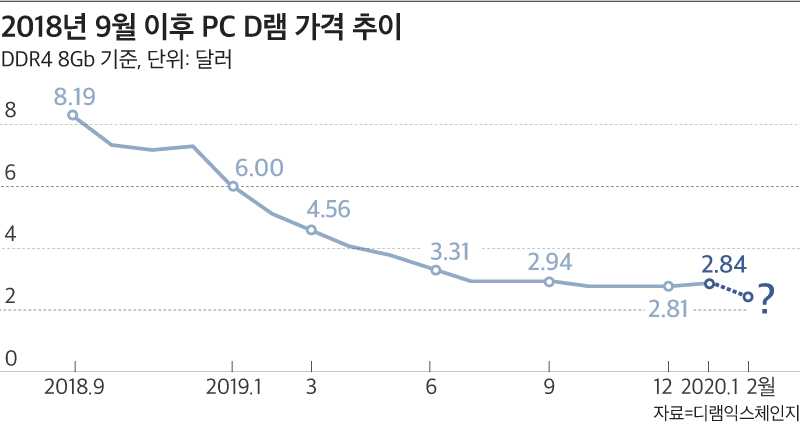
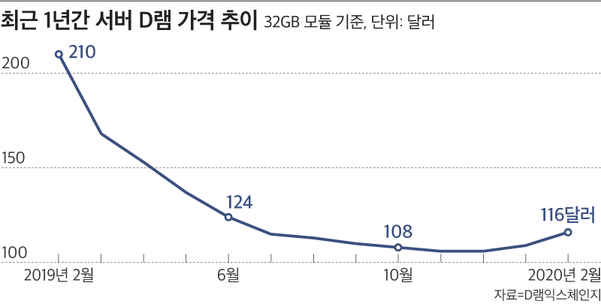
**Ex-9 Reading(4/24)**

Yes, Python is slow and I don't care.

<번역> 네 Python은 느립니다, 하지만 저는 신경쓰지 않습니다. 위 글을 읽고 짤막한 소감을 적으세요.

프로그래밍을 다루는 우리는 으레 그래왔듯이 항상 더 뛰어난 알고리즘을 가진 코드를 개발하기 위하여 노력해 왔습니다. 하드웨어의 성능이 소프트웨어의 효율속도를 따라가지 못할만큼 훌륭한 언어들의 등장과 뛰어난 프로그래밍 기술들이 많았지만, 오늘날의 기준으로 본다면 하드웨어의 성능은 무어의 법칙[[1]](#footnote-1)에 따라 18개월마다 200%가 되는 경향이 있습니다. 또한 최근의 D램의 가격 추이를 본다면 메모리 값이 더욱 저렴해 진다는 사실을 쉽게 알 수 있습니다.

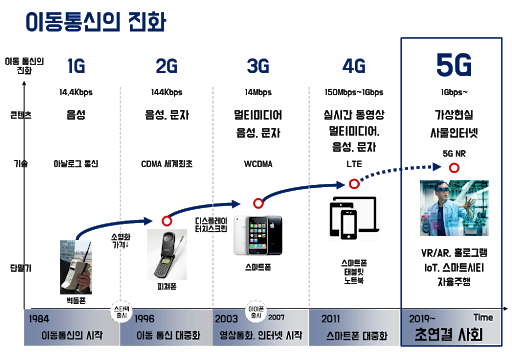




짧은 시간 동안에도 세상이 빠르게 변했습니다. 컴퓨터를 구성하는 가장 중요한 두 하드웨어인 중앙처리장치(CPU)와 메모리(RAM)이 급격하게 저렴해지면서, 고도로 집약적인 코드의 존재가치가 흐려졌습니다. 대신 2010년대를 서두로 모바일 디바이스의 시대가 열리면서, 스마트폰 제작 회사나 플랫폼 지원회사와 같은 IT산업 계열 기업들의 무한경쟁시대가 열렸습니다. 더 이상 기업들은 자신들의 고객들을 유치하기 위해서 최적의 코드를 요구 할 필요성이 없어졌습니다. 자신의 경쟁사가 아이폰 11 또는 갤럭시 20을 언제 출시하는지, 그 위에 어떤 OS와 어떤 프로그램을 설치하여 사람들의 관심을 받을 수 있는지가 더 중요해진 것입니다.

네트워크의 발전 속도도 절대적으로 본다면 그 속도가 느리다고 할 수는 없을 것입니다. 1세대 이동통신(1G)를 시작으로, 통신방식의 진화는 역사적으로 8년에 한번 진행되었습니다. 발표된 자료에 따르면 현재 서비스 중인 4세대 이동통신(4G)은 정지상태 기준 1Gbps[[2]](#footnote-2)를 지원합니다. 곧 서비스가 광범위하게 시행 될 5세대 이동통신(5G)은 현재 6GHz 이하 저속 광역망에서는 2Gbps까지 밖에 지원하지 않지만 곧 서비스 될 초고속 근거리망 28GHz 에서는 무려 20Gbps에 해당하는 경이로운 비트전송률 지원이 가능 할 것이라고 전망하고 있습니다.

하지만 문제는 CPU 연산 속도와의 차이에 있습니다. 유선통신망은 모바일 이동통신망의 속도에 발맞춰 따라가지 못하기도 하고, 빠르게 성장하는 무선통신망이 이정도로 놀라운 비트전송률을 지원하게 되더라도, 그것보다 더욱 더 빠른 CPU 연산속도에는 따라가지 못한다는 점입니다. 이는 네트워크의 본질적인 속도(지연시간)가 CPU 연산속도보다 빠르지 않으면 결국 개발자가 집약적인 프로그래밍을 하여도 제품의 개발 효율성에 있어서는 큰 영향을 미칠 수 없다는 것을 의미합니다.



물론 런타임과 Input Output 처리시간의 절대값이 중요하지 않다는 것은 아닙니다. 직접 기계를 다루면서 네트워크를 크게 필요로 하지 않는 영역인 항공, 자동차, 기계장치 영역 등에서는 절대적으로 런타임의 크기와 안전을 위한 코드의 신뢰성이 중요할 것입니다. 하지만 안전 및 신뢰성이 중요한 그런 특수한 영역들을 제외하고는 현대의 IT 자원들은 코드 최적화보다는 새로운 제품과 기술 출시에 초점이 맞춰져 있습니다. 우리는 그러한 마켓의 요구가 시간에 따라 움직이는 것을 유심히 지켜 보아야 할 필요가 있습니다.

위의 내용을 바탕으로, 프로그래밍을 해야 하는 사람들은 이제 더욱 더 높은 수준의 최적화보다는 빠르게 개발에 착수 할 수 있는 생산성에 초점이 맞춰 질 수 밖에 없습니다. 하드웨어는 이제 매우 저렴하여 그 공급이 넘쳐나지만, 번뜩이는 아이디어로 빠르게 코드를 구현해 내어야 할 인재는 그에 비해 충분하지 않습니다. 그들은 생산성을 높일 필요가 있고, 그 수요에 따라 같은 개발 시간 내에 상대적으로 생산성이 높은 언어가 이 시장에서의 최종 승자가 될 것입니다.

Python은 개발자들에게 훌륭한 도구가 될 수 있습니다. 가장 소중한 자원은 이제 개발자의 시간이 되어버렸습니다. 만약 코드에서 성능문제가 발생한다면, 모든 부분을 최적화 할 필요 없이 병목현상이 일어나는 부분만 찾아 해결하면 됩니다. 또한 파이썬 인터프리터가 해결하기 힘든 병목 현상이라면, Cython[[3]](#footnote-3)을 활용하여 처리 할 수 있습니다. 최적화와 개발속도 모두를 적절히 분배할 줄 아는 사람이 현재 IT 기업들이 요구하는 인재상에 부합할 것입니다.

1. 인텔의 고든 무어가 1965년에 주장한 법칙. $1000로 살 수 있는 반도체의 집적회로 성능은 18개월마다 2배로 증가한다는 법칙이다. 좀 더 광범위하게 정의하면 컴퓨터의 성능은 일정 시기마다 배가하며 기하급수적으로 증가한다는 법칙을 말한다고 볼 수 있다. [↑](#footnote-ref-1)
2. bps는 bit per second 이므로, 바이트 속도 기준 변환시 8로 나누어 주어야 한다. [↑](#footnote-ref-2)
3. Python의 속도를 높이고자 아예 C언어로 변환하여 컴파일하는 방법을 택한 패키지. 파이썬의 문법을 확장하여 정적 타입 선언 기능을 넣어, 기존의 C 코드를 쉽게 접착할 수 있도록 만들었다. Python C API를 이용하여 C 코드를 직접 접착하는 기존의 방식보다 훨씬 사용하기 쉽고, C언어로 변환되어 컴파일 된다는 점 때문에 ctypes로 덧씌우는 것 없이 네이티브 코드(\*.so, \*.dll)를 직접 사용할 수 있다. 이것을 사용하면 적게는 열 배 정도에서 많게는 천 배 이상까지 속도 향상을 경험할 수 있다. [↑](#footnote-ref-3)