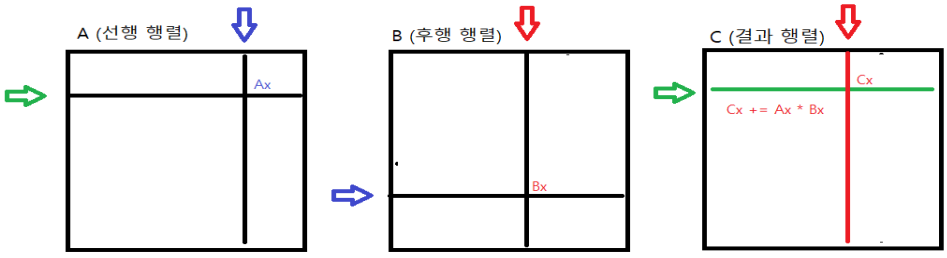
# 집중교육2- Vector Programming (1차과제 보고서)

201521032 한태희

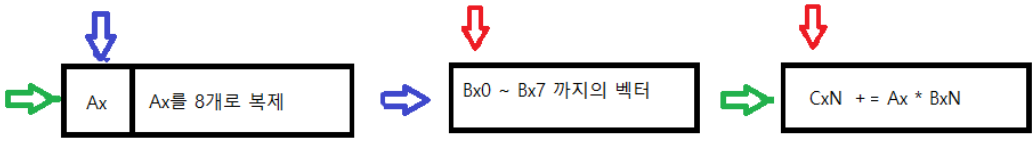
## 1. 디자인과 구현

첫번째 실습 과제의 주제는 인텔에서 제공하는 AVX 연산을 이용해 행렬 곱을 구현하는 것이다. 이를 위한 디자인은 실습 시간에 제공되었던 행렬 곱셈 방법을 참고로 하였다.

행렬 곱의 구현 방법은 간단하게 구현할 수 있는 $O(n^3)$의 알고리즘을 따른다.




스칼라 연산을 통한 구현은 3중 반복문으로 이를 구현하였다. 첫번째 반복문에서 행렬 A의 행 좌표를 증가시킨다. 두번째 반복문에서 행렬 A의 열 좌표를 증가시키고 A 행렬에서 Ax의 값을 정한다. 세번째 반복문에서 행렬 B의 열 좌표를 증가시키며, A 행렬의 열 좌표를 B 행렬의 행 좌표로 간주해 B 행렬에서 Bx의 값을 구한다. 그리고 A의 행 좌표와 B의 열 좌표에 대응되는 C 위치인 Cx에 Cx +=Ax*Bx 연산을 한다.

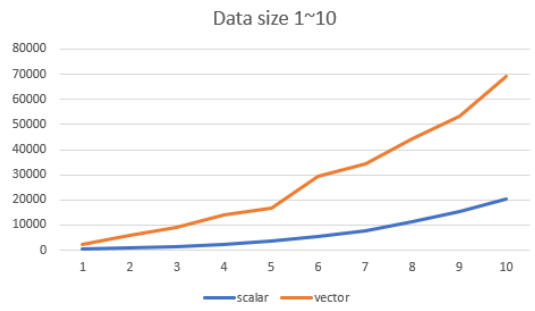



벡터 연산을 통한 구현은 스칼라 연산에서 세번째 반복문을 변화시켜 응용하였다. 스칼라 방식에선 기존 반복문에서 B 행렬의 행 좌표를 한 칸 이동시켰지만, 벡터 연산을 할 때는 8칸씩 이동시킨다. 그 사이의 8개의 float 값들을 벡터로 추출하여 Ax를 8개로 복제한 벡터와 곱해 결과 벡터를 만든다. 그 뒤, 결과 벡터를 대응하는 C의 위치에 더해 넣는다.

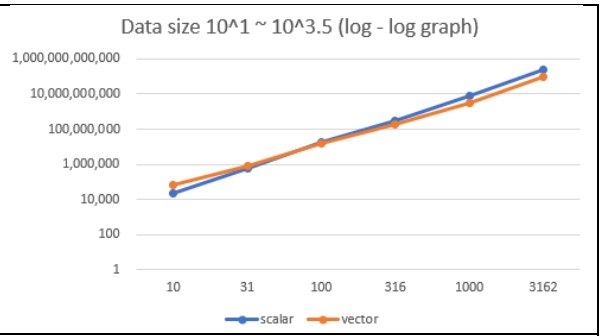
이 구현 방식은 B 행렬과 C 행렬에서 추출한 벡터의 각 요소들이 동일한 열에 위치하고 있다는 전제가 있다. 하지만, n * m의 연산을 수행하면 각 열의 마지막 부분에 다른 열의 정보, 논리적으로는 빈 공간이 입력되게 된다. 이 경우에는 결과 벡터를 바로 넣는 것이 아닌, 유효한 값들만 C에 입력되도록 예외 처리를 했다.

## 2. 성능 측정과 비교

성능 측정에는 독립변수로 1. 사용한 계산방법 (scalar vs vector) 2. 데이터의 크기를 고려하였고, 종속변수로 __rdtsc() 함수를 이용해 연산에 필요한 클럭 횟수를 측정했다. 측정은 두개의 크기가 똑같은 정사각행렬의 곱셈으로만 진행했다. 첫번째 측정은 1에서 10 사이의 크기를 1씩 늘려가며 측정을 진행했다. 두번째 측정은 10^1에서 10^3.5 사이의 크기에서 지수를 0.5씩 늘려가며 측정했다.




<데이터의 크기가 1에서 10 사이인 경우>




<데이터의 크기가 10^1에서 10^3.5 사이인 경우>

데이터의 크기가 1에서 10 사이인, 데이터가 작은 환경에선 벡터 연산을 사용하는 것보다 스칼라 연산을 사용하는 것이 훨씬 좋은 속도를 보여주었다. 벡터 연산을 하기 위해서는 데이터를 분배하고 취합하는 스칼라 연산에서는 없는 제어 과정이 추가되는데, 제어 과정도 컴퓨터의 연산 자원을 소모하기 때문에 그것이 기존 연산보다 더 많은 시간을 사용하는 것으로 보인다.

반대로 데이터의 크기가 엄청나게 큰 환경에서는 벡터 연산을 해서 연산 속도를 높이려는 의도가 효과를 보기 시작했다. 데이터의 크기가 100보다 작을 때는 스칼라 연산이 효율이 좋았지만, 그 이상에선 벡터 연산이 더 좋은 효율을 보이기 시작했다. 위의 오른쪽 그래프는 로그 그래프라 그 차이가 미미하게 표시되는 것 같지만, 실제 측정 값은 3162에서 스칼라 연산은 233017166902 (2.33* e^11)틱, 벡터 연산은 100620403535(1.00* e^11)틱이 측정되었다. 엄청나게 긴 시간에서 2배 차이이다. 실제 측정을 위해서 기다렸던 입장으로는 몇 분 이상의 차이로 매우 큰 성능 향상의 체감을 느낄 수 있었다.

다만, 10000 크기의 데이터를 측정하는 데는 두 함수를 사용하는 경우 모두 시간이 너무 오래 걸려 측정을 포기했는데, 벡터 연산을 하는 경우도 십 분을 기다렸지만 연산이 끝나지 않았다. 이는 병렬로 8개의 연산을 처리하더라도, 데이터가 너무 커져버리면 O(n^3)의 알고리즘 한계상 연산 요구량이 기하급수적으로 늘어나 벡터 연산을 통한 성능 향상이 무의미하게 되기 때문일 것이다. 그래도 100개 이상, 4000개 이하 정도의 경우에는 속도 향상을 확실하게 체감할 수 있을 정도로 연산 주기가 짧아지므로, cpu가 벡터 연산을 지원하고, 적당히 큰 데이터를 계산해야 하는 경우는 벡터 연산을 사용하는 것이 합리적인 선택이라는 결론을 얻을 수 있었다.