[Lab-1] vector programming

소프트웨어학과 201720707 나용성

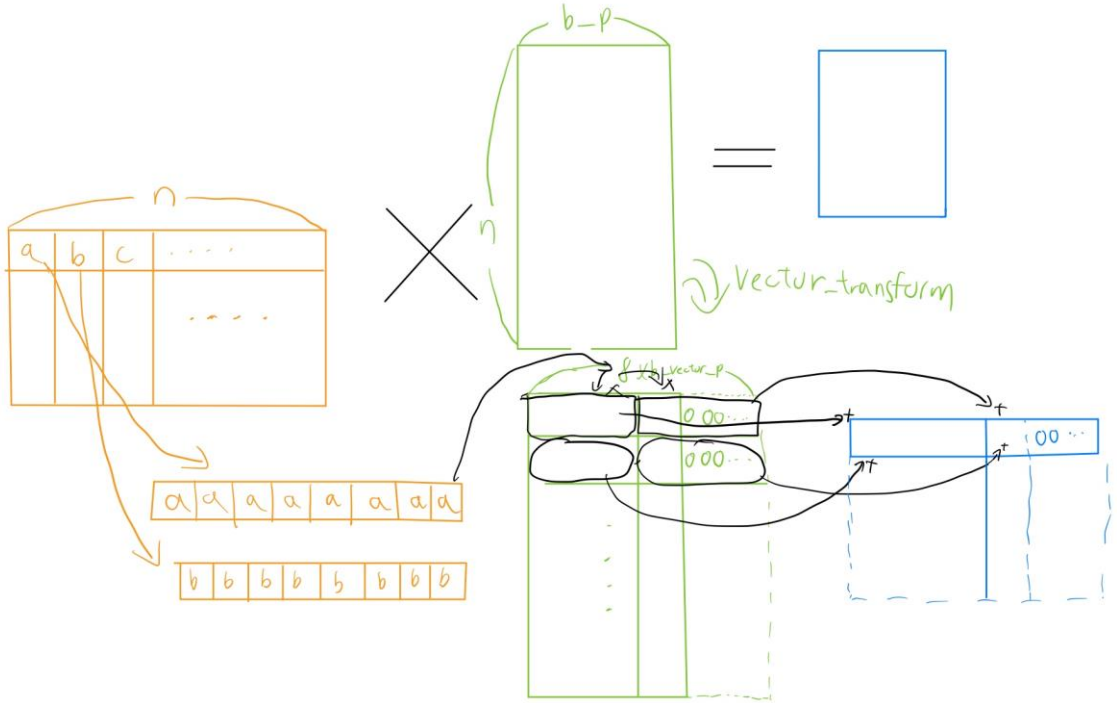
[아이디어설명]

구현의 아이디어는 과제 안내의 hint를 m*n 크기 행렬과 n*p 크기 행렬의 곱으로 일반화시키는 아이디어를 생각해보았다.

힌트에서는 곱해지는 행렬 중 첫번째 행렬의 각칸을 4칸짜리 벡터로 확장시켜 두번째 행렬의 알맞은 행과 곱한 행을 결과행렬에 더해가는 아이디어였다.

이 아이디어를 확장시키려면 먼저 a행렬의 칸은 각 칸을 확장시켜 쓰기 때문에 미리 처리하지 않아도 된다. 하지만 b행렬의 경우 열의 길이가 8의 배수가 아니면 __m256 벡터에 load할 수 없게 된다. 그러므로 b행렬의 열의 길이가 8의 배수가 되지 않으면 일종의 zero padding을 하여 열의 길이를 8의 배수로 맞춘다. 그 다음 b행렬의 각 행을 8칸 씩 벡터로 미리 저장하여 압축한다. 그 다음 a행렬의 각 칸을 8칸짜리 벡터로 확장하여 미리 처리된 b행렬로 만든 벡터행렬과 곱해 열과 행렬의 알맞은 위치에 더해나간다.

결과의 검증은 main.c파일의 전역변수로 미리 정의되어있는 두 행렬의 곱의 답을 저장해두었는데 이 행렬과 값이 다른 지 확인하여 검증한다.



위의 그림은 아이디어를 시각화한 모습이다. 기존의 b행렬에 zero padding한 뒤 vector화 시켜 vector 행렬로 압축시킨 뒤 a행렬의 각 칸을 확장시킨 뒤 확장된 칸의 열과 같은 인덱스의 b_vector 행렬의 행의 벡터들과 곱해 결과 행렬에 더해나가는 과정을 그리고 있다.

[성능비교]

```
st201720707@sce394-1:~$ make test1
gcc -mavx2 -o matmul main.c
./matmul -v 1
Elapsed time with AVX : 8043
correct!
```

벡터를 이용한 곱셈에서의 시간 측정 및 정답 여부 판별 결과 시간은 8043이 걸렸고 제대로 답을 알아냈음을 알 수 있다.

```
st201720707@sce394-1:~$ make test2
gcc -mavx2 -o matmul main.c
./matmul -v 2
Elapsed time with Non-AVX : 14427
correct!
```

스칼라를 이용한 곱셈에서도 마찬가지로 답은 제대로 계산해냈지만 곱셈에 걸린 시간은 14427이다.

벡터를 이용했을 때 스칼라에서 걸린 시간의 절반에 가까이 줄어들었다. 이론적으로는 __m256벡터 하나에 float 8개가 들어가고 한번에 연산되므로 처리속도가 8배가 될 수 있을 것 같지만 메모리에 read write하는데 시간이 많이 소비되기 때문에 그만큼의 성능향상은 되지 않았고 2배 가량의 성능향상이 이뤄졌다. 메모리에 저장하는 횟수를 줄일 수 있으면 더 많은 성능향상이 있을 수 있다. 예를들어 더 큰 벡터를 사용할 수 있다면 read write를 하는 횟수가 줄어들 수 있을 것이다.