Midterm Project Report

2023021577 서상현

1. 개요

GEMM(General Matrix Multiplication)은 [그림 1]과 같이 두 행렬의 곱셈을 뜻하며 AI 모델이 가장 많이 수행하는 연산으로 $O(n^3)$의 시간 복잡도를 가지고 있다.

$$\begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1k} \\ A_{21} & A_{22} & \cdots & A_{2k} \\ \vdots & \vdots & \vdots & \vdots \\ A_{m1} & A_{m2} & \cdots & A_{mk} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} & \cdots & B_{1n} \\ B_{21} & B_{22} & \cdots & B_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ B_{k1} & B_{k2} & \cdots & B_{kn} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & \cdots & C_{1n} \\ C_{21} & C_{22} & \cdots & C_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ C_{m1} & C_{m2} & \cdots & C_{kn} \end{bmatrix}$$

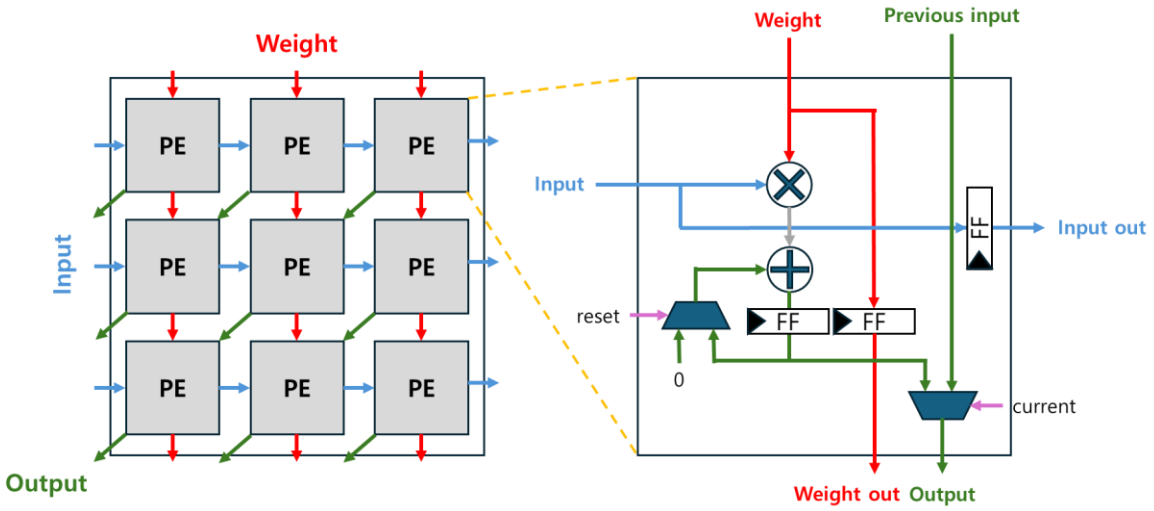
$$C_{11} = A_{11}B_{11} + A_{12}B_{21} + \ldots + A_{1k}B_{k1}$$

[그림 1]

Systolic array와 systolic tensor array는 모두 GEMM 연산을 가속하는 하드웨어이다. 특히, systolic tensor array는 systolic array보다 더욱 data reuse와 병렬성을 증가시킨 구조이다. 두 가속기는 PE(Processing Element)에 weight를 고정하는 방식인 weight stationary와 output을 누정하는 output stationary 방식으로 나뉜다. 본 프로젝트에서는 output stationary 방식의 systolic array와 systolic tensor array를 구현한다. Scala 언어를 사용하는 오픈소스 HLS(High Level Synthesis)인 Chisel을 이용하여 구현하며 테스트 코드를 작성해 구현한 하드웨어가 정상적으로 GEMM 연산을 수행하는지 확인한다.

2. 구현 내용

- Systolic array 구조

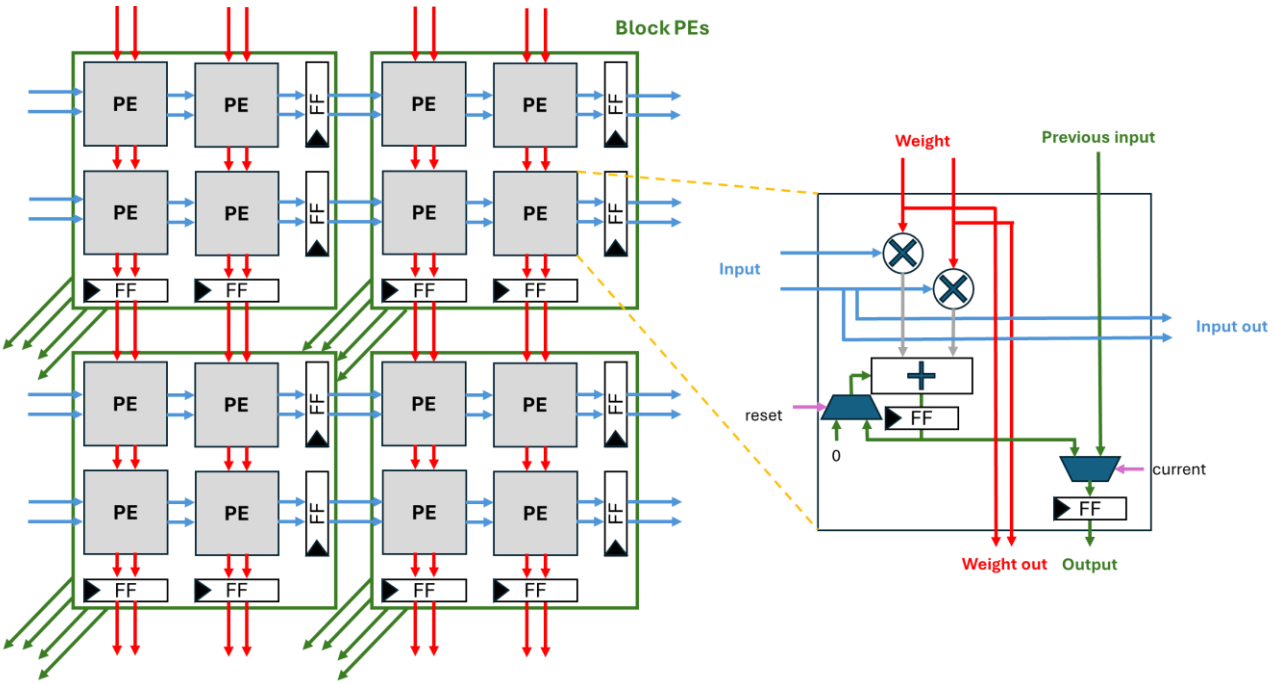


[그림 2]

[그림 2]는 구현한 systolic array의 구조이다. Systolic array는 2개의 입력을 받아 곱셈 결과를 누적하는 PE들이 연결되어 있다. PE는 5개의 입력과 2개의 출력을 가지고 있다. 각 signal은 다음과 같다.

a. Input : 곱셈의 피연산자로 flipflop을 통해 다음 사이클에 같은 값을 통과시킨다.
b. Weight : 곱셈의 피연산자로 input과 같이 flipflop을 통해 다음 사이클에 같은 값을 통과시킨다.
c. Reset : 곱셈 연산 결과를 초기화할지 선택하는 signal로 1일 경우 곱셈 결과와 0을 더함으로써 누적된 값을 초기화한다. 0일 경우 누적된 값에 곱셈 결과를 더한다.
d. Current: 출력 결과를 선택하는 signal로 1일 경우 현재 PE의 결과를 출력하고 0일 경우 previous output을 출력한다.
e. Previous output : 대각선에 위치하는 PE의 output.
f. Input out : flipflop을 통해 통과된 input.
g. Weight out : flipflop을 통해 통과된 weight.
h. Output : previous output과 현재 PE의 output 중 current signal에 의해 선택된 결과이다.

- Systolic tensor array 구조



[그림 3]

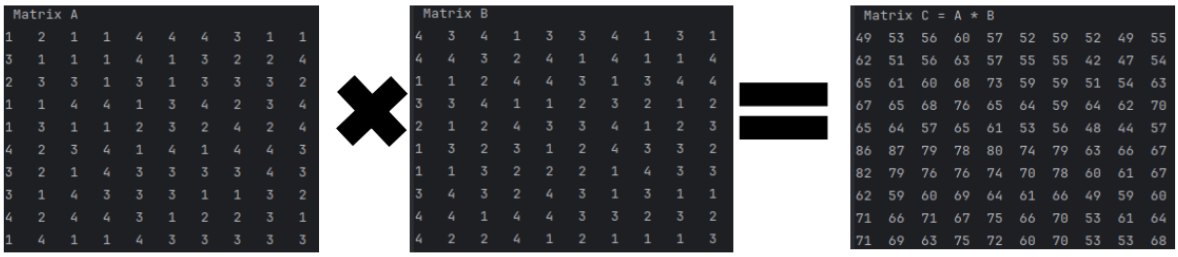
[그림 3]은 직접 구현한 systolic tensor array의 구조이다. Systolic tensor array는 systolic array와 거의 비슷한 구조를 가지고 있지만 하나의 PE에 더욱 많은 입력을 넣고 여러 PE를 block으로 묶어서 더욱 빠르게 행렬 곱셈을 수행할 수 있는 구조이다. [그림 3] 왼쪽의 초록색 상자는 4개의 PE를 묶은 block을 나타내며 하나의 block은 한 cycle에 동시에 연산을 수행할 수 있다.

그림의 오른쪽과 같이 PE는 여러 입력을 받을 수 있으며 systolic array의 PE와 다른 점은 input out, weight out이 flipflop을 거치지 않고 통과해 block 내에서 같은 행의 PE는 같은 input을, 같은 열의 PE는 같은 weight를 가지게 된다. 또한 block 내의 PE들은 reset, current signal을 공유하며 동시에 output을 출력한다.

3. Test Code

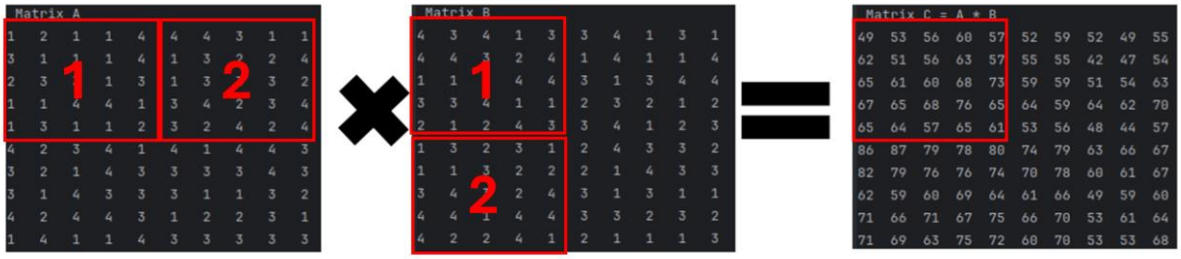
- Systolic array test

Systolic array는 [그림 4]의 행렬 곱셈을 수행한다. Systolic array는 PE의 개수를 configurable하게 구현하였으며 5x5 PE를 가지는 systolic array에서 아래 행렬을 입력으로 넣어 결과를 출력하도록 test code를 작성하였다.



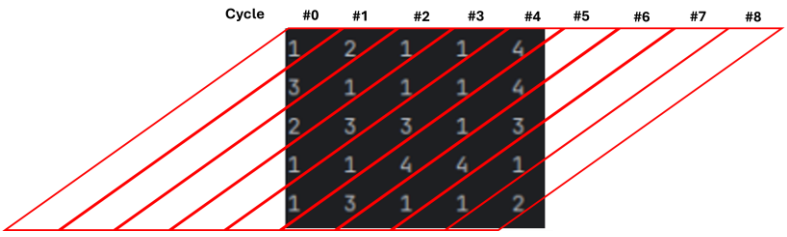
[그림 4]

Test 행렬은 10x10이기 때문에 5x5 행렬로 나누어 clock 타이밍에 맞추어 입력을 넣어준다. matrixmultpiy함수는 [그림 5]와 같이 5x10과 10x5 행렬을 입력하여 5x5 결과를 출력하는 것을 테스트하는 함수이다. 예시로 5x5 결과를 출력한다. 그림과 같이 우선 1로 표시된 5x5 행렬을 입력한 후 PE를 초기화하지 않고 2번째 5x5 행렬을 입력한다. 그리고 차례에 맞추어 그림 오른쪽의 결과를 확인할 수 있다.



[그림 5]

5x5 행렬에서의 input은 [그림 6]과 같이 대각선이 하나의 입력이 한 사이클에 입력으로 들어간다. 따라서 5x5 행렬이 모두 입력되는데 총 9사이클이 걸리며 1번 PE는 #4까지만 입력을 받으며 #5에 누적연산결과를 가지게 되고 #6에 결과를 출력할 수 있다. Weight의 동일하게 대각선이지만 가로로 입력된다.

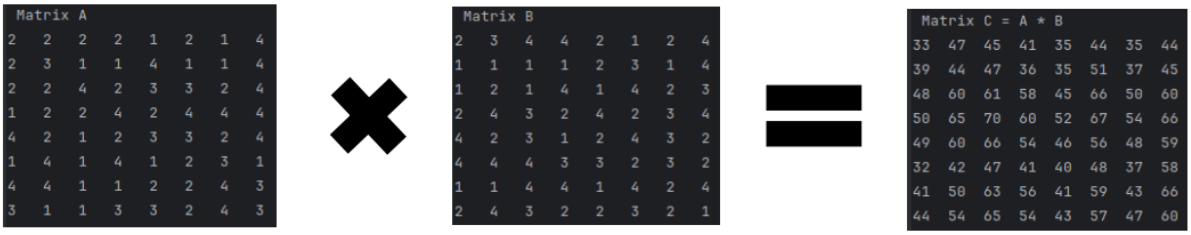


[그림 6]

결과는 해당 연산이 종료되는 cycle에 곧바로 출력하였다. 따라서 cycle #15에 5x10 10x5 행렬 곱셈의 첫 번째 결과가 출력된다.

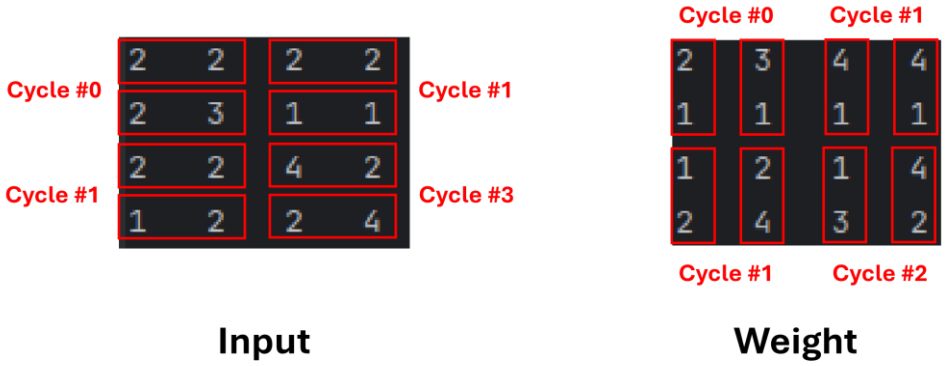
- Systolic tensor array test

Systolic tensor array는 [그림 7]을 통해 검증한다. Systolic tensor array의 configuration은 {2x2}x{2x2}x2로 설정하였기 때문에 input과 weight의 입력이 한 cycle에 8개 들어가지만 systolic array와 약간 다른 입력 방식을 이용하기 때문에 4x4로 쪼개어 data를 feeding한다.



[그림 7]

Systolic tensor array는 systolic array와 다른 방식으로 data를 입력한다. input과 weight는 [그림 8]과 같이 입력된다. 따라서 4x4의 첫 번째 output은 #2에 연산을 끝낸 후 #3에 출력한다.



[그림 8]

- 실행 결과

두 가속기 모두 clock cycle에 맞추어 데이터 입력 후 연산이 끝나는 타이밍에 맞추어 expect() 메소드를 이용해 예상한 결과와 일치하는지 확인한다. [그림 9] 왼쪽은 Systolic array test 실행 결과를, 오른쪽은 systolic tensor array test 실행 결과 성공적으로 끝난 것을 보여준다.

[그림 9]