 실행시킨다고 하는 방식으로,IPC(을)를 향상 시킨다.

그림 1 을 보자. 단순한 실행 유닛이 하나의 파이프라인(1 클럭 간에 실행할 수 있는 명령은 하나이므로,IPC =1)를, 실행 유닛을 세 개 가지는 슈퍼 스칼라로 변경하면, IPC는 최대 3이 된다. 게다가 잘 사용되는 정수 연산이나 로드 스토어를 한개씩 강화해 슈퍼 스칼라화 하면, IPC는 최대 5까지 끌어 올려지게 된다.

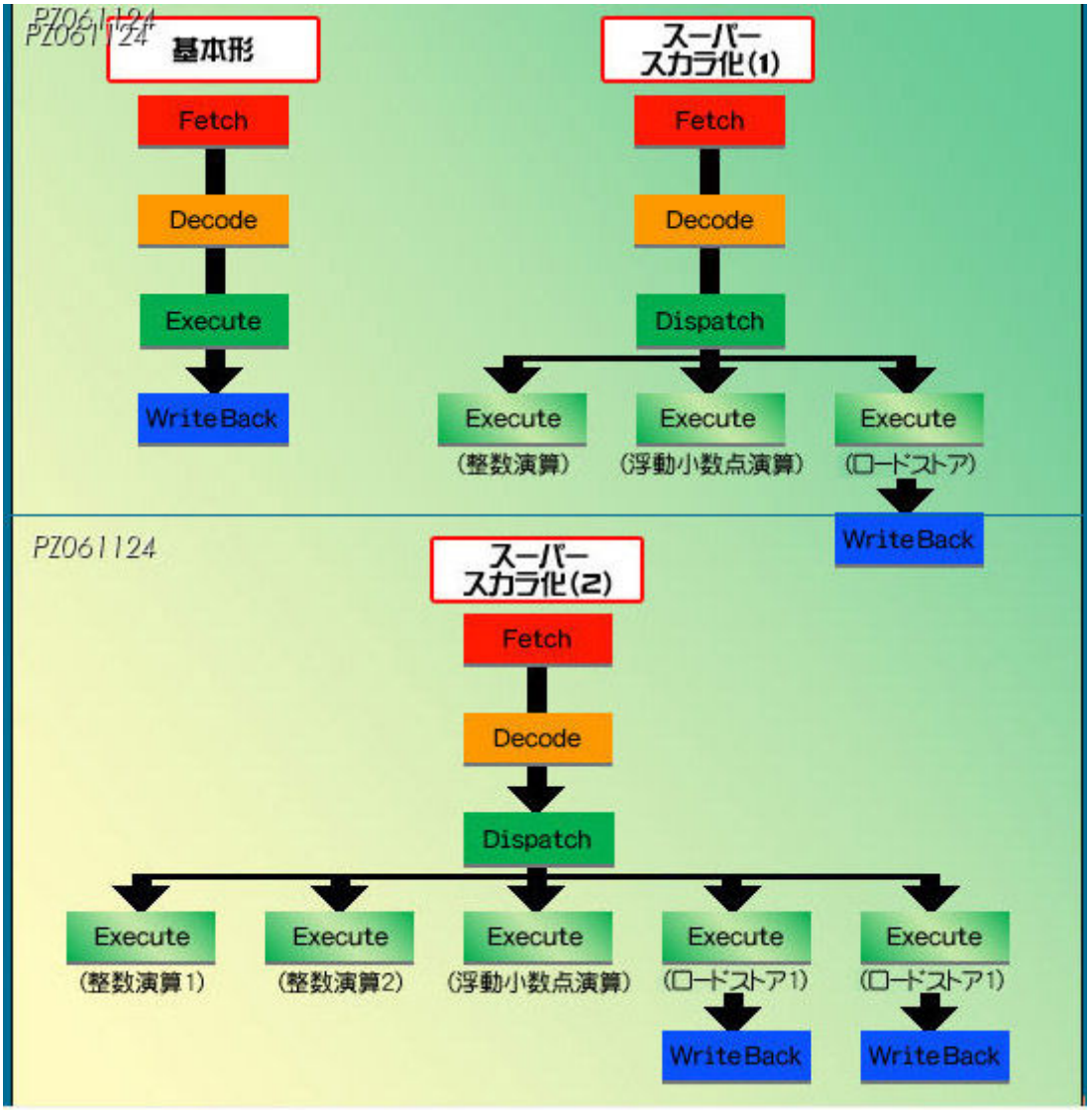


그림1

그런데, 여기서「최대3 」「최대5 」라고 한 것처럼,「최대」라고 하고 있는데는 이유가 있다. 이 최대치는 어디까지나 이상적으로 일이 진행되었을 경우의 값이며, 실제로는 좀처럼 거기까지 오르지 않는 것이다.

　예를 들면 그림 1 위오른쪽의 구성에서, IPC가 3이 되기 위해서는, 프로그램도 그림 2와 같이, 예쁘게 줄지어 있을 필요가 있다.

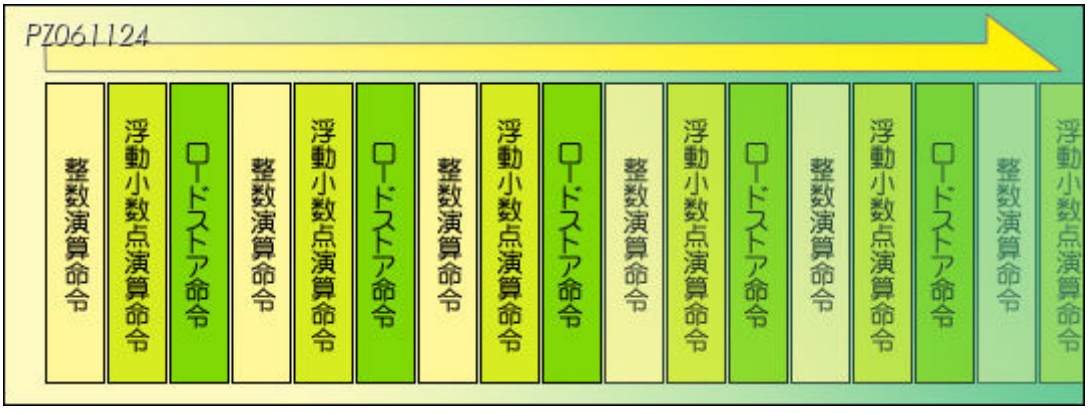


그림2

그러나, 실제로는(어디까지나 예이지만) 그림 3과 같은 이미지가 되어 있는 것이 보통이다.

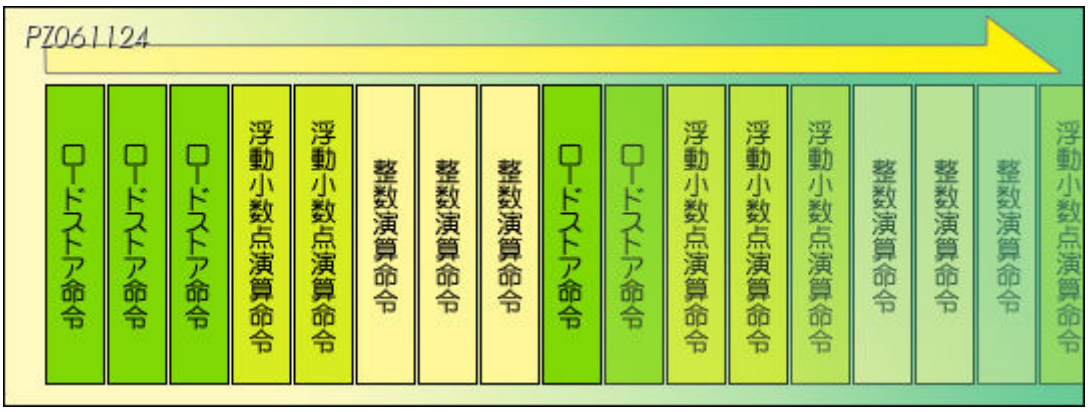


그림3

요컨데「모든 실행 유닛을 항상 이용할 수 있는 상태에 있다」라고는 되어 있지 않다.
IPC 중시로 진화하고 있는 대표적인 CPU는, AMD의 Athlon과 Athlon XP/64 이다. 특히, Athlon 64는 IPC 3 에 꽤 가까운 곳까지 끌어 올려지고 있다고 하지만, 이것은 어디까지 피크치 일때의 이야기이고, 통상1 ~1.5 이고, 조건이 좋다면 2 정도라 할 수 있다.

IPC를 한층 더 끌어올리기 위해, 2005년 가을 시점에서는 명령 변환과 아웃 오브 오더를 병용하는 것이 필수 조건이 되고 있다.

그림 4는 그 예로, 단지 스켈러를 탑재한 것 만으로는 명령의 유닛으로부터 다음의 유닛으로 차례로 흘러 가 실행될 뿐이므로, 실행 유닛을 나누어도 별로 의미가 없다. 그런데, 실행순서를 임의로 바꿀 수 있는 아웃 오브 오더를 채용하면, 그림 5와 같이, 13 사이클 걸려 있던 것이 6 사이클로 끝나는 등, 효율적으로 처리할 수 있다는 것이다.

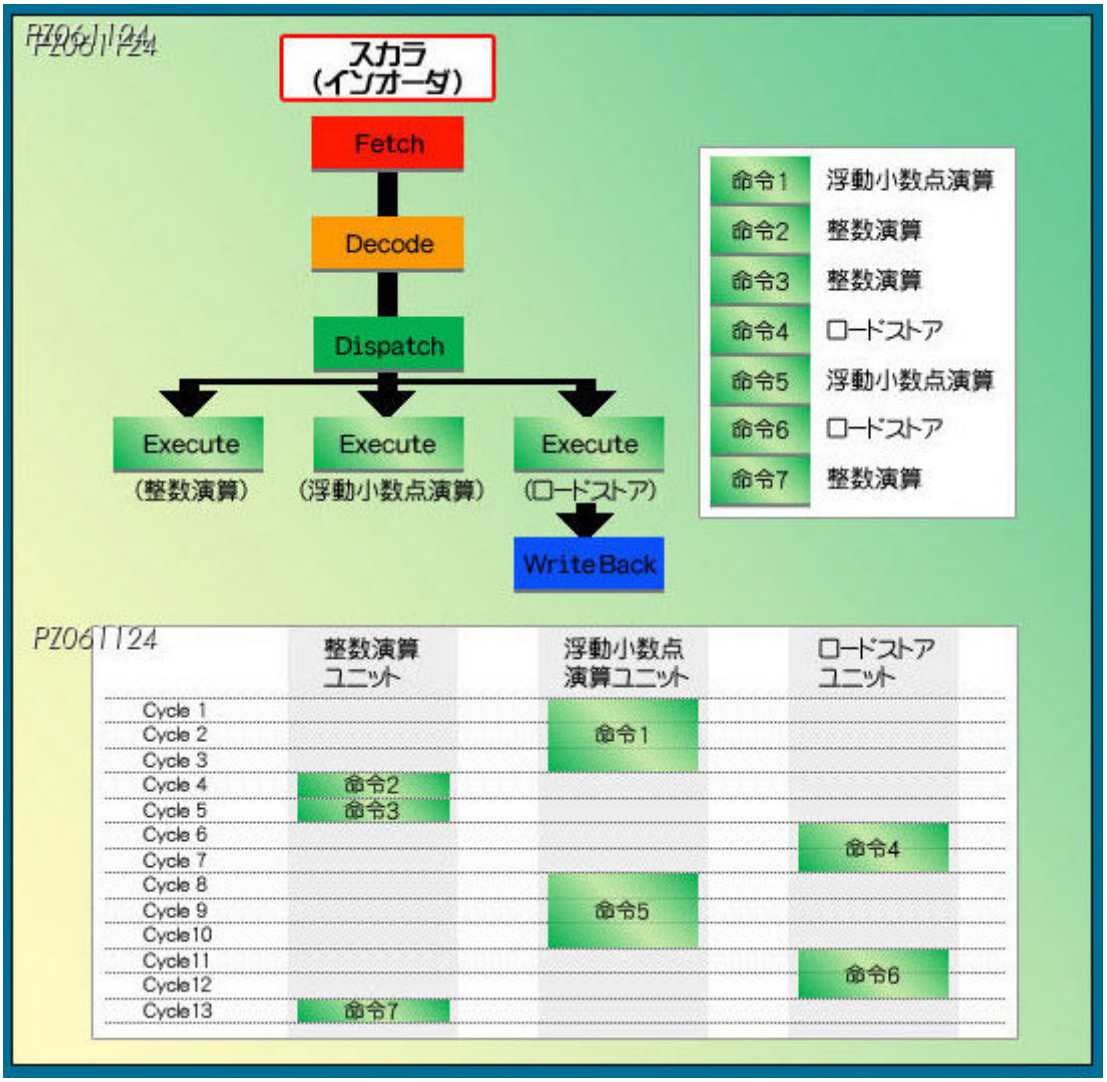

PZ06T124
スカラ
(インオーダ)
Fetch
Decode
Dispatch
Execute
(整数演算)
Execute
(浮動小数点演算)
Execute
(ロードストア)
WriteBack
命令1 浮動小数点演算
命令2 整数演算
命令3 整数演算
命令4 ロードストア
命令5 浮動小数点演算
命令6 ロードストア
命令7 整数演算
PZ06T124
整数演算ユニット
浮動小数点演算ユニット
ロードストアユニット
Cycle 1
Cycle 2
Cycle 3
Cycle 4
Cycle 5
Cycle 6
Cycle 7
Cycle 8
Cycle 9
Cycle10
Cycle11
Cycle12
Cycle13
命令1
命令2
命令3
命令4
命令5
命令6
命令7

그림4

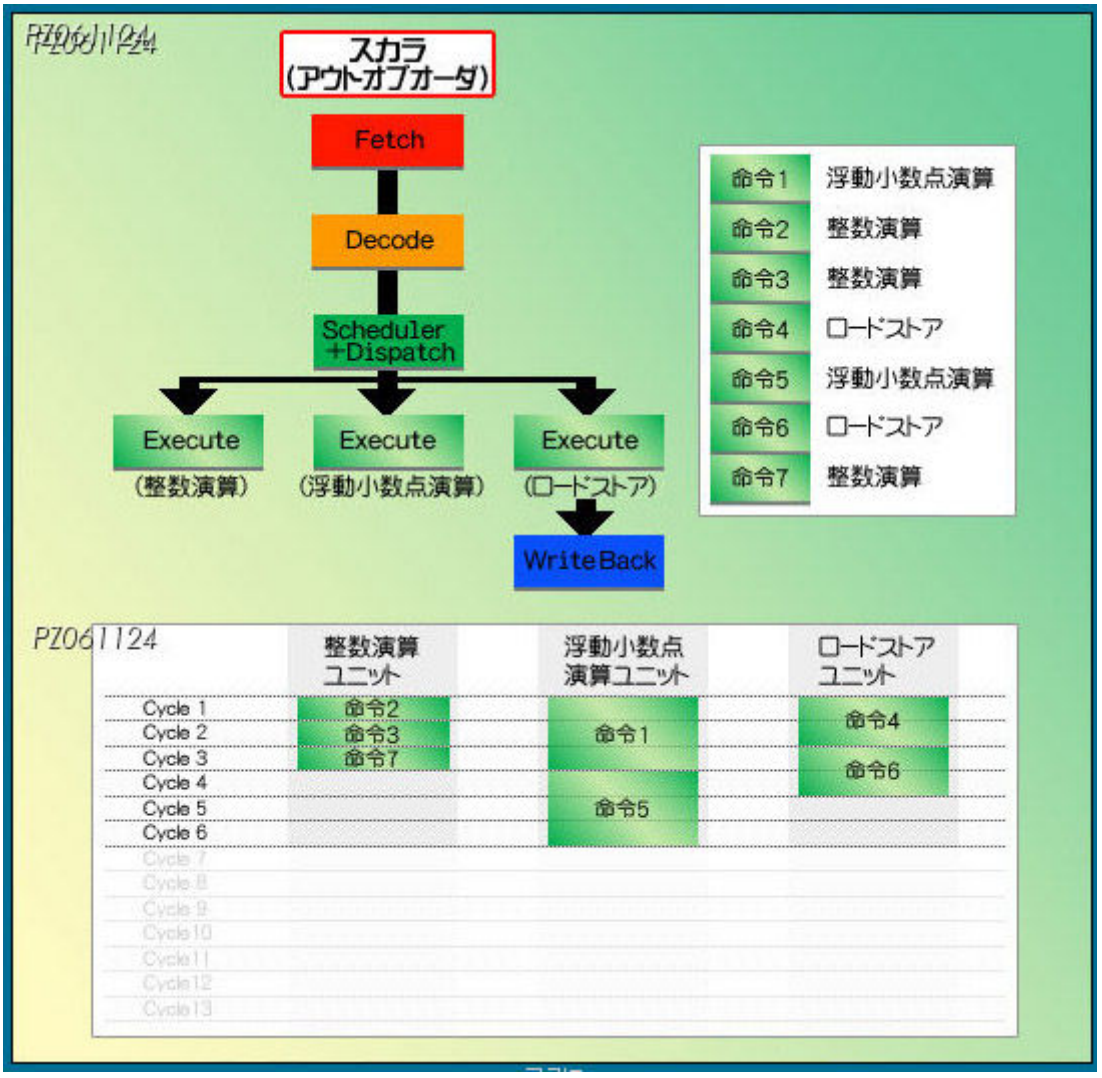


그림5

반대로, 동작 클럭을 올리려면 어떻게 할까. 이것은 제 2회 에서 말한 것처럼, 오로지 파이프라인 단수(stage)를 늘리는 것이 유리한 계책이라고 할 수 있다.

이 방향성의 대표가 Pentium 4로, 실제로 파이프라인은 31단을 가지고 있다. 이것은, 설계 단계에서5GHz의 실현을 염두에 둔 결과라고 한다. 단지, 여기까지 단수가 증가하면, 분기 미스에 의한 파이프라인 하자드(hazard)의 영향이 매우 커지므로, 분기 예측을 강화해, 효율을 떨어뜨리지 않게 한다고 하는 어프로치가 필요하게 된다. 결국, 어느 쪽의 어프로치가 보다 바람직한가는 통틀어 말할 수 없다. 이 문제는 첫머리에서 말했듯이(2), 즉, 읽어들인 데이터를 고속으로 처리하는 것과 관련하고 있다. 명령과 다른, 데이터 쪽은 조건 분기도 아무것도 없는 것으로, 단지 읽어들여 처리해 출력을 한다. 덧붙여서, CPU의 데이터 처리 능력은 아래의 식으로부터 계산 가능하다.

데이터 처리 성능 = 1 사이클 사이에 처리할 수 있는 명령수×동작 클럭
Performance = DPC (Data Per Cycle) × Frequency

명령 처리 성능과 같은 형태로 상상할 수 있지만, 결국은 1회로 취급할 수 있는 데이터의 수를 늘리던지, 동작 클럭을 올리던지.. 하면, 데이터 처리 성능은 향상된다.

그러나, 데이터의 수를 막무가내로 늘릴 수는 없다. 이것은 프로그램의 호환성에 관계해 오기 때문이다. 마음대로 확장하면, 구 버젼의 프로그램과의 호환성이 없어져 버린다. 단적으로 말하면, 지금의 x86 프로세서의 유일한 어드밴티지는 호환성인 이유로, 이것을 잃으면 아무도 따라 오지 않게 된다. 플레이 스테이션 → 플레이 스테이션2 → 플레이 스테이션3 에서 같은 게임이 동작 한다든가, Xbox360에서는 어떻게든 Xbox와의 호

환성을 유지하려고 노력한다고 하는 이야기처럼 PC의 세계에서도 같은 것이다.

따라서, 데이터 처리 속도를 올리려면 , 역시 동작 클락을 올리는 것이 유리한 계책이라는 것이 된다.

이렇게 해서, 명령의 처리 속도와 데이터의 처리 속도의 양쪽 모두를 감안하여,「어느 어프로치가 최적인가」를 결정하게 된다. 2005년 가을 시점에서 주류가 되고 있는 Pentium 4 ,Pentium M ,Athlon 64 의 3 제품으로 비교하면, 대체로 아래와 같은 경향을 가지고 있다.

Pentium 4 ： 동작 클럭 최고로 중시. 원래 디자인의 단계에서, 명령의 처리 능력 뿐만이 아니라, 데이터의 처리 능력을 높이는 일도 중요시한 설계이다. 발표 당시 , 프레스 릴리스에는「멀티미디어 시대를 맞이해 PC에 요구되는 데이터 처리 능력은 한층 더 높아질 것이다」라는 문언이 있었다. 즉, Intel은 명령 처리 능력과 데이터 처리 능력의 양쪽 모두를 끌어올리기 위해서는, 동작 클럭을 올릴 수 밖에 없다고 판단하였다.

Pentium M ：IPC 최고로 중시. 어쨌든「동작 클럭을 올리지 않고 , 얼마나 성능을 올릴까」에 주력 된 설계로, 캐쉬 미스를 줄인다든가 분기 미스를 최소한으로 한다든가로,「IPC (을)를 끌어올린다」라기 보다는, 오히려「IPC (을)를 떨어뜨리지 않는다」설계가 특징적이다.

Athlon 64 ：IPC 중시. Pentium M (정도)만큼 특화한 설계는 이루어지지 않았다. 물론, Pentium M과 같이,IPC를 떨어뜨리지 않기 위한 설계부분도 보여지지만, 전체적으로는「IPC (을)를 끌어올린다」는 방향의 어프로치에 다가서 있다. 또한, Pentium 4 만큼은 아니지만, 동작 클락도 중시하여 결과적으로Pentium 4 (와)과 Pentium M 의 사이에 위치하는 설계가 되어 있다.

칼럼 : CPU 확장 명령

여기까지 말해 온 것 예외에도, MMX ,SSE ,SSE2 ,SSE3 ,3DNow! ,Enhanced 3DNow! ,3DNow! Professional 등의 확장 명령이 있다. 아래 표2 에는 이들 확장 기능에 대한 각각의 특징을 요약해 정리하였다. 이러한 명령을 사용하면, 1회에 복수 데이터에 대해서 같은 처리를 적용할 수 있다. 이 방식을SIMD (Single Instruction, Multiple Data)이라고 부르지만, 이것들은 어디까지나 x86 명령의 확장으로, 모든 x86 명령을 옮겨놓을 수 있는 것은 아니다. 또「CPU는 이것을 서포트하고 있지만, 다른CPU는 서포트하고 있지 않은」것도 많기 때문에, 세상에 있는 모든 프로그램이 이것들을 활용하고 있는 것은 아니다. 단순하게 CPU의 성능이라고는 보기 어려운 요소이다.

PZ061124 拡張命令	データ幅	整数演算			浮動小数点演算	
		8bit	16bit	32bit	32bit	64bit
MMX	32bit	4	2	N/A	N/A	N/A
SSE	64bit	8	4	2	2	N/A
SSE2	128bit	16	8	4	4	2
SSE3	128bit	16	8	4	4	2
3DNow!	64bit	8	4	2	2	N/A
Enhanced 3DNow!	64bit	8	4	2	2	N/A
3DNow! Professional	64bit	8	4	2	2	N/A

표2

고성능화를 저해하는, 열과 소비 전력과 리크 전류

그런데, 여기까지는CPU 의 성능을 얼마나 끌어올리는가 하는 방법론의 이야기로 계속 했기 때문에, 이 쯤에서 심각한 과제가 되고 있는 것이「CPU 의 고성능화를 저해하는 요인」으로, 그것은「발열이 너무 많다」라고 하는 것이다.

우선은 조금 기초지식으로, 최근의 CPU는 대부분「CMOS 」라고 불리는 프로세스로 제조되고 있다. CMOS 란, MOS FET이라 하는 트랜지스터를 페어로 해 이용하는 것이며, 이 FET(Field Emmition Transistor , 전계 효과 트랜지스터)라고 하는 것이 CPU의 구조의 원점이다. FET의 G(Gate , 게이트), S(Source , 소스), D(Drain , 드레인)이라고 하는 세 개의 단자를 가져, G-S사이에 전압을 걸면, 거기에 대응해 S-D 사이에 전류가 흐른다고 하는 움직임을 갖는다(그림6 좌측).

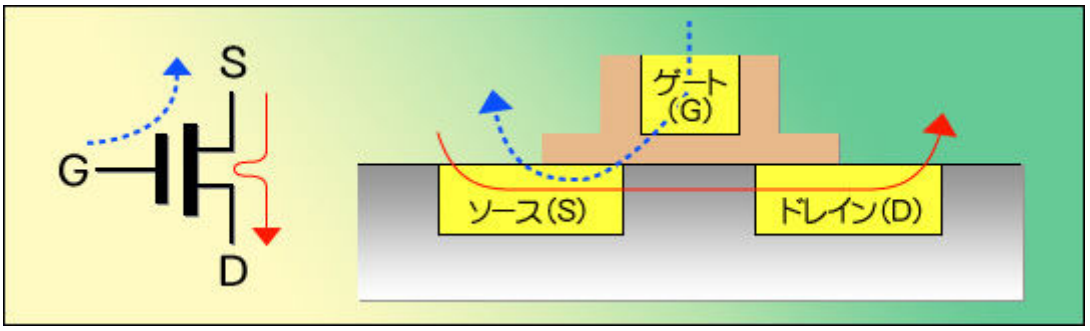


그림6

입력「전압」으로 출력「전류」를 컨트롤 할 수 있다는 것이, 여기서의 포인트다. 논리 회로에서는,0 인가 1일까 하고 말하는 상태를 보관 유지할 필요가 있지만, FET를 사용했을 경우, 예를 들면 0를 0V ,1 를 5V로 결정해 버리면, 이 시점에서 실제로 전류를 흘릴 필요는 없다. FET를 사용해 논리 회로를 구축하는 케이스에서는, 전류가 흐르는 것은 상태가 바뀔 때(0 →1 , 혹은1 →0)만 되고 있다.

그런데 이 FET는, 실리콘 위에 그림 6 우측과 같은 구조로 구축되고 있다. G-S 사이에 전압을 걸면(방금전 썼지만) 여기는 절연 구조이므로 매우 저항값이 높다. 옴의 법칙에서는 아래와 같이 되어 있기 때문에, 전압(E)가 일정한 저항(R)이 대단히 큰 값이라면, 흐르는 전류는 매우 적게 되는 것이다.

E = I × R → I = E ÷ R
(전압 = 전류 × 저항 → 전류 = 전압 ÷ 저항)

그리고, 소비 전력은 아래에 나타내는 대로 전압×전류로 나타내지기 때문에, 전류가 적게 되면 소비 전력도 작아지다.

W = E × I
(소비 전력 = 전압 × 전류)

그런데, 반도체에서 FET의 성질상, S-D 사이는 한 번 전류가 흐르기 시작하면, 저항값은 지극히 작아진다(그것이「반」도체라는 것이다). 그래서, 소비 전력을 요구하는 이 식에, 방금전의 옴의 법칙을 생각하면, 아래와 같은 식이 된다.

W = (I × R) × I = I2 × R
(소비 전력= 전류2 × 저항)

요컨데, 저항(R)이 한없이 0에 가깝게 되면, 그 경우도 소비 전력은 0에 가까워지는 것으로 된다.

단지, 어느 쪽 상태로 해도,「0 에 가깝다」≠「0 이다」라고 하는 것이 어려운 부분이다. 원래 완전하게 0이라면, 트랜지스터는 동작하는 것에 해당되고 전기를 먹지 않을 것이지만, 실제로 그것이 가능한 것은 초전도의 세계에서 통상 다소나마 소비 전력이 발생한다. 그리고 소비 전력이 무엇으로 바뀌는가 하면, 대부분이 열이 되는 것이다. 이것은 별로 반도체에 한정한 이야기는 아니다. 모터이든 전구이든, 어쨌든 전류가 흐르는 곳은 열이 반드시 발생한다. 상온초전도체가 등장하지 않는 한, 현재 발열은 피할 방법은 없고, 결과적으로 발열은

트랜지스터의 동작(=CPU 의 동작)에 대한 물건의 현상이 되고 있다.

지금까지의 CPU 와 발열이 그만큼 큰 문제가 되지 않았던 것은,「전류가 흐르는 것은 상태 변화시만」이기 때문이다. 그리고, 변화가 생기는 빈도는 CPU의 동작 클락에 비례하므로, 결과적으로 소비 전력과 발열도 동작 클럭의 상승에 비례해 증가해 가는 경향에 있다.

그렇다고는 해도, 제2 회에서 쓴 것처럼, 프로세스 룰(CPU 제조시의 배선 간격)을 축소해 나가면, 보다 낮은 전압으로 트랜지스터가 동작하게 된다. 어느 정도 전류가 흘러도, 전력은 작은 채로 유지할 수 있다.그러니까, 지금까지 문제가 되지 않았던 것이다.
　여기서 이과의 시간.우선, 반복이 되지만, 소비 전력의 계산식은 아래와같다.

W = E × I
(소비 전력 = 전압 × 전류)

이것에, 방금전과 같이, 옴의 법칙을 적용시키면, 전압을 내리는 것의 메리트는 훨씬 알기 쉬워진다.

W = E × (E ÷ R) = E2 ÷ R
(소비 전력 = 전압2 ÷ 저항)

즉, 전압을 내리면, 그 하락폭의 제곱에 비례하고 전력이 내린다. 결과적으로 발열도 줄일 수 있다.

이런 순환은, 0.18μm 프로세스 근처까지는 거의 문제 없게 성립해 왔다. 소비 전력을 그만큼 바꾸지 않고 속도를 올려졌던 것이다. 제2장에서의 (표)2에서 나타내 보였던 대로, Pentium III의 TDP (Thermal Design Power : 열설계 전력, 여기에서는 소비 전력과 거의 동의)를 거의 바꾸지 않고 동작 클락을 올려 올 수 있던 것이지만, 그것은 이, 프로세스 룰의 미세화에 의하는 것이 크다.

　하지만, 0.18μm 프로세스를 넘어서부터, 이런 순환이 성립하지 않아졌다. 그것은,「리크 전류」라고 하는 새로운 요인이 나왔기 때문에이다. 풀어 쓰면「새는 전류」이지만, 요컨데 트랜지스터가 움직이지 않을 때에서도, 그림7과 같이 여기저기로부터 전류가 새고, 흘러 버리는 현상이다. 트랜지스터가 동작하고 있지 않는(스윗치가 오프된 상태)때에도 전류가 흘러 버린다. 즉, 본래 흐르지 않아야 할 곳에 전류가 흘러 버린다. 그리고, 전류가 흘러 버리기 때문에, 소비 전력은(설계자가 상정했던 것보다도) 아득하게 커진다는 것이다.

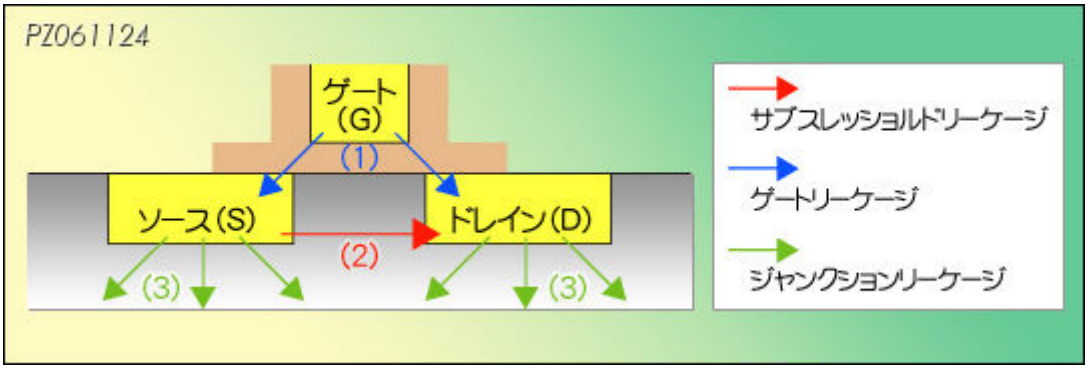


그림 7

소형화 되고, 프로세스의 축소에 의해서 배선끼리의 거리가 줄어들어, 절연이 충분하지 않게 되어 버리고 있다. 그렇다고 해서 두께를 더하면 트랜지스터의 스위칭 속도가 떨어져 동작 클락을 올려지지 못하게 된다.
　수도의 패킹같은 것으로 예를 든다면, 패킹을 두껍게 하면 물이 샐 걱정은 없어지지만, 여닫기에 힘이 들어가고, 시간이 걸린다. 그럼, 패킹을 얇게 해 나가면, 점차 물이 새가게되며, 더 얇게 하면, 패킹이 수압에 밀리게 되어, 물이 점점 누락이 될 것이다. 현재의 프로세스 룰에서는, 확실히 이런 점차 누락 상태가 발생해 버리

는 경우가 있어, 이 리크 전류가, 트랜지스터의 고속화에「기다렸다」는 듯이 나타난 건 형태가 되어 버렸던 것이다.

누설(Leakage ：리크 하는 것=새는 것)에는 크게 나누어3 종류 있다.

(1) 게이트 절연막으로부터의 리크(Gate Leakage)
(2) 소스→드레인으로부터의 리크(Subthreshold Leakage)
(3) 소스/ 드레인 자체로부터의 리크(Junction Leakage)

누설 자체는 이전부터 알려져 있던 이야기이다. G(게이트)에 제어하는「전압」을 걸치면, 거기에 대응해S(소스) → D (드레인) 방향으로 전류가 흐르는……여기서 중요한 것은, G/S 의 사이에는 이론상 전류가 흐르지 않는 것이다. 물론 실제로는 완전하게 0 이라고 할 수는 없지만, FET의 G/S 사이의 저항은 매우 높기 때문에, 무시할 수 있는 정도의 크기에 들어가 있다.

현재 문제가 되어 있는 것은(2)번의 Subthreshold Leakage이지만, 지금 보다 앞 선 프로세스 룰이 미세화해 나가면, 그 때는 Gate Leakage 나 Junction Leakage 쪽이 문제가 된다고 예측되고 있어 메이커 회사마다 모두에게 이것을 줄이기 위한 다양한 기술개발에 열중하고 있다.

그런 중,「이 문제에 언제 대처해야 하는가」라고 하는 타이밍을 완전하게 오인한 것이Intel 이다. Pentium 4의 경우, 0.13μm 프로세스에 있어서의 리크 전류는 전체의 10%정도로 들어가고 있었기 때문에, 그만큼 적극적인 대처를 할 필요가 없었다. 그러므로Intel은「90nm 프로세스에 대해도 그 연장으로 괜찮아」라고 할 전망을 세워 리크 전류에 본격적인 대처가 필요한 것은65nm 세대부터로 생각하고 있었던 것이다.

그런데 ,90nm 프로세스에서는 리크 전류의 비율이 30 ~50%라고 하는 값을 나타내, 이대로 동작 클락을 올려 가면 전혀 장난이 아닌 것이「만들어 버리고 나서」판명 해 버렸다.

CPU가 소비하는 전력은 메인보드로부터 공급되므로, 그 전력이 크면, 메인보드측에서 거기에 알맞을 만한 전력을 공급해야 한다. 게다가, 소비한 전력의 대부분은 열의 형태로 방출되기 때문에, 이 열을 제대로 받아 들여 발열/ 확산해 주지 않으면CPU 의 온도가 자꾸자꾸 오르고, 마지막에는 CPU 자체가 파괴되어 버린다. 따라서, 메인보드의 전력 공급 회로를 강화하는 것과 동시에, 발열 대책을 충실시키지 않으면 PC의 코어 파트가 온전히 사용할 수 없는 것이다.

4GHz 추월을 목표로 했던 Prescott 코어의 Pentium 4는, 소비 전력이 너무 커서 게다가 이것에 관련되는 다양한 트러블이 분출했기 때문에, 동작 클락을3.8GHz로 마지막 하지 않을 수 없었다. 게다가, 이Prescott 코어에서 계속 Pentium 4로서 개발하고 있었던 Tejas(보다 고 클럭으로 동작하며, 한층 더Hyper-Threading 테크널러지를 강화한 제품)을 캔슬 하지 않을 수 없게 되었던 것이다.

그래서 이런 문제에 직면해, 서둘러 등장시킨 것이"Pentium 4 의 코어를2 개 붙인"듀얼 코어의Pentium D , 개발 코드네임 Smithfield (스미스 필드) 제품이다. 이 Pentium D 에 있는, 한층 더 65nm 프로세스 룰를 축소한 Presler까지 이어왔다. 그리고 Pentium 4 (라고 하는지,Pentium D)의 노선을 계보를 이어주는 Presler가 최후가 될 것이라고 보여지고 있다.

데이터 처리 능력은, 아직 당분간 중요하지 않다.
Intel은 동작 클럭 중요시의 생각이 막힌 것을 나타내 보이고 있다.

여기서 다시 엔진의 예를 꺼내면, 고회전형 엔진은 레이스등에서 잘 사용되지만, 일반차에는 거의 사용되지 않았다.그 큰 이유는, 성능을 끌어 내는 것이 어렵기 때문이다. 고회전형 엔진은 항상 고회전으로 으로 하지 않으면 파워가 나오지 않으며, 도시지역에서는 비효율적이다.

또, 엔진의 부근의 부분들이 열화가 격렬한 것도 결점이라고 할 수 있다. 회전수가 높기 때문에 기구적으로 고장나기 쉽고, 빈번히 오버홀을 실시하지 않으면 안 된다든가, 엔진 오일이 곧 열화 한다든가 말한 결점도 가지고 있다. 한층 더 세세한 말을 말하면, 발열이 많고, 연비가 나쁘다고 하는 문제도 있어, 적어도 일상 용도에는 사용되는 일이 없다.

약간 무리이지만, 이것은 CPU에도 들어맞는 이야기다. 클럭 중시의 CPU는 성능을 끌어 내는 것이 어렵다. 캐쉬 미스를 최소함으로 억제해 메모리의 대역폭을 충분히 확보해야 하면서, 파이프라인 스톨등이 발생하지 않게 하지 않으면 전혀 성능이 나오지 않는 것이다.

동작 클럭이 높은 관계로「성능은 낮지만 오래 간다」는 것은 사용할 수 없으며, 고클럭이기 때문에 발열과 소비 전력은 많아진다. 결과적으로, AMD의 Athlon 64나 Intel의 Pentium M 등, 차와 엔진으로 말하면 적재량 승부의 고토르크형, 즉 IPC 중시형이 최근에는 주목받고 있다. 더 말하면 살아 남는 결과로 연결되었다고 하는 것이다.

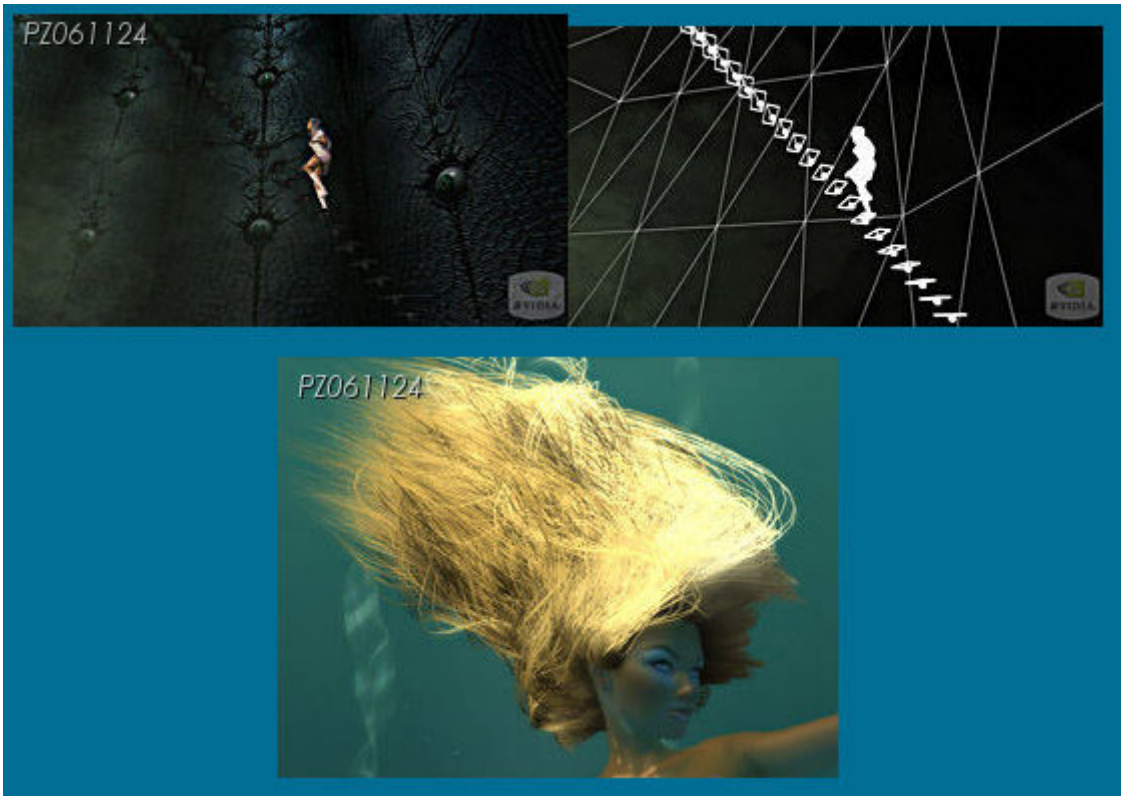
또, 여기까지 오게 된 경위에는「Intel이 생각한 만큼에, 데이터의 양은 많아 지지 않았다」라고 하는 요인도 관련되어 있다. 데이터의 양이 많은 처리되는 것은, 예를 들면 동영상의 encode라든지 트랜스 코드라든가 한 것이다.이러한 용도에 한정하고 말하면, 변함 없이Pentium 4 시리즈는 압도적으로 고속으로 있다.

하지만 게임이나 일반적인 Windows 조작에서 데이터의 양은 그만큼 많지 않고, 오히려 처리의 방법이 매회 바뀌는 패턴, 즉 단순한 데이터 처리에서는 짧은 패턴이 많은 것이다.
게임에서 생각해 보자. 적 AI의 캐릭터가 수만체 있고, 이것이 독립해 마음대로 움직이고 있으면, 대량의 AI라 하는 데이터를 처리하는 것이니까, 분명하게「데이터량이 많다」라고 할 수 있는 레벨이다.하지만, 그런 만들기의 게임에는 현재 뵈었던 적이 없고, 많아봐야 수십체 정도. 100개 넘는 패턴은 보기 드물다.

여기서「MMORPG (은)는?」라고 생각한 사람은 있겠지만, 그것은 또 이야기가 달라진다. PvP등을 구현하는 MMORPG에서 동시에100를 넘는 플레이어 캐릭터가 등장하는 것도 드물지 않지만, 이 경우 플레이어 캐릭터의 처리는, 네트워크로 연결된 각각 다른 PC나, 게임 서버측에서 행해지고 있다.자신의 PC가 처리해야 할 AI의 수는 그만큼 없어지고, 결과적으로 데이터량은 그다지 많지 않은 것이다.

게임의 미래를 이야기를 하면, 점차 데이터량이 많아지는 경향에 있는 것은 틀림없다. 예를 들면 현재의FPS에서, 벽면이나 마루에 있는 섬세한 입체형을 texture로 실현되고 있다. 많은 입체면이 있는 것처럼 보여도, 거기에는 거대한 1개의 다각형이 있는 것이며, 입체면을 표현하는 texture가 붙여지고 있을 뿐이다. 그러니까, 벽의 입체상태와 충돌 했을 때의 기세에 의해서, 데미지가 다르거나 하는 것 같은 일은 없다.

그러나, 향후 리얼리티가 추구되어 가면, 벽면의 입체을 제대로 데이터로 가져서, 섬세한 충돌 시뮬레이션을 실시해서, 그 결과에 따라 데미지가 결정되는 (울퉁불퉁의 마루에서 구르면 크게 찰과상 입을 수 있다, 라든지)는 일이 일어날 수 있다. 그리고, 거기에 필요한 데이터의 계산량은, 현재와는 비교가 되지 않을 것이다.



혹은 최근, 그래픽 카드의 데모에 수준 높은 그래픽의 사람의 머리카락의 표현으로 생각해 보자. 현재 머리카락 1본 1책의 운동 방정식은 생각보다는 조잡하기 때문에, 움직임은 별로 자연스럽지 않다. 이것은, 머리카락 1본 단위로 공기 저항까지 생각해 운동 방정식을 풀고 있으면, 그 처리가 너무 무거워서, 그래픽스의 데모로 되지 않을 것이다. 게임에 새로운 리얼함이 요구되게 되면, 머리카락의 계산량 하나만 취해도 데이터량은 큰 폭으로 증가할 것이다.

PC의 성능이 향상되고, 사용 용도가 계속 퍼지는 이상, "언젠가"는 데이터량의 폭발이 일어난다.이것은 우선 틀림없는 것이다. 그러나, 그것은 나중에 일어나는 것이라 생각된다. 필자 개인으로서는, 이러한 폭발이 일어나는 것은, CPU의 수가 더 비약적으로 많아지고 나서일거라고 생각한다. 즉 1개의 CPU 패키지에 복수의 CPU 코어가 들어가거나, CPU가 네트워크 경유로 제휴해 움직이는, 소결합의 클러스터가 실용적으로 되고 나서야 되는게 아닐까 생각하고 있다. 지금은, 데이터량의 폭발을 생각할 필요는 없다. 따라서, 우리들에게는「고속의CPU 」=「명령 처리 성능의 높은 CPU 」라고 생각해 두면 된다고 생각한다.

출처 : 4gamers 작가:大原雄介 편집: felfin

원본출처 : 4gamer.net

IP Address : 211.230.xxx.148