컴퓨터구조 추가 자료

- 4장, 5장 part -

< 캐시 메모리의 작동 원리 >

출처1: https://namu.wiki/w/%EC%BA%90%EC%8B%9C%20%EB%A9%94%EB%AA%A8%EB%A6%AC
출처2: https://code-lab1.tistory.com/322

캐시 메모리는 참조 지역성(Locality)의 원리를 이용한다.

참조 지역성은 시간 지역성(Temporal locality), 공간 지역성(Spatial Locality), 순차적 지역성(Sequential Locality)으로 나뉘는데, 시간 지역성이란 for나 while 같은 반복문에 사용하는 조건 변수처럼 한번 참조된 데이터는 잠시 후에 또 참조될 가능성이 높다는 것이고, 공간 지역성이란 A[0], A[1]과 같은 데이터 배열에 연속으로 접근할 때 참조된 데이터 근처에 있는 데이터가 잠시 후에 사용될 가능성이 높다는 것이며, 순차적 지역성이란 분기(branch)가 발생하는 비순차적 실행이 아닌 이상 명령어들이 메모리에 저장된 순서대로 실행하는 특성을 이용한 원리로 순차적일수록 다음 순서의 데이터가 사용될 가능성이 높다.

CPU가 메모리에 데이터를 요청할 때, DRAM에 접근하기 전에 일단 캐시 메모리에 접근하여 데이터 존재 여부를 확인한다. 캐시 메모리는 메인 메모리인 DRAM보다는 그 사이즈가 매우 작아서 데이터를 모두 저장할 수 없다. DRAM이 보통 수 GB 단위 정도인데 인텔 i5, i7에 들어가는 캐시 메모리는 작게는 수 KB ~ 많게는 수 MB 정도이다. 캐시 메모리는 DRAM의 데이터 일부를 가지고 있다가 CPU가 요청한 데이터가 캐시 메모리에 없으면 CPU를 잠시 기다리게 한 후 DRAM에서 해당 데이터를 가져온 후 CPU에게 넘겨 준다. CPU는 캐시의 존재를 알고 있지만, 그 위에서 실행되는 프로그램은 메인 메모리의 주소만 지정하거나 캐시 적중률을 위해 힌트를 줄 수는 있어도, 프로그래머가 캐시 메모리를 직접 지정할 수는 없다. 이렇게 그 존재가 외부에 드러나지 않기 때문에 캐시 메모리는 CPU에 투명(transparent)하다고 한다. 투명하지 않은 작은 온칩 메모리는 Scratchpad Memory라고 부른다.

```
for i in 0..n
  for j in 0..m
    for k in 0..p
      C[i][j] = C[i][j] + A[i][k] * B[k][j];
```

예를 들어 위와 같이 배열의 곱셈을 하는 코드가 있다고 하자.
배열의 마지막 차원에 원소를 연속적으로 정렬하는 언어에서는 루프 순서인 j와 k의 위치를 바꾸는 것만으로도 엄청난 속도 증가를 달성할 수 있다. 특히, 원소가 10만 개가 넘거나 L1, L2 캐시가 감당하지 못할 만큼 큰 배열의 경우 엄청난 효율을 자랑한다.
첫 번째 코드는 A[i][k] 배열이 캐시에 있다. k가 마지막 차원에서 연속적으로 증가하기 때문이다. 하지만 B[k][j]는 첫 번째 차원에서 k가 증가하므로 캐시 미스가 나게 된다.

```
for i in 0..n
  for k in 0..p
    for j in 0..m
      C[i][j] = C[i][j] + A[i][k] * B[k][j];
```

반면, 위와 같이 코드를 바꾸는 것만으로도 프로그램 속도를 향상시킬 수 있다. A[i][k]는 내부 루프에서 고정되어 있고, B[k][j]는 마지막 차원에서 j가 증가하므로 공간적 지역성을 충분히 활용하며 캐시 적중율을 높일 수 있다. 물론 수학적으로도 옳은 식이다. 다만 순서가 다를 뿐이다. 이처럼 참조의 지역성 원리를 잘 이해하고 사용할 수 있다면, 프로그램의 성능을 개선하는 데 큰 도움을 줄 수 있다.

< 외부 입출력 장치와 프로그램의 연결 >

※ 별도 강의자료(5장 PPT)에 있는 fopen 함수에 대하여 이해해보세요.