병렬처리 Final Exam

조교 김진표, 박대영, 신준식, 정재훈

pp-ta@aces.snu.ac.kr

THUNDER Research Group
Seoul National University
서울대학교 천둥 연구실

Final Exam

- 기존 과제와 같은 방식으로 진행
- 2D Convolution 연산을 CPU 및 GPU 를 이용해 최적화
  - 이미지/영상 처리 분야에 흔하게 사용되는 연산

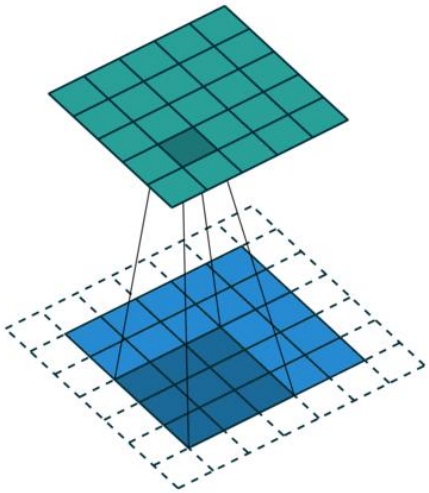
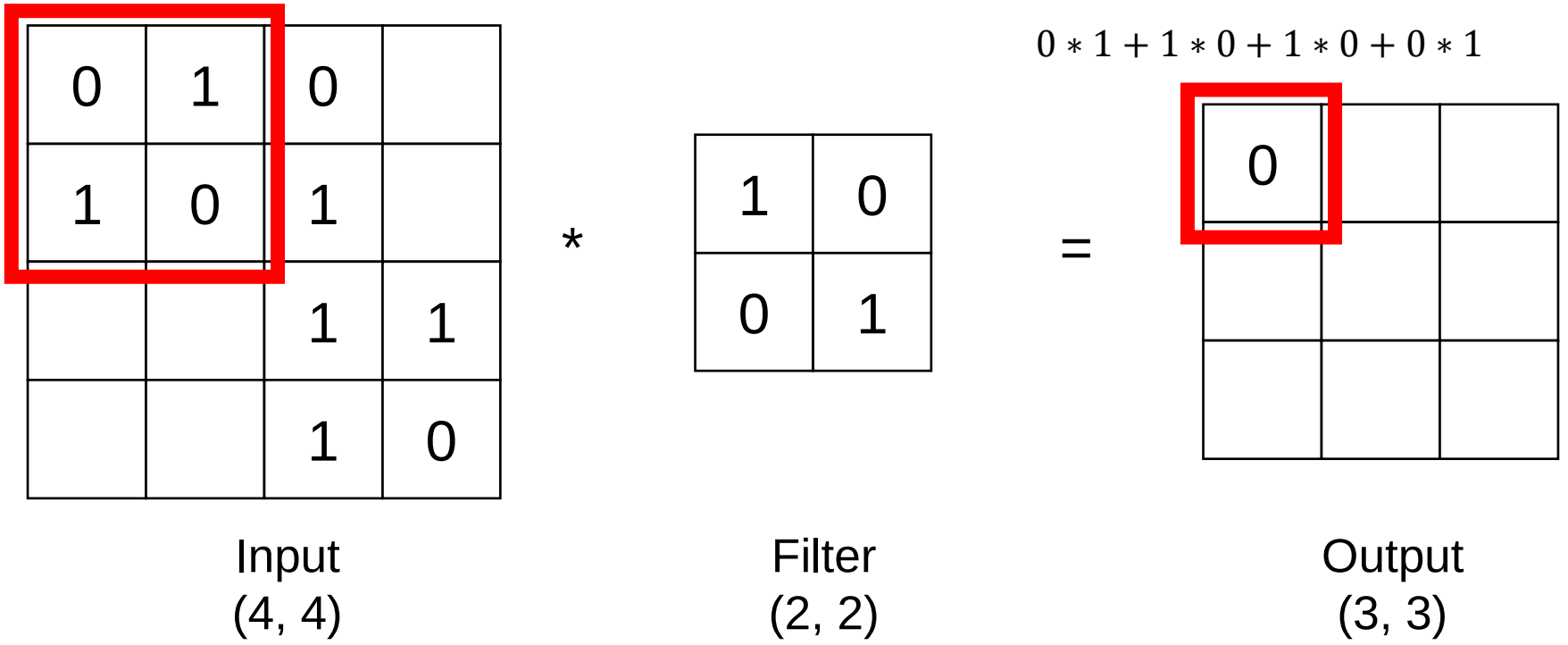


Image from https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

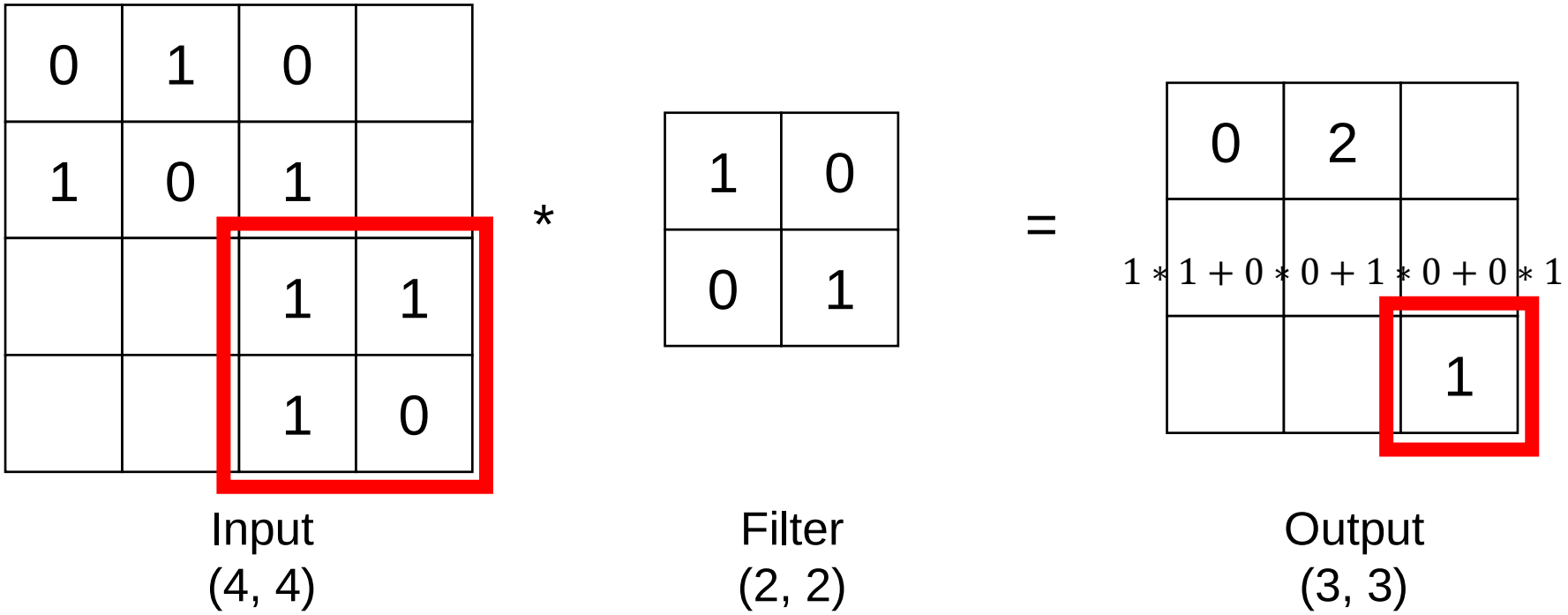
2D Convolution

- 필터를 입력 위에서 움직이면서 내적 값을 계산
  - 특정 영역이 필터와 유사하다면, 대응되는 출력 값이 높게 나옴



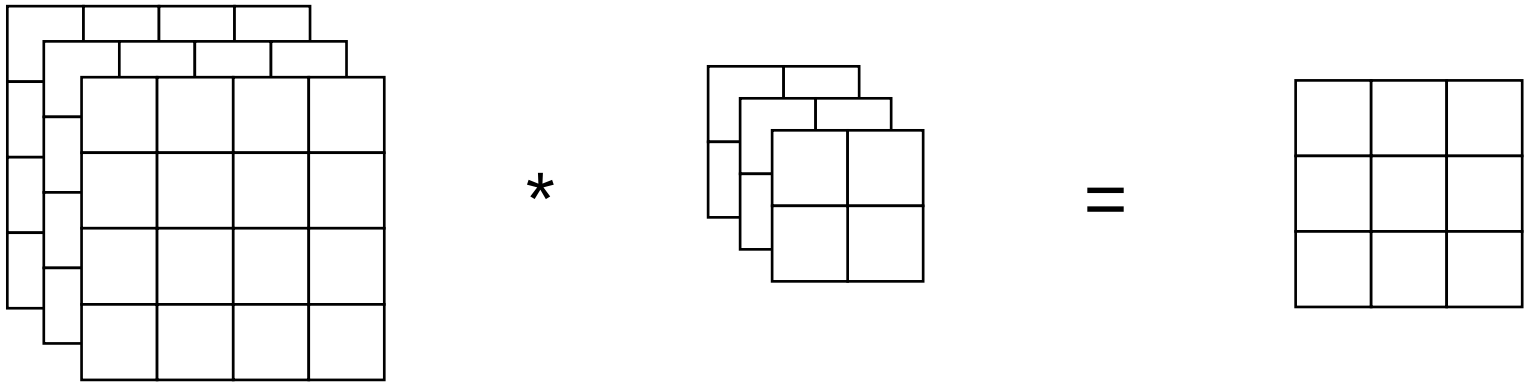
2D Convolution

- 필터를 입력 위에서 움직이면서 내적 값을 계산
  - 특정 영역이 필터와 유사하다면, 대응되는 출력 값이 높게 나옴



2D Convolution

- 입력의 채널이 여러 개라면?



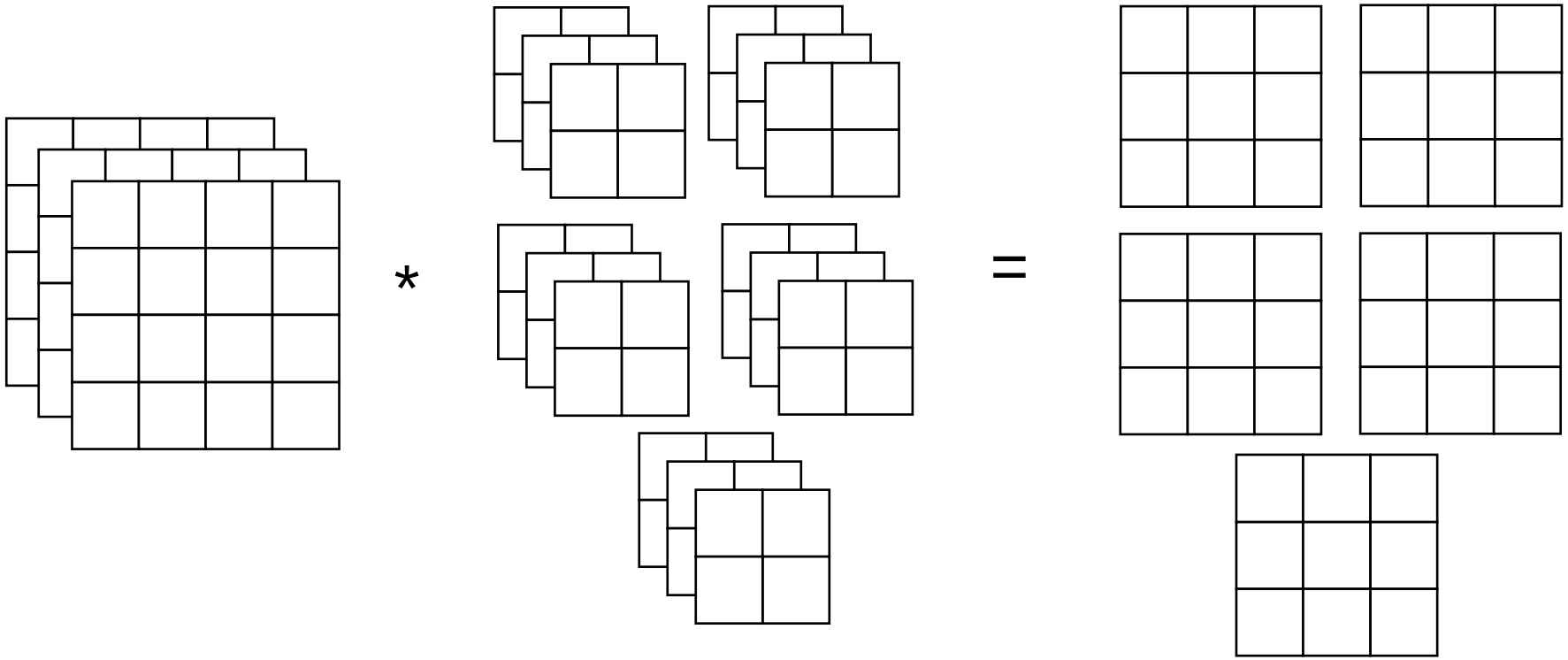
Input
(C, H, W) = (3, 4, 4)

Filter
(C, R, S) = (3, 2, 2)

Output
(OH, OW) = (3, 3)

2D Convolution

- 출력의 채널이 여러 개라면?



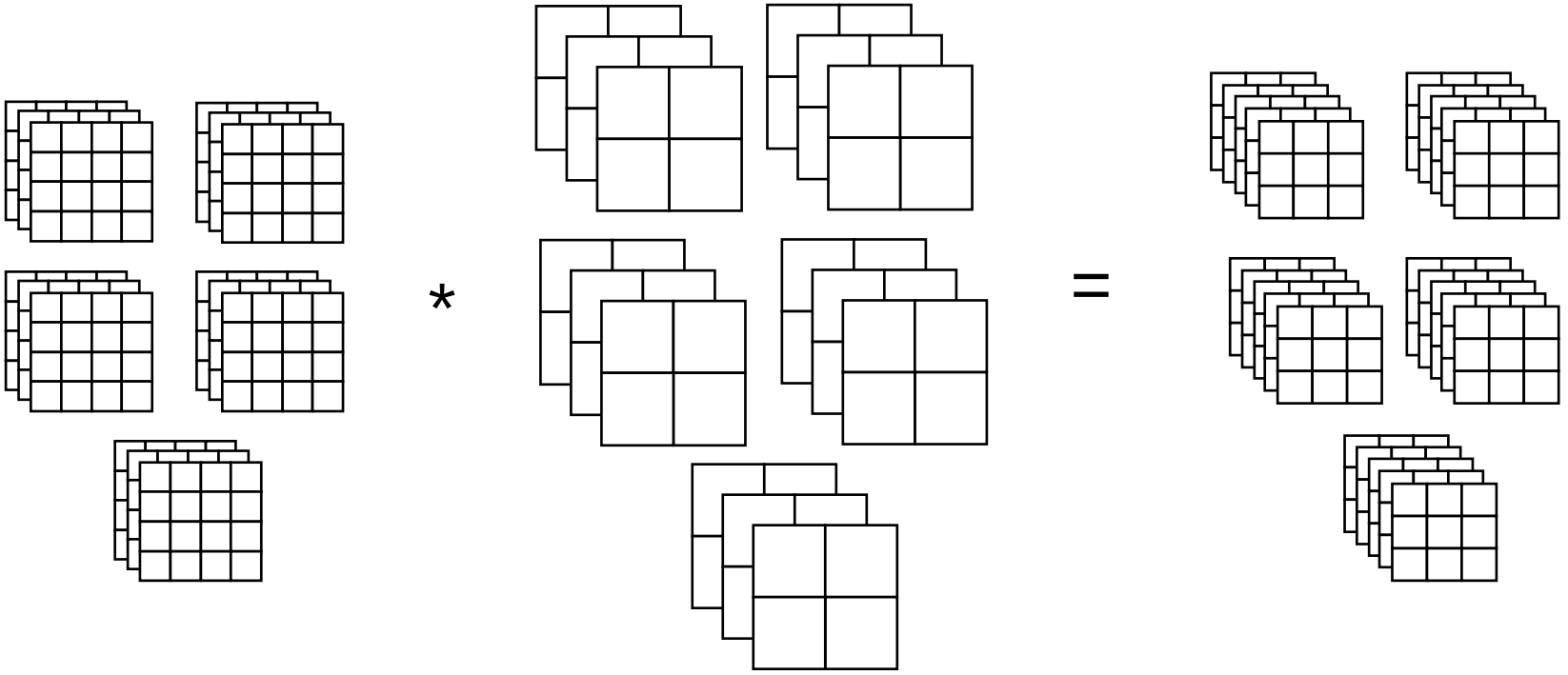
Input
(C, H, W) = (3, 4, 4)

Filter
(K, C, R, S) = (5, 3, 2, 2)

Output
(K, OH, OW) = (5, 3, 3)

2D Convolution

- 입력이 여러 장이라면?

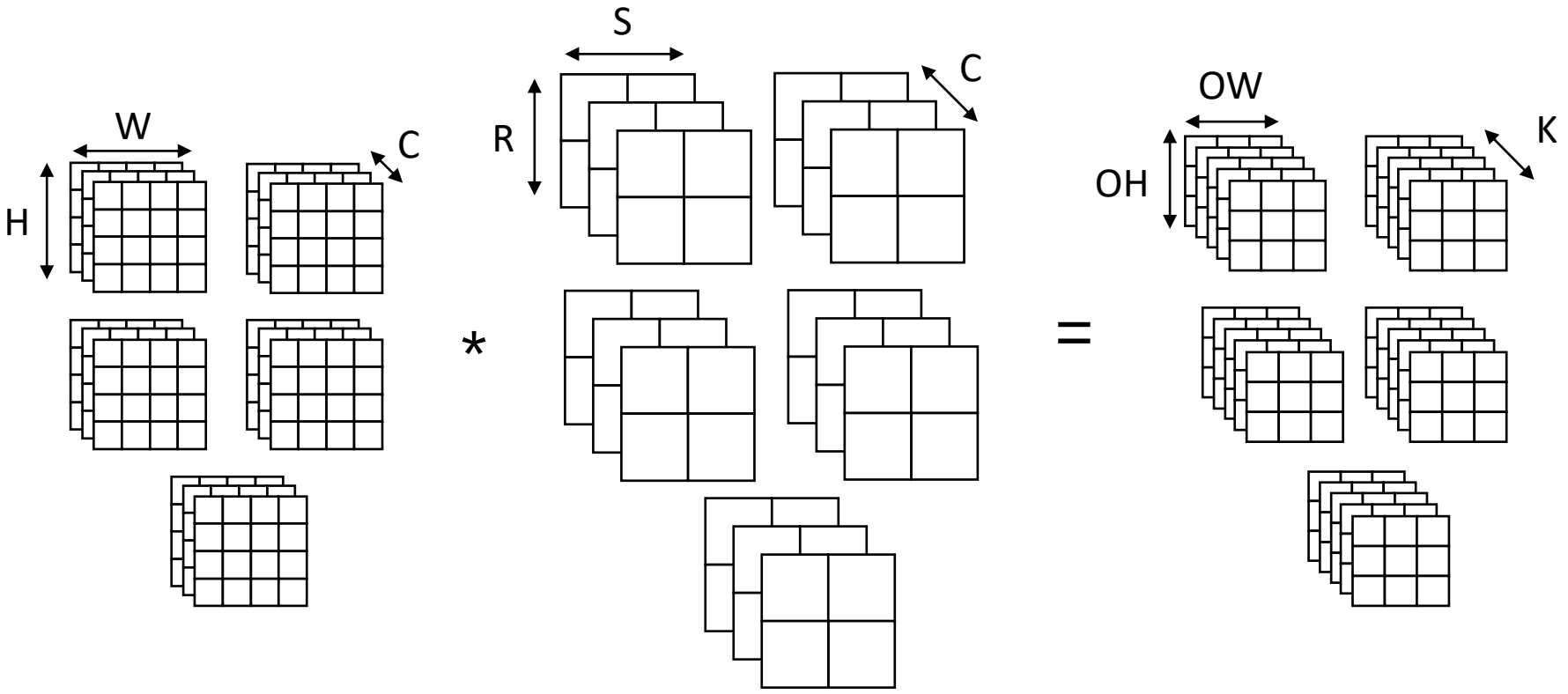


Input
(N, C, H, W) = (5, 3, 4, 4)

Filter
(K, C, R, S) = (5, 3, 2, 2)

Output
(N, K, OH, OW) = (5, 5, 3, 3)

2D Convolution



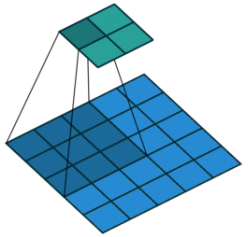
Input
(N, C, H, W) = (5, 3, 4, 4)

Filter
(K, C, R, S) = (5, 3, 2, 2)

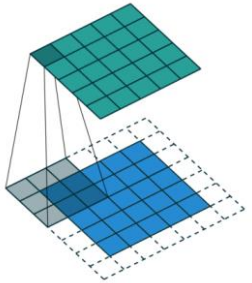
Output
(N, K, OH, OW) = (5, 5, 3, 3)

Convolution Variations

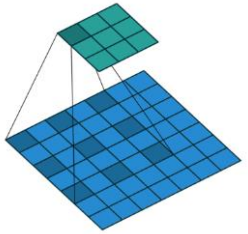
- Stride
  - 필터가 움직이는 간격



- Pad
  - 입력 가장자리에 0을 추가하여 출력 크기 조절



- Dilation
  - 필터 원소 하나하나 사이의 간격



- 이해에 도움이 되는 자료
  - https://github.com/vdumoulin/conv_arithmetic

2D Convolution

- OH 및 OW 계산 방법

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

From https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

목표

- convolution.cpp 를 병렬화 하여 빠르게 하는 것

- A: 계산 노드 2개의 CPU 를 사용
  - 2개 노드의 모든 CPU 코어 사용 가능

- B: 계산 노드 2개의 GPU 를 사용
  - 2개 노드의 모든 GPU 사용 가능

뼈대 코드

- /skeleton/final 에 주어진 뼈대 코드를 각자의 홈 디렉토리로 복사해 작업
  - `cp -r /skeleton/final ~/`
  - `Makefile, run.sh, convolution.cpp …`
  - `make` 로 컴파일이 가능하도록 `Makefile` 을 제공
  - 실행을 위한 `run.sh` 스크립트를 제공

- 뼈대 코드를 수정해 convolution 연산을 최적화하는 것이 목표
  - 파일 추가 및 Makefile 수정 가능. 파일 이름 변경도 가능
    - kernel.cl, convolution.cu 등
  - main.cpp, util.cpp, util.h convolution.h 는 수정 불가능
  - make 로 컴파일 되고, make clean 으로 object/executable 파일들이 지워져야 함

2D Convolution - 실행

- make 로 컴파일
- `run.sh` 으로 실행

```
kjp4155@login:~/final$ ./run.sh -h
salloc: Granted job allocation 74962
Usage: ./main [-pvh]  [-n num_iterations] N C H W K R S
Options:
  -h : print this page.
  -v : validate convolution. (default: off)
  -n : number of iterations (default: 1)
  -p : padding size. (default: 0)
  -d : dilation size. (default: 0)
  -s : stride size. (default: 1)
   N : number of images, i.e., batch size. (default: 8)
   C : channel dimension. (default: 8)
   H : image height. (default: 8)
   W : image width. (default: 8)
   K : number of filters. (default: 8)
   R : filter height. (default: 3)
   S : filter width. (default: 3)
salloc: Relinquishing job allocation 74962
kjp4155@login:~/final$
```

2D Convolution - 실행

- make 로 컴파일
- `run.sh` 으로 실행

```
kjp4155@login:~/final$ ./run.sh -v -n 5 -p 2 -d 2 -s 3 8 16 256 256 3 8 8
salloc: Granted job allocation 74990
Options:
  Input size: N = 8, C = 16, H = 256, W = 256
  Output size: N = 8, K = 3, OH = 82, OW = 82
  Filter size: K = 3, C = 16, R = 8, S = 8
  Number of iterations: 5
  Validation: on

Initializing... done!
Initializing Convolution...
Calculating...(iter=0) 0.209871 sec
Calculating...(iter=1) 0.209470 sec
Calculating...(iter=2) 0.211528 sec
Calculating...(iter=3) 0.213077 sec
Calculating...(iter=4) 0.209907 sec
Validating...
Result: VALID
Avg. time: 0.210771 sec
salloc: Relinquishing job allocation 74990
kjp4155@login:~/final$
```

2D Convolution - 평가 기준

- A (50점)
  - 정확성 점수 (25점)
    - 무작위 인자들에 대해 -v 옵션을 주어 값 검증을 통과해야 함
    - 총 10개의 test case
  - 성능 점수 (25점)
    - 상대 평가. 수강생 중 가장 좋은 성능을 만점으로 하고, 성능이 2배 떨어질 때마다 10% 감점.
    - ex. 1등 프로그램이 1초가 걸린다면, 8초가 걸리는 프로그램은 성능 점수의 70% 부여
    - 적당히 큰 workload로 성능을 측정할 예정
- B (50점)
  - A 와 같음

제약 조건

- 배운 병렬화 방법 모두 사용 가능
  - A: Pthread, OpenMP, MPI
  - B: OpenCL, CUDA, MPI
- 외부 라이브러리 사용 금지
  - Intel MKL, clBLAS, cuDNN, cuBLAS 등 ...
- 새 파일을 추가 하거나 이름 변경 가능
  - kernel.cl 추가, convolution.cpp를 convolution.cu로 변경 등
  - Makefile 로 잘 컴파일 및 실행 되도록 유의
  - main.cpp, util.cpp, util.h convolution.h 는 수정 불가능
  - 부득이하게 수정이 필요하다 생각되는 경우 문의

제약 조건

- 메모리 레이아웃 변경, 루프 순서 변경, 패딩 데이터/연산 추가 등의 최적화 기법은 허용
- 연산량을 줄이는 변경은 어떤 것이든 불허
  - 계산 복잡도가 다른 알고리즘을 써 연산의 양을 줄이거나, 결과에 영향이 적을 만한 연산을 제외하는 등
- 애매한 것은 조교에게 문의할 것

제약 조건

- `convolution_init()`
  - memcpy 불가
  - cudaMalloc 등 메모리 할당만 가능

- `convolution()`
  - 모든 노드간/노드내 통신과 계산을 수행

제출 방법

- 각자의 홈 디렉토리에 `~/submit/final/A` 및 `~/submit/final/B` 디렉토리 생성
- 구현한 파일들을 디렉토리 내에 복사

- 애초에 제출 디렉토리에서 작업하는 것도 가능
  - 제출기한 전에 코드 정리를 할 것 (디버깅 코드 삭제 등)

- check-submit 유틸리티 활용

```
kjp4155@login:~/final$ check-submit
Username: kjp4155
------------------------------------
HW1 prob1.txt   : X file not found
HW1 prob2.txt   : OK
HW1 main.cpp    : X empty file
------------------------------------
HW2 mat_mul.cpp : OK
------------------------------------
HW4 mat_mul.cpp : OK
------------------------------------
HW5 mat_mul.cpp : OK
HW5 kernel.cl   : OK
------------------------------------
HW6 mat_mul.cu  : X file not found
------------------------------------
------------------------------------
Final-A: ['run.sh', 'main.cpp', 'util.cpp', 'convolution.cpp', 'convolution.h', 'Makefile', 'util.h']
Final-B:
```

주의사항

- **제출 기한: 5월 18일 (수) 23:29**
  - 기한이 되면 프로그램이 자동으로 제출 파일을 복사해 감
  - 파일명, 디렉토리, 형식을 정확히 지켰는지 체크

- **조교가 실수를 수정해줄 것이라 기대하지 않을 것**
  - 제출 디렉토리 확인
  - 제출 파일명 확인
  - 제출한 파일 내용 확인
  - 프로그램이 불필요한 출력을 하는 지 확인
  - 디버깅 코드 등 불필요한 코드를 모두 제거했는지 확인

5/11 추가

- 뼈대 코드의 main.cpp 에 avg. time 대신 avg. FLOPS 를 출력하도록 변경
- 뼈대 코드의 main.cpp 에 dilation 의 기본값을 1로 변경

- 정확성 평가 시 입력 제약 조건
  - dilation: 1~5
  - padding: 0~10
  - stride: 1~5
  - Node 개수: 1~2
  - N, C, H, W, K, R, S: 1~1024

5/11 추가

- 성능 평가 시 입력 제약 조건
  - dilation: 1
  - padding: 0
  - stride: 1
  - Node 개수: 2
  - N, C, K, H, W: 32 이상의 적당히 큰 2의 지수승
  - R, S: 16

- 향후 공지사항은 본 슬라이드에 추가할 예정

5/11 추가

- 뼈대 코드 main.cpp 오류 수정

- Hint
  - 행렬 곱에서 C를 행 기준으로 나누었던 것처럼 convolution 의 출력을 나누어서 처리하기
  - N 기준으로 나누고, 부족하다면 K 도 나누면 될 것

5/13 추가

- Reference 성능
  - 작년 학부 수업에서 1등을 한 학생의 convolution 커널을 가져와 돌린 성능
  - `./run.sh -n 10 32 64 256 256 64 16 16`
  - 2Node, 8GPU 기준 약 8000 GFLOPS

- 1등의 성능이 너무 높은 경우 성능 평가 기준을 완화할 수 있음

5/13 추가

- Vector io 예시 코드
  - `/skeleton/vector_io`
  - make 로 컴파일, ./run.sh 로 실행
  - main.cpp 에 세 가지 방식의 vector copy 연산이 구현되어 있음
  - vector io 를 사용한 경우 vector copy 성능이 향상되는 것을 확인할 수 있음

```
Max # blocks: 1073741824
[   0.01 MB] scalar: 0.842105 GB/s, vector2: 1.280000 GB/s, vector4: 1.230769 Gb/s
[   0.02 MB] scalar: 2.560000 GB/s, vector2: 2.560000 GB/s, vector4: 2.560000 Gb/s
[   0.03 MB] scalar: 5.120000 GB/s, vector2: 4.923077 GB/s, vector4: 5.120000 Gb/s
[   0.06 MB] scalar: 10.240000 GB/s, vector2: 10.240000 GB/s, vector4: 10.240000 Gb/s
[   0.12 MB] scalar: 24.380952 GB/s, vector2: 24.380952 GB/s, vector4: 20.480000 Gb/s
[   0.25 MB] scalar: 39.384615 GB/s, vector2: 48.761905 GB/s, vector4: 40.960000 Gb/s
[   0.50 MB] scalar: 81.920000 GB/s, vector2: 81.920000 GB/s, vector4: 97.523810 Gb/s
[   1.00 MB] scalar: 141.241379 GB/s, vector2: 163.840000 GB/s, vector4: 195.047619 Gb/s
[   2.00 MB] scalar: 240.941176 GB/s, vector2: 282.482759 GB/s, vector4: 273.066667 Gb/s
[   4.00 MB] scalar: 356.173913 GB/s, vector2: 481.882353 GB/s, vector4: 655.360000 Gb/s
[   8.00 MB] scalar: 431.157895 GB/s, vector2: 390.095238 GB/s, vector4: 564.965517 Gb/s
[  16.00 MB] scalar: 489.074627 GB/s, vector2: 579.964602 GB/s, vector4: 601.247706 Gb/s
[  32.00 MB] scalar: 496.484848 GB/s, vector2: 652.099502 GB/s, vector4: 693.502646 Gb/s
[  64.00 MB] scalar: 520.126984 GB/s, vector2: 652.099502 GB/s, vector4: 734.296919 Gb/s
[ 128.00 MB] scalar: 533.898167 GB/s, vector2: 657.826851 GB/s, vector4: 762.046512 Gb/s
[ 256.00 MB] scalar: 541.060888 GB/s, vector2: 661.562145 GB/s, vector4: 773.856827 Gb/s
[ 512.00 MB] scalar: 544.008301 GB/s, vector2: 665.762540 GB/s, vector4: 781.353204 Gb/s
[   1.00 GB] scalar: 545.281331 GB/s, vector2: 667.139176 GB/s, vector4: 782.373438 Gb/s
[   2.00 GB] scalar: 546.168891 GB/s, vector2: 666.873996 GB/s, vector4: 786.481155 Gb/s
[   4.00 GB] scalar: 546.453521 GB/s, vector2: 665.788960 GB/s, vector4: 788.180776 Gb/s
[   8.00 GB] scalar: 546.863196 GB/s, vector2: 666.834238 GB/s, vector4: 788.403008 Gb/s
[  16.00 GB] scalar: 546.970169 GB/s, vector2: 667.079492 GB/s, vector4: 788.727320 Gb/s
kjp4155@login:~/vector_io$
```