Computer Hardware & Industry Perspective

크래프톤 정글 8기 09 박창현

CPU의 성능 향상이 둔화된 이유와 기술적 대안

서론
CPU 성능의 향상은 한때 무어의 법칙에 따라 비약적인 발전을 보였다. 하지만 최근 몇년간의 향상 그래프는 점차 성능이 둔화되는 방향으로 흘러가고 있으며 무어의 법칙이 아직 유효한지도 재검토되고 있는 상황이다. 이러한 성능 둔화는 일시적인 현상이 아닌, 기술적-물리적 한계에 기인한 구조적 추세로 분석된다. 본 보고서에서는 성능 향상 둔화에 대한 원인과 배경을 살펴보고, 이러한 정체에 대응하는 기술적 대안들을 정리한다. 또 개발자가 도전할 수 있는 분야와 방향성에 대해 다뤄본다.

본론
1) CPU 성능 향상 둔화의 주요 원인
CPU 성능 정체의 배경에는 물리적 한계, 전력 및 발열 문제, 설계상의 한계 등이 복합적으로 작용하고 있다. 아래에서 주요 원인을 살펴본다

- 반도체 공정의 물리적 한계와 무어의 법칙 둔화 : 반도체 제조 공정이 수 나노미터 영역까지 미세화됨에 따라 소자 크기는 한계에 부딪히게 되어 더이상 작게 만드는 것이 어려울 정도의 수준까지 도달하게 되었다. 무어의 법칙이 예견한 트랜지스터 집적도의 기하급수적 증가도 최근에는 속도가 느려지며 성능 향상 폭이 눈에 띄게 감소하고 있다. x86 프로세서들도 제조 한계에 부딪혀 성능 개선이 정체되는 양상을 보이고 있으며, 더 미세하게 만들기위한 공정 난이도와 개발 비용 급등으로 더더욱 어려워지고 있다.

- 전력 소모와 발열에 따른 Power Wall : CPU 성능을 높이는 전통적인 방법은 클럭 주파수를 올리는 것이었으나, 전력 소모와 발열이 주파수에 비례해 급증하는 물리 법칙으로 인해 전력 장벽(Power Wall)에 봉착했다. 한동안 프로세서들은 3~4GHZ대의 클럭 주파수에서 발열과 안정성 한계로 인해 정체되었으며, 그 이상의 속도 향상이 어려웠다. 트랜지스터를 동시에 최고 성능을 내게 동작시키더라도 발열과 전력으로 인해 한계를 넘을 수 없던 것이다.

- 아키텍처의 성숙과 설계상의 한계 : x86 프로세서 아키텍쳐는 40년동안 지속적으로 성능을 개선해왔으며, 설계적인 측면에서 더이상 혁신적인 개선에 대한 여지가 감소한다는 평가가 나오는 상태이다. 현대 CPU는 파이프라이닝, 분기 예측 등의 명령어 수준 병렬성(ILP) 향상 기법을 이미 최대한 활용하고 있으며 근본적인 설계의 변화 없이는 더이상 개선이 어렵다는 추세이다. 실제로 엔비디아 CEO 젠슨 황은 “CPU의 트랜지스터 수는 매년 50%씩 늘지만 성능 향상은 10%에 불과하며, 복잡한 범용 CPU보다 GPU와 같은 병렬 프로세서가 훨씬 빠르게 발전하고 있다"고 언급한 바 있다. 이 말을 통해 현재 CISC방식으로는 투입된 트랜지스터 대비 성능 향상을 크게 이끌어 낼 수 없어 한계에 봉착한 것이다.

2) 성능 정체에 대응하는 기술적 대안들
CPU 성능 향상이 둔화됨에 따라, 업계와 학계에서는 이를 극복하거나 보완하기 위한 다양한 대체 기술과 구조적인 해결책을 모색해왔다. 대표적인 대응 전략으로는 멀티코어 및 병렬 컴퓨팅, GPU 등 가속기의 활용, AI 전용 가속기 개발, 칩렛(chiplet) 기반 구조 등이 있다. 이러한 접근들은 각각 다른 방식으로 연산 성능을 향상시키거나 병목을 해결하여, 단순한 CPU 클럭 상승에 의존하지 않고도 전체 시스템 성능을 끌어올리고자 한다. 아래에서는 각 대안 기술의 개념과 역할을 살펴본다.

- 멀티코어와 병렬 컴퓨팅 : 멀티코어(multicore)는 단일 프로세서 칩 내에 여러 개의 코어를 집적함으로써 병렬 처리 성능을 높이는 기술이다. 이는 단일 코어의 클럭을 높이기 어려워진 상황에 대한 직접적인 대응책으로 등장하였다. 2005년경 이후 CPU 업계는 클럭 향상의 한계를 인식하고, 남는 트랜지스터 예산을 활용해 코어 수를 늘리는 방향으로 선회하였다. 그 결과 오늘날 소비자용 CPU에서도 듀얼코어, 쿼드코어를 넘어 8코어, 16코어까지 채택되고 있으며, 서버용 프로세서는 한 소켓에 수십~수백 개의 코어를 탑재하는 시대가 되었다. 멀티코어 설계 덕분에 프로세서 전체 성능은 코어 수에 비례하여 향상될 수 있었고, 병렬 컴퓨팅을 통해 작업 처리량(throughput)을 높이는 전략이 자리잡았다.

- GPU 가속 활용 : CPU만으로 성능 향상이 어려워지면서, GPU와 각종 가속기(accelerator)가 범용 연산 일부를 대체하거나 보조하는 역할을 맡게 되었다. GPU(Graphics Processing Unit)는 원래 그래픽 연산을 위해 발전한 프로세서이지만, 수천 개 이상의 코어를 통한 대규모 병렬 처리에 특화되어 있어 과학 계산, 머신러닝 등 데이터 병렬 작업에서 탁월한 성능을 발휘한다. 실제로 GPU는 동시 연산 코어 수와 메모리 대역폭 측면에서 CPU를 압도하여 행렬 계산, 벡터 연산등에서 수십 배 이상의 성능 우위를 보인다. 프로그래밍 모델도 CUDA, OpenCL 등 GPGPU(범용 GPU 컴퓨팅) 기술이 정착되어, 개발자들은 병렬로 처리 가능한 작업을 GPU로 오프로딩(offloading)함으로써 전체 성능을 높일 수 있다. 현대의 딥러닝, 영상/신호처리, 과학 시뮬레이션 분야에서는 GPU 없이는 현실적인 성능을 내기 어려울 정도로, GPU 가속은 CPU 성능 정체 시대에 필수적인 대안이 되었다.

- 칩렛 기반 구조 : 칩렛이란 하나의 큰 반도체 칩을 기능별로 나눈 여러 개의 작은 칩 조각(die)으로 제작한 후, 고속 인터커넥트로 묶어 하나의 시스템처럼 구동하는 설계 방식이다. 전통적으로 고성능 CPU는 단일의 크고 복잡한 모놀리식 다이(monolithic die)로 제작되었지만, 공정 미세화의 한계와 비용 상승으로 인해 일정 크기 이상의 다이는 수율 저하와 제조 난이도가 급격히 높아진다. 칩렛 아키텍처는 이러한 문제를 해결하기 위해 기능을 분할하여, 예컨대 코어 블록과 I/O 인터페이스 블록을 별도의 칩으로 만들고 패키지 내에서 연결하는 식이다. 이 접근법의 핵심 장점은 각 블록을 가장 적합한 공정 노드에 최적화하여 제조할 수 있다는 것이다 . 실제로 AMD는 Zen2 EPYC 프로세서(2019)부터 칩렛 구조를 도입하여, 고성능이 필요한 코어 칩(let)은 7nm 공정으로, 메모리 컨트롤러와 I/O 기능을 담당하는 칩은 14nm 공정으로 제작하였다. 이를 통해 전체 비용 절감과 성능 최적화를 동시에 달성하고, 하나의 패키지에 최대 64코어 이상의 프로세서를 구현하는 데 성공했다. 칩렛은 또한 작은 다이들을 사용함으로써 제조 수율 개선과 설계 유연성을 얻을 수 있다. 문제 발생 시 개별 칩렛만 교체할 수 있고, 다양한 조합으로 제품 스케일 업(scale-up)이 가능하다는 점에서 확장성과 경제성이 뛰어나다.

3) 개발자가 도전할 수 있는 새로운 분야

CPU 성능 향상의 둔화는 소프트웨어 개발자들의 개발 패러다임에도 변화를 요구하고 있다. 과거에는 하드웨어 성능 향상이 빠르게 이루어졌기 때문에, 코드를 최적화하지 않아도 시간이 지나면 빨라지는 "공짜 점심"이 가능했다는 우스갯소리가 있었다. 하지만 이제는 자동적인 성능 향상을 기대하기 어려워졌기 때문에, 개발자들은 새로운 접근과 기술을 습득하여 직접 성능 향상을 이끌어내야 한다. 특히 다음과 같은 분야들은 현대 개발자들에게 유망한 도전 과제로 떠오르고 있다:

- 병렬 프로그래밍 및 멀티스레드 개발 : 앞서 언급한 멀티코어 시대에 효율을 끌어내기 위해, 개발자는 애플리케이션을 병렬화하는 능력이 필수가 되었다. 작업을 여러 쓰레드로 분할하여 동시 실행하고, 락 경쟁이나 동기화 이슈를 최소화하는 등 동시성 프로그래밍기법을 익혀야 한다. 예를 들어 대용량 데이터 처리나 영상 렌더링, 서버의 동시 요청 처리 등에서 멀티스레딩과 분산 컴퓨팅을 활용하면 큰 성능 향상을 얻을 수 있다. 개발 도구 측면에서도 멀티코어 최적화를 지원하는 라이브러리(OpenMP, TBB 등)와 패러다임(함수형 프로그래밍의 불변성 활용 등)을 학습하여 적용하는 것이 요구된다.

- GPU 및 AI 가속기 활용 능력 : CPU만으로 한정된 성능을 넘어서기 위해, GPU 프로그래밍(CUDA, OpenCL 등)이나 AI 가속기(TPU, FPGA 등)의 활용은 개발자의 중요한 역량이 되고

있다. 예를 들어 머신러닝 모델 훈련이나 추론 코드를 작성할 때는 GPU를 활용한 딥러닝 프레임워크(PyTorch, TensorFlow 등)를 다룰 줄 알아야 하며, 영상처리 애플리케이션에서는 OpenCL이나 금속(API)을 통해 GPU의 병렬 연산을 활용해야 최상의 성능을 낼 수 있다. 또한 최근 클라우드 환경에서 제공되는 다양한 전문 가속기(API로 접근 가능한 ML 가속 칩 등)를 효과적으로 사용하는 법도 익혀야 한다. 요컨대 개발자는 이기종 컴퓨팅(heterogeneous computing)환경에 대비하여, 어떤 연산을 CPU vs GPU/가속기에서 처리할지 분배하고 최적화하는 기술 스택을 학습해야 한다.

- 하드웨어 친화적(low-level) 프로그래밍 및 최적화 : 성능을 극대화하기 위해 하드웨어와 밀접하게 연관된 코딩 기법을 연구하는 것도 개발자에게 새로운 도전 분야다. 컴파일러의 자동 최적화에만 의존하지 않고, 경우에 따라 어셈블리 수준의 튜닝이나 CPU SIMD 명령어(AVX, SSE 등) 활용을 통해 성능을 끌어올릴 수 있다. 또한 메모리 계층 구조를 이해하고 캐시 미스 회피, 데이터 배치 최적화 등을 구현하는 것이 중요해졌다. 가비지 컬렉션 언어보다는 Rust, C/C++같이 시스템 수준 제어가 가능한 언어를 선택하여 메모리 관리와 병행성을 세밀하게 조정하는 것도 고려된다. 더 나아가 임베디드 분야에서는 직접 FPGA를 프로그래밍하거나, HW/SW 코디자인을 통해 특정 알고리즘을 하드웨어 가속기화하는 등 소프트웨어와 하드웨어의 경계를 넘나드는 개발이 이뤄지고 있다. 이러한 노력들은 개발자에게 높은 난이도를 수반하지만, 한정된 자원에서 최대 성능을 뽑아내는 경쟁력을 부여해 줄 것이다.

위와 같은 분야들은 CPU 성능 정체 시대에 개발자가 주도적으로 성능을 개선할 수 있는 길을 열어준다. 결국 현대의 소프트웨어 개발자는 하드웨어의 동향을 이해하고 이를 활용/보완하는 복합적인 역량이 요구되며, 이는 곧 시스템 전체를 보는 안목과 전문성으로 이어져 커리어 발전에도 중요한 자산이 될 것이다.

결론

CPU 성능 향상 둔화는 기존 방식의 한계를 드러내면서 동시에 새로운 혁신의 방향성을 제시하고 있다. 이제는 과거처럼 하드웨어 성능 향상에 안주하기 어려운 시대라는 것이다. 개발자는 성능 최적화와 병렬화에 대한 감각을 키우고, GPU나 특수 가속기 등 새로운 도구를 적극 활용해야 한다. 또한 시스템 전체를 보는 시야를 갖추어, 어떤 작업을 어떤 자원으로 처리하는 것이 최선인지 판단하고 조율하는 능력이 중요해질 것이다. 이는 개발자들에게 더 높은 수준의 난이도를 요구하지만, 반대로 말하면 더 큰 성취감과 경쟁력을 가져다줄 수 있다. 하드웨어와 소프트웨어의 경계가 가까워지는 현 시점에서, 개발자는 기존보다 폭넓은 지식을 갖춘 전략적 문제 해결사로 거듭나야 할 것이다.