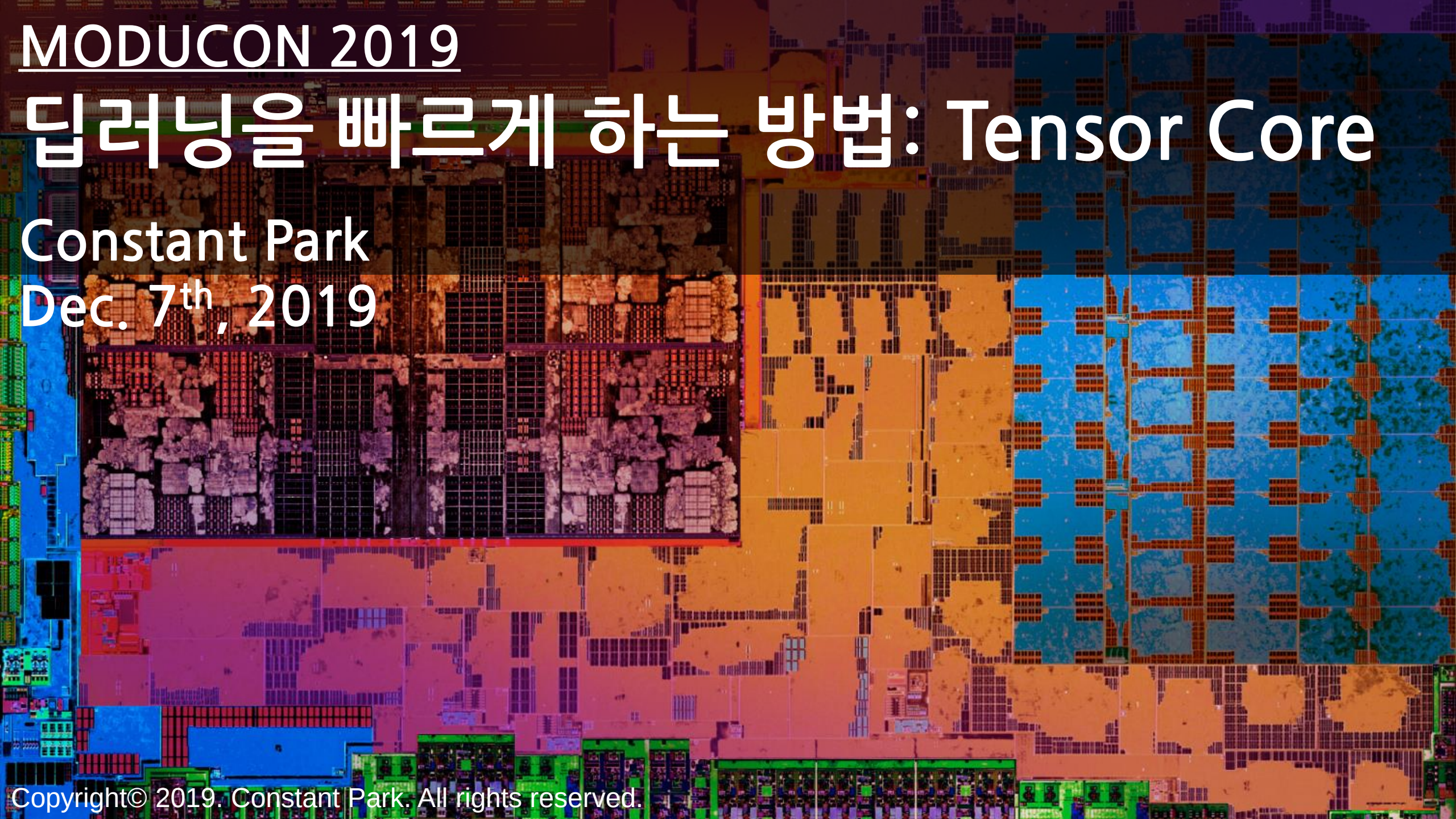
MODUCON 2019

딥러닝을 빠르게 하는 방법: Tensor Core

Constant Park

Dec. 7th, 2019



음.. 저는 이런 사람입니다

- **Sang-Soo Park (박상수)**
  - https://constantpark.github.io/ & sonicstage12@naver.com
  - Ph.D. candidate student in Hanyang university
- **Community Activities**
  - 풀잎스쿨 (CUDA를 활용한 딥러닝 가속) 퍼실이 (19.07 ~ 19.09)
  - **Neural Acceleration Lab** 랩장 (19.11 ~)
- **Research Area**
  - Neural Acceleration Optimization (CPU/GPU)
  - **NPU & SoC Design** (RISC-V based)
- **Project completed and doing**
  - Deep Learning Framework for Home Appliance
  - DIMM based Acceleration for Recommendation System

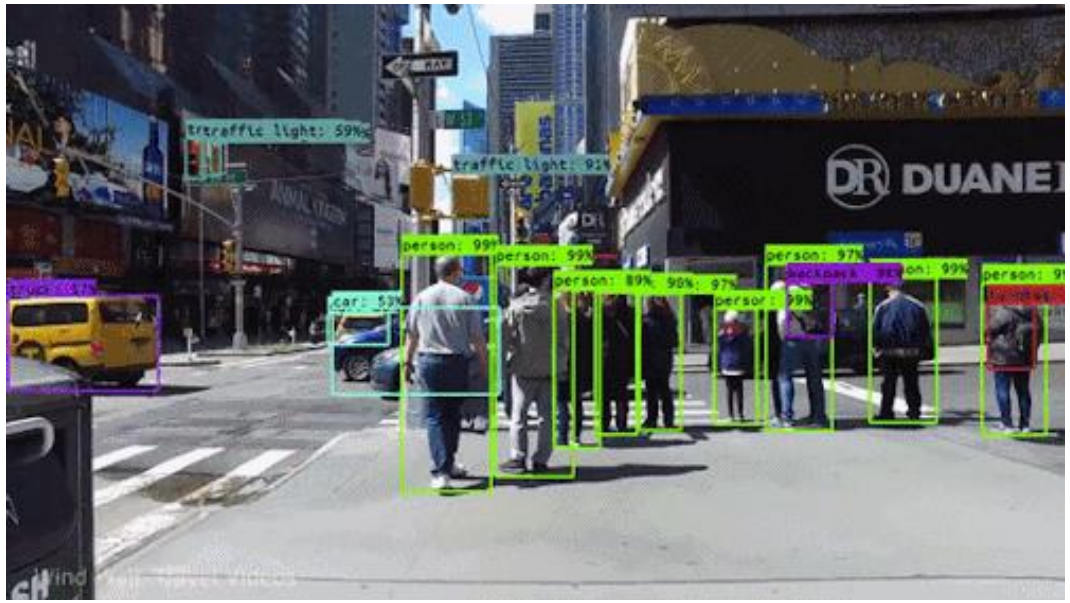
이 강의는 딥러닝 프레임워크를 사용해본 사람을 대상으로 합니다.

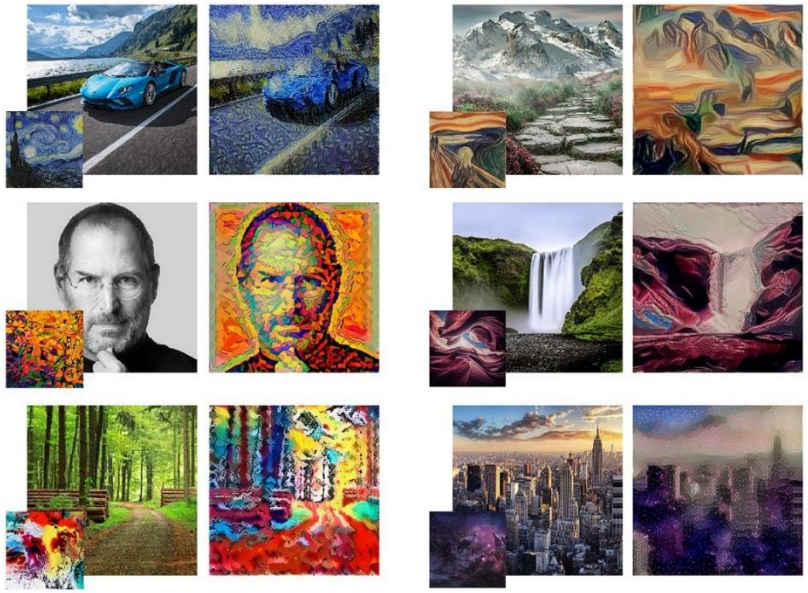
목차

- **딥러닝과 행렬 곱셈**
- 딥러닝 가속을 위한 다양한 가속 시스템
- GPU에서의 딥러닝 가속 방법: CUDA 코어/텐서 코어
- 정리

딥러닝 (Deep Learning)

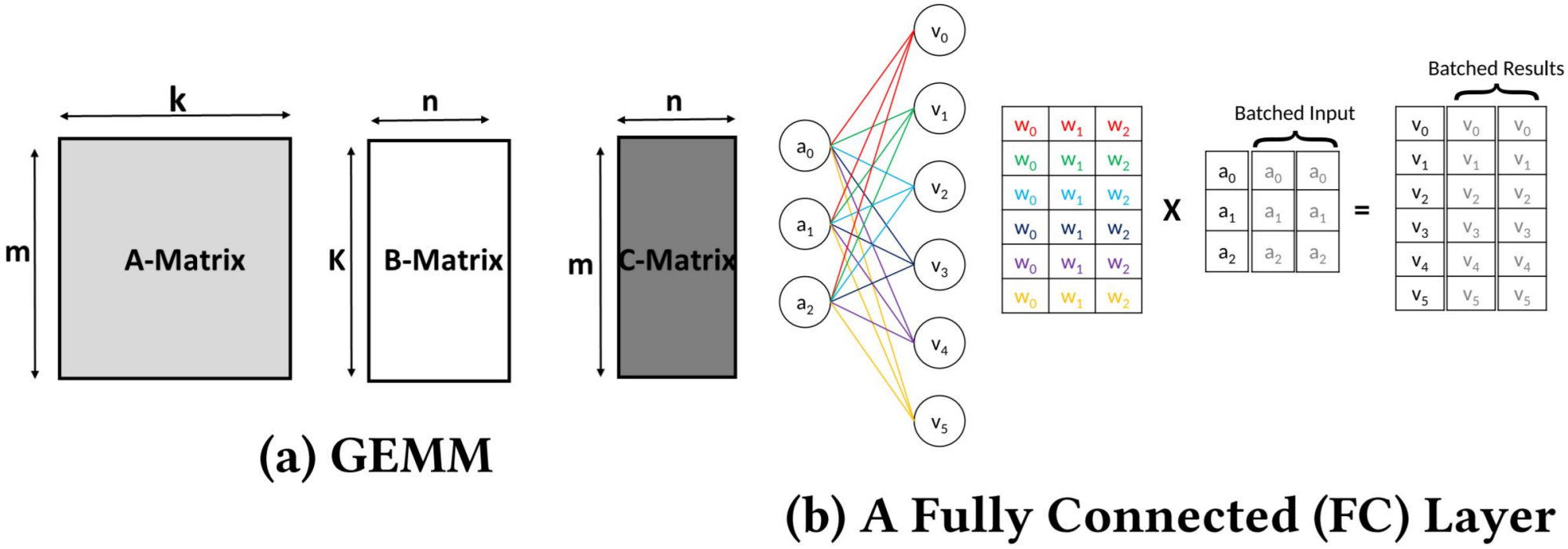
- **다양한 문제를 해결하는데 효과적인 방법**
  - **Convolutional neural network (CNN):** 컴퓨터 비전, 자율주행차
  - **Recurrent neural network (RNN):** 자연어 처리, 번역 시스템
  - **Generative adversarial network (GAN):** 이미지 생성 (예: Style transfer)





딥러닝 (Deep Learning) 모델을 계산하는 방법

- **행렬 곱셈: 딥러닝을 컴퓨터에서 계산하고 구현하는 방법**
  - General matrix multiplication (GEMM): 일반적인 행렬 곱셈
  - Basic linear algebra subprograms (BLAS): 행렬을 계산하기 위한 API
  - 딥러닝은 BLAS를 사용하여 GEMM을 계산
    - Convolution, Recurrent, Fully-connected 레이어는 행렬의 형태로 변환되어 계산

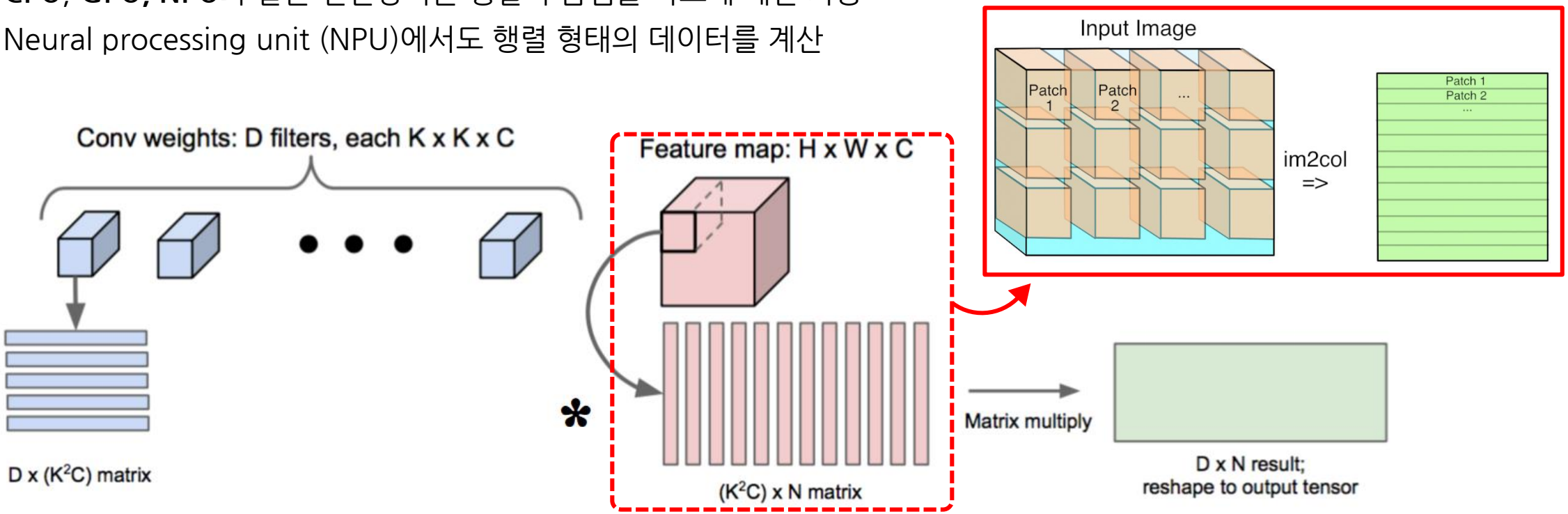


딥러닝 (Deep Learning)을 계산하는 방법

CNN을 계산하는 방법

Image to colmun (Im2col)

- 입력 이미지를 **행렬 형태**로 변환하는 방법
- **Patch**: 컨볼루션 커널이 이동하면서 feature map에 대응되는 부분 (receptive field)
- **CPU, GPU, NPU**와 같은 연산장치는 행렬의 곱셈을 빠르게 계산 가능
- Neural processing unit (NPU)에서도 행렬 형태의 데이터를 계산

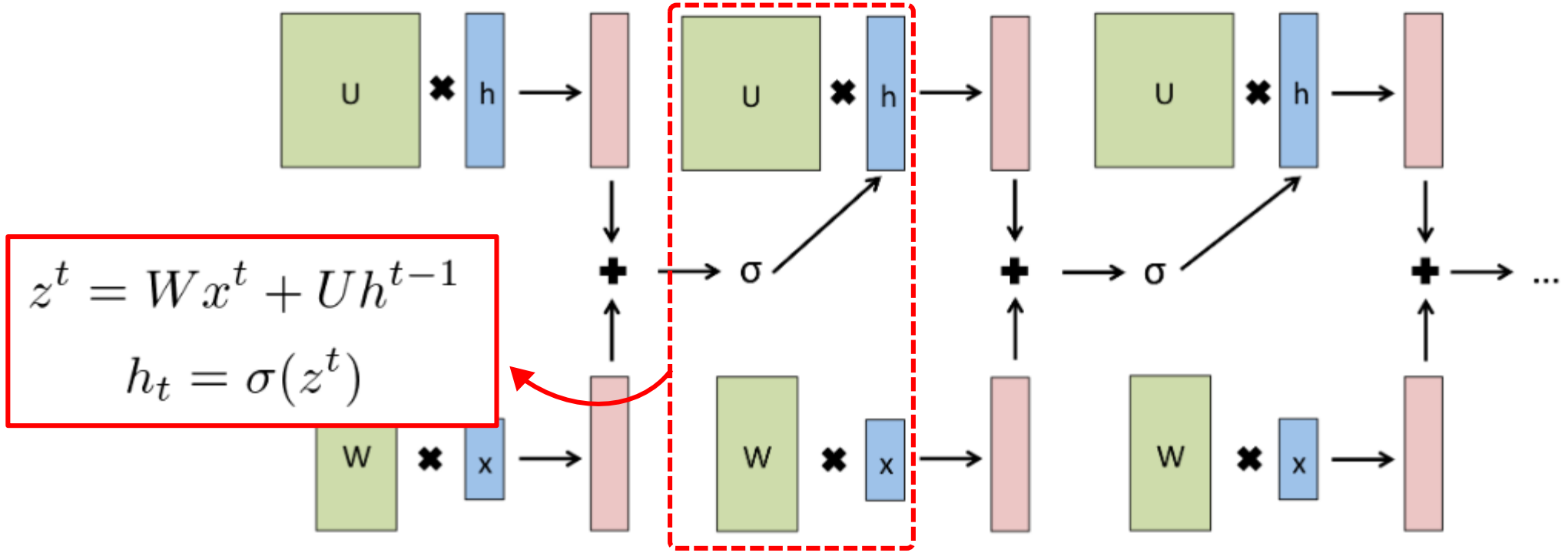


딥러닝 (Deep Learning)을 계산하는 방법

RNN을 계산하는 방법

Input과 Hidden state를 행렬의 형태로 변환

- GRU와 LSTM에서는 Input과 Hidden state에 대한 weight가 존재
- Input과 Hidden state에 각각 weight를 곱하는 과정을 행렬 곱셈



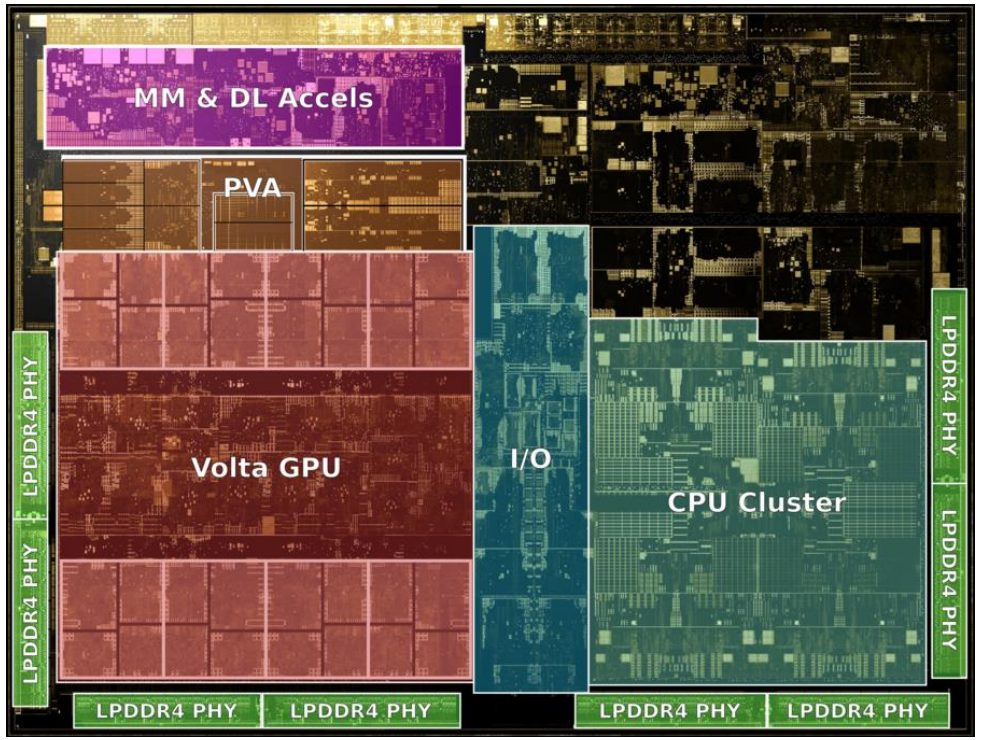
목차

- 딥러닝과 행렬 곱셈
- **딥러닝 가속을 위한 다양한 가속 시스템**
- GPU에서의 딥러닝 가속 방법: CUDA 코어/텐서 코어
- 정리

딥러닝 가속을 위한 다양한 시스템

- **가속기가 시스템에서 연결되는 방식에 따른 구분**
  - **서버 시스템:** PCI-Express에 연결하여 동작하는 방법 (FPGA*, GPU, TPU)
  - **임베디드 시스템**: **칩 내부**에서 CPU의 작업을 도와주는 보조 프로세서 (Co-processor)

서버 시스템



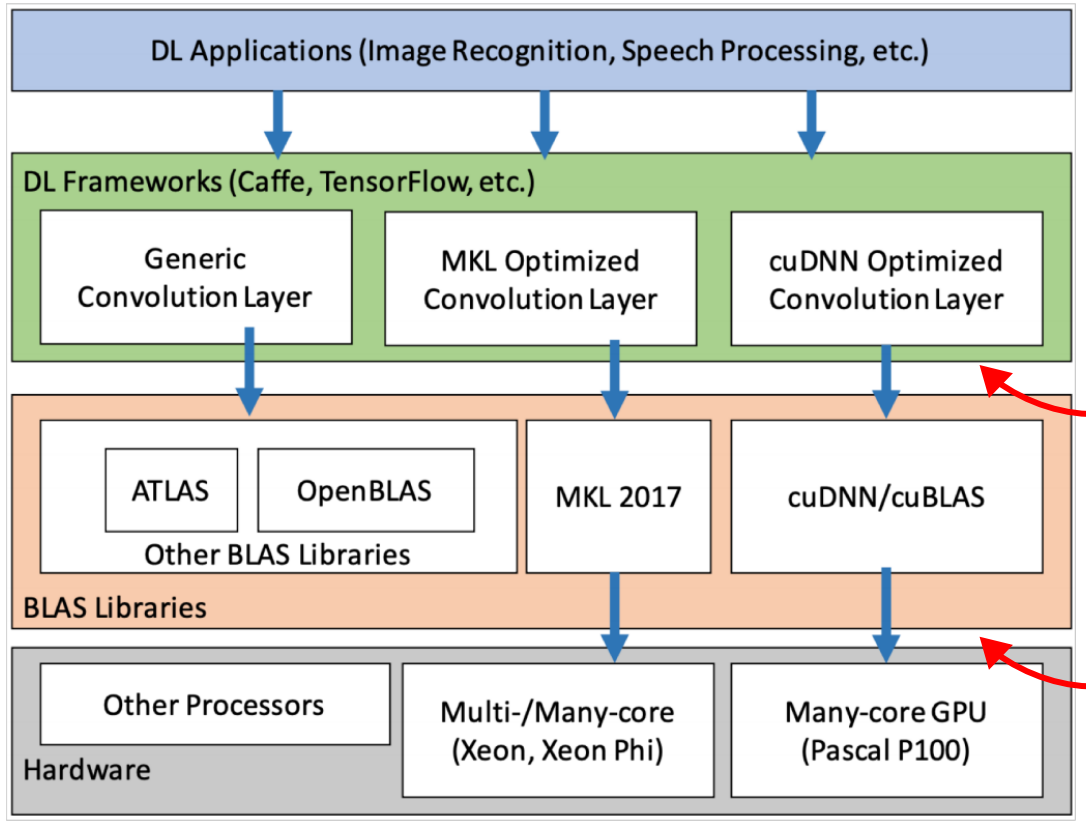
임베디드 시스템

*FPGA (Field Programmable Gate Array): 프로그래밍이 가능한 반도체 회로

딥러닝 가속을 위한 다양한 시스템: 서버 시스템

Graphic Processing Unit (GPU)

- 딥러닝의 학습과 추론을 위해 가장 많이 사용되고 있는 장치
- GPU 하드웨어를 사용하기 위한 BLAS (cuBLAS) 또는 신경망 라이브러리 (cuDNN)



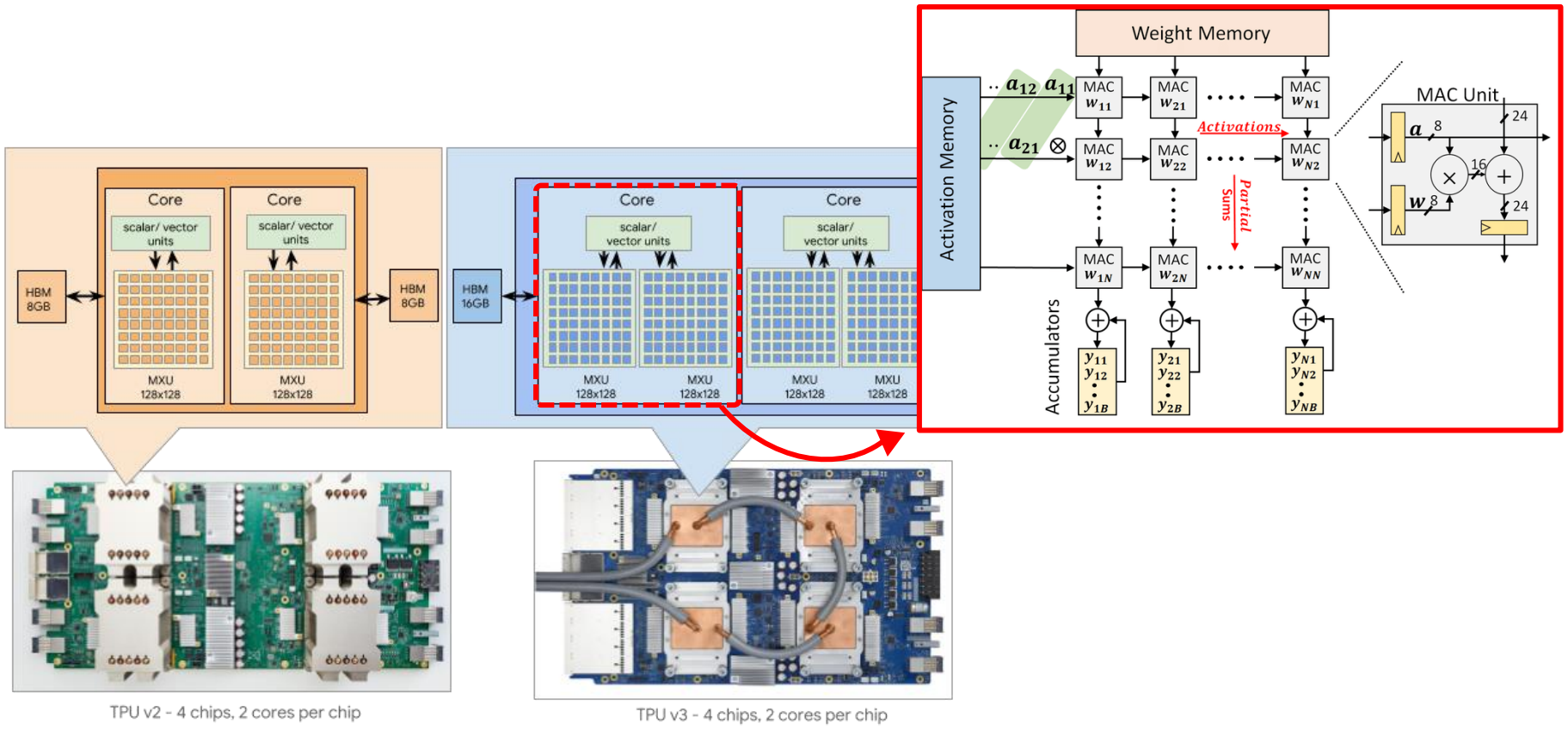

```python
# Variables Init
init = tf.initialize_all_variables()

# Session Run
sess = tf.Session()
sess.run(init)

# Training
for step in xrange(0, 201):
    sess.run(train)
    if step % 20 == 0:
        print step, sess.run(W), sess.run(b)
```

```
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    C[i] = A[i] + B[i];
}

int main()
{
...
    // Kernel invocation with N threads
    VecAdd<<<M, N>>>(A, B, C);
...
}
```

딥러닝 가속을 위한 다양한 시스템: 서버 시스템

Tensor Processing Unit (TPU)

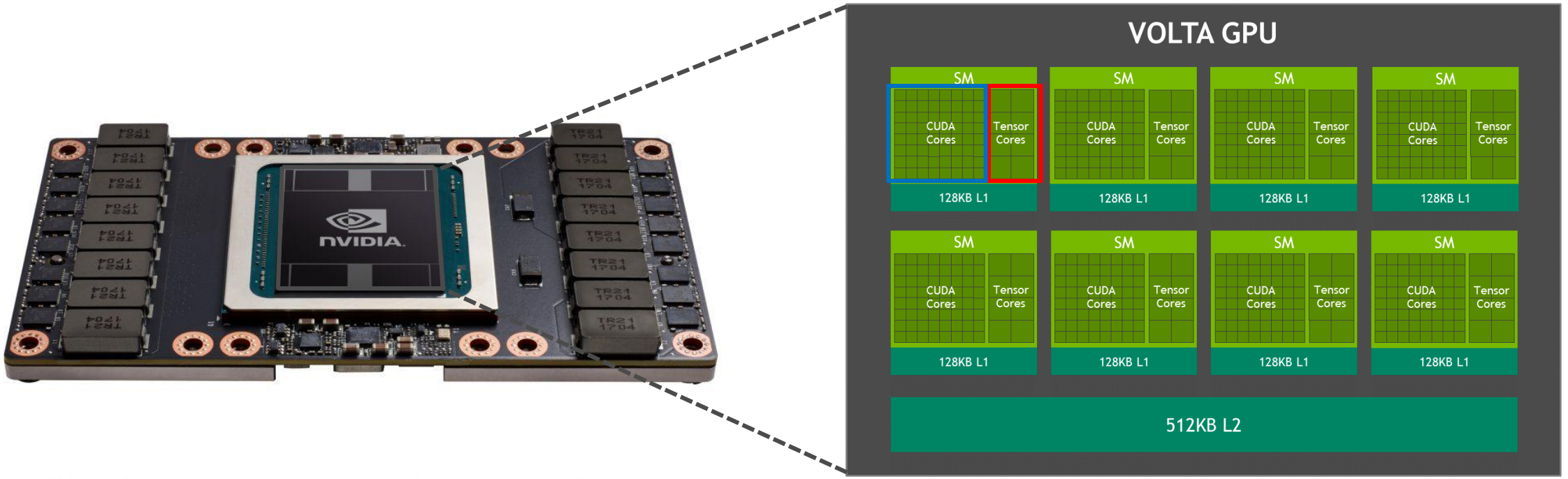
- 딥러닝 연산에서 자주 사용되는 **행렬/벡터의 곱셈**에 특화된 전용 장치
- 많은 개수의 **Multiply and Accumulate (MAC) 연산기**를 포함하는 구조



딥러닝 가속을 위한 다양한 시스템: 텐서 코어

GPU 속의 텐서 코어 (Tensor Core)

- **4x4 크기의 행렬 곱셈**을 위한 전용 하드웨어 (TPU)
  - 4x4 크기의 행렬 곱셈에는 64번의 곱셈과 56번의 덧셈이 필요
  - 텐서 코어에서는 이러한 4x4 행렬 곱셈을 **1 Cycle**에 연산 가능
- Volta 아키텍처 부터 사용 가능



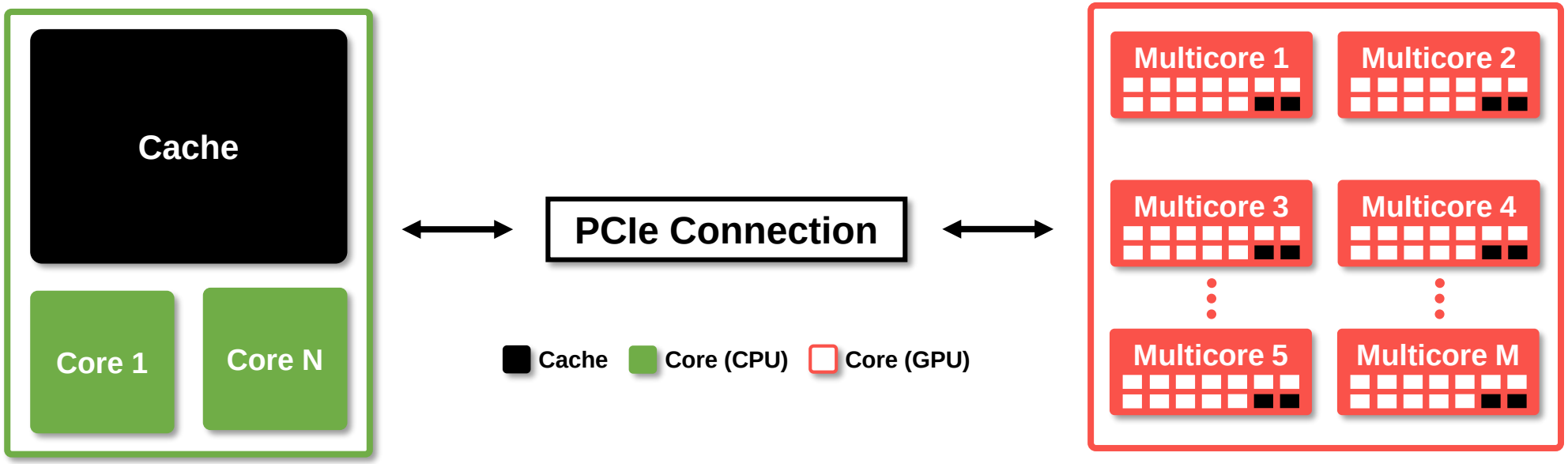
목차

- 딥러닝과 행렬 곱셈
- 딥러닝 가속을 위한 다양한 가속 시스템
- **GPU에서의 딥러닝 가속 방법: CUDA 코어/텐서 코어**
- 정리

GPU에서의 딥러닝 가속 방법: GPU 특징

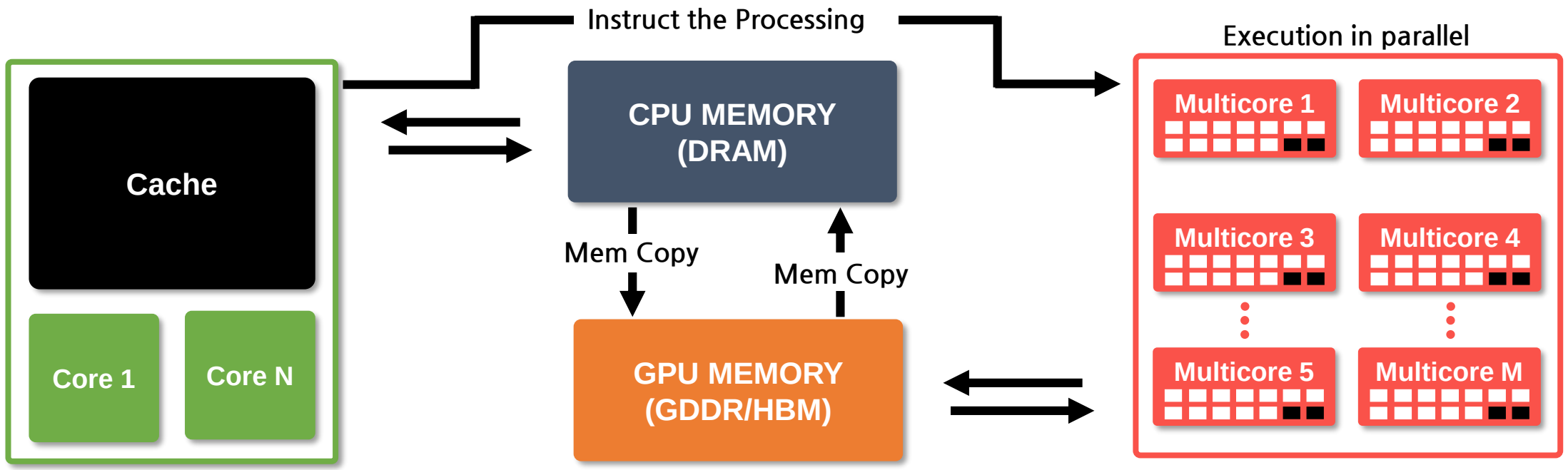
GPU와 CPU의 차이점

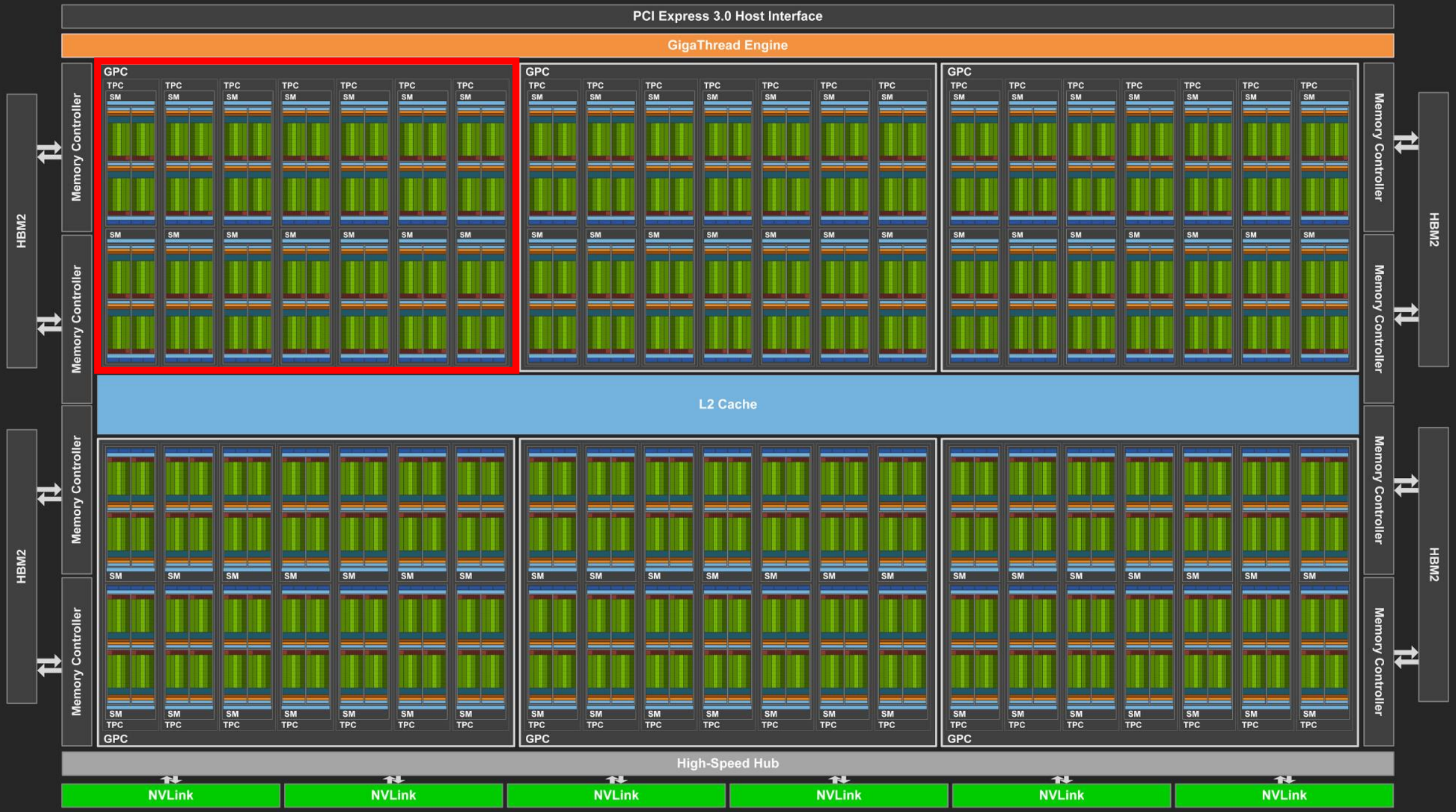
- CPU: 큰 크기의 캐시, 작은 크기의 연산 장치
- GPU: 작은 크기의 캐시, 큰 크기의 연산 장치
  - 병렬처리에 특화된 하드웨어 (많은 개수의 연산 장치)
  - 딥러닝과 같은 병렬성이 높은 작업에 적합
  - **여러 개의 연산 장치가 하나의 제어 장치를 공유하는 구조**



GPU에서의 딥러닝 가속 방법: GPU에서의 동작 과정

- GPU에서 딥러닝 모델을 연산하는 과정
  - 메모리 할당과 복사 (Memory allocation and copy)
    - CPU (Host) to GPU (Device) or GPU (Device) to CPU (Host)
  - 작업의 실행 (Instruct the processing)
    - CPU가 GPU에게 작업을 시작하도록 명령
  - 병렬처리 (Execution in parallel)

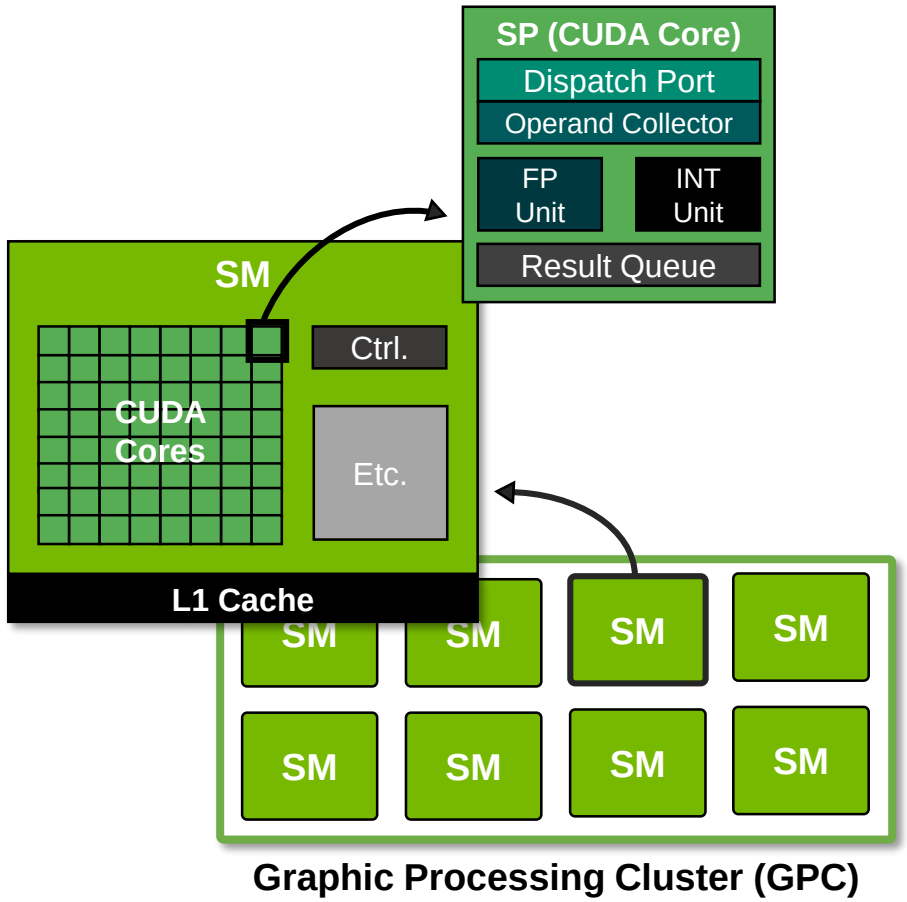



PCI Express 3.0 Host Interface
GigaThread Engine
GPC
TPC
SM
Memory Controller
HBM2
L2 Cache
High-Speed Hub
NVLink

GPU에서의 딥러닝 가속 방법: CUDA 코어

계층적인 하드웨어 구조

- **CUDA core (Streaming processor, SP)**
  - GPU 구조에서 가장 기본적인 연산 단위
  - 멀티코어 CPU에서의 하나의 코어에 해당
- **Streaming multiprocessor (SM)**
  - 여러 개의 CUDA Core와 L1 캐시를 포함
  - CUDA Core 여러 개는 하나의 컨트롤 로직을 공유
  - 서로 다른 SM 간에는 각 SM에 저장된 데이터에 접근 불가
- **Graphic processing cluster (GPC)**
  - SM의 그룹
- **GPC, SM, SP의 개수**
  - 아키텍처의 종류에 따라 다름 (Pascal, Volta, Turing)



GPU에서의 딥러닝 가속 방법: CUDA 코어

계층적인 프로그래밍 모델

- **Thread**
  - 멀티코어에서의 thread와 동일한 개념
  - 하나의 thread는 하나의 CUDA Core에서 동작
- **Warp**
  - 하드웨어에서 스레드를 처리하는 기본적인 단위
  - 32개의 스레드가 묶인 단위
- **Thread Block**
  - 프로그래밍 모델에서 thread를 처리하는 단위
  - 여러 개의 thread가 묶여 thread block
  - Thread block은 Warp 단위로 나눠져, SM에서 동작
- **Grid**
  - Thread block이 여러 개 묶인 단위

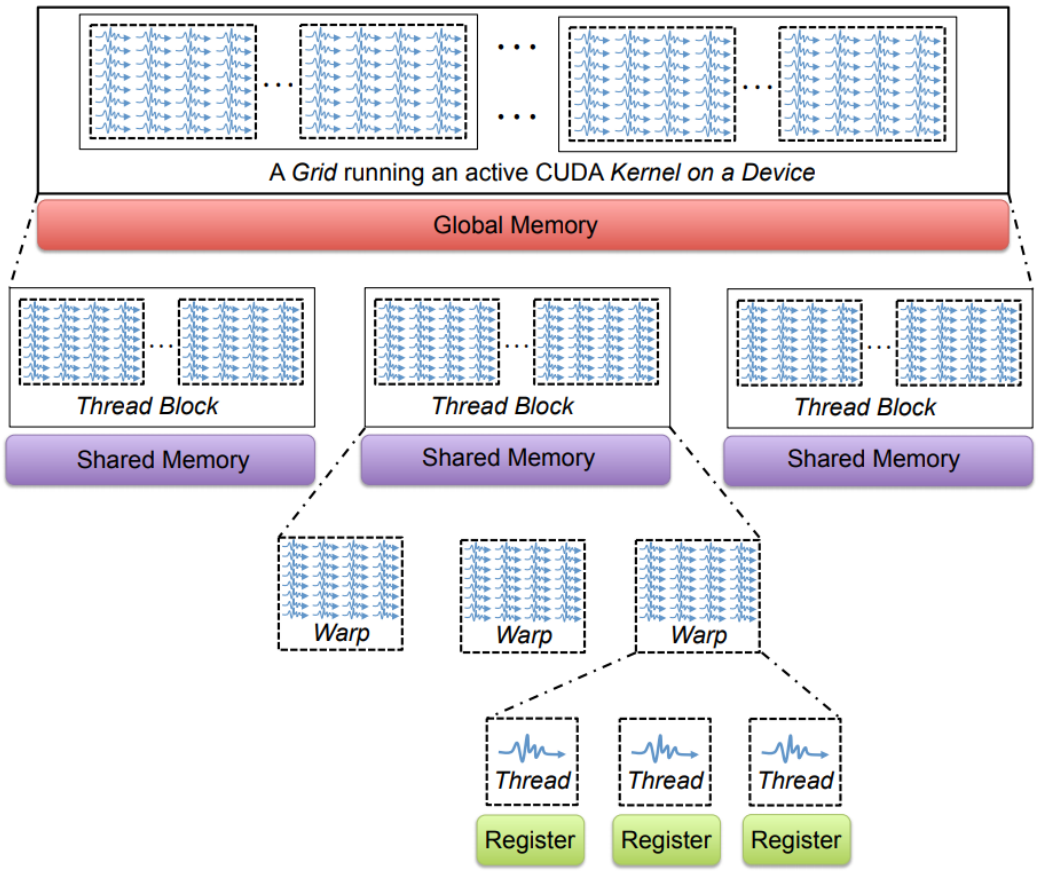


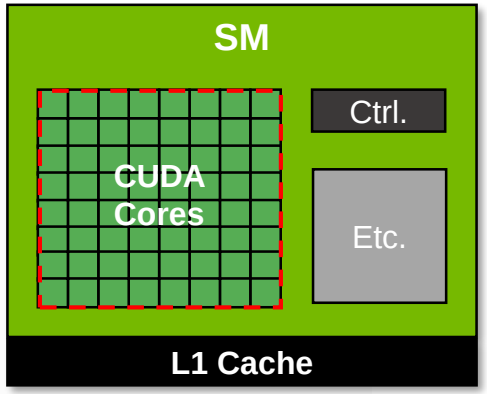
Figure 2: The memory and thread hierarchies in the CUDA programming model.

GPU에서의 딥러닝 가속 방법: CUDA 코어

커널 (Kernel)

GPU에서 동작하는 C/C++ 함수

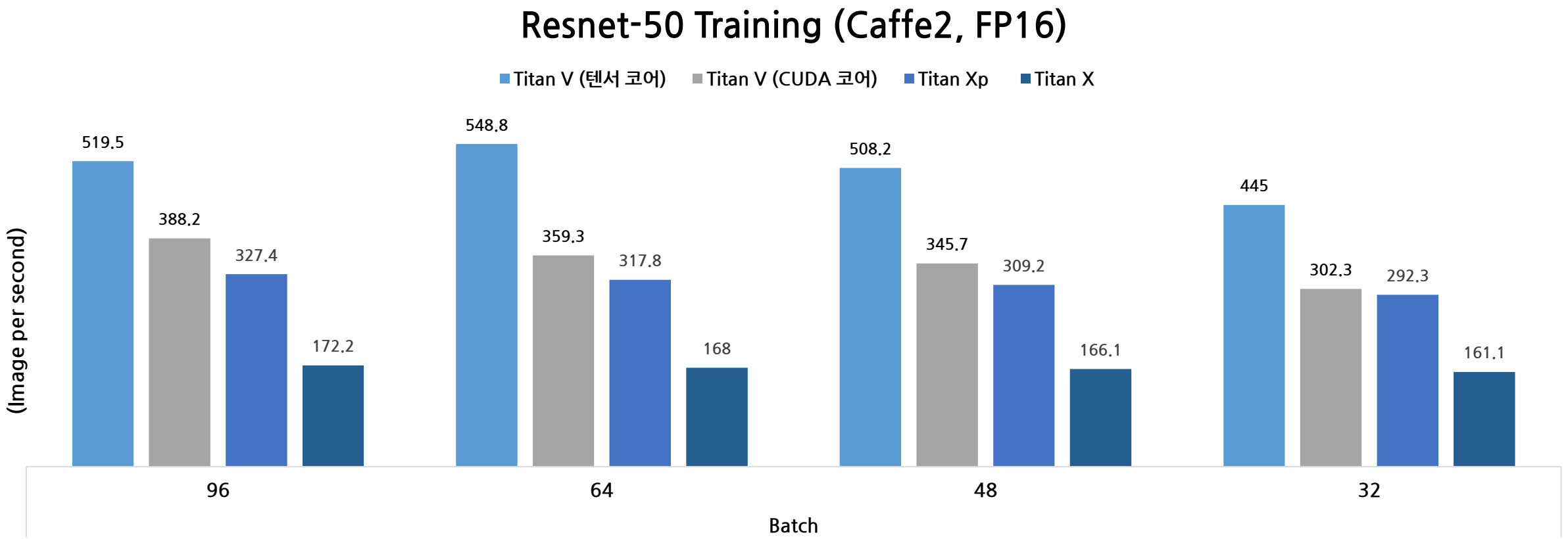
- Control-level parallelism: 한 종류의 함수를 1개씩 thread가 담당
  - 예) 덧셈만 하는 thread, 곱셈만 하는 thread
- **Data-level parallelism**: 데이터를 쪼개서 1개의 thread가 같은 함수를 수행
  - 예) 100개의 데이터를 **100개의 thread가 1개씩 맡아서 덧셈과 곱셈을 수행**




```
2   # Commnon vector addition
3   for(i = 0; i < N; i++){
4       c[i] = a[i] + b[i];
5   }
6
7   # GPU vector addtion
8   __global__ void vecAdd(double *a, double *b, double *c, int n){
9       int id = blockIdx.x*blockDim.x+threadIdx.x;
10      c[id] = a[id] + b[id];
11  }
```

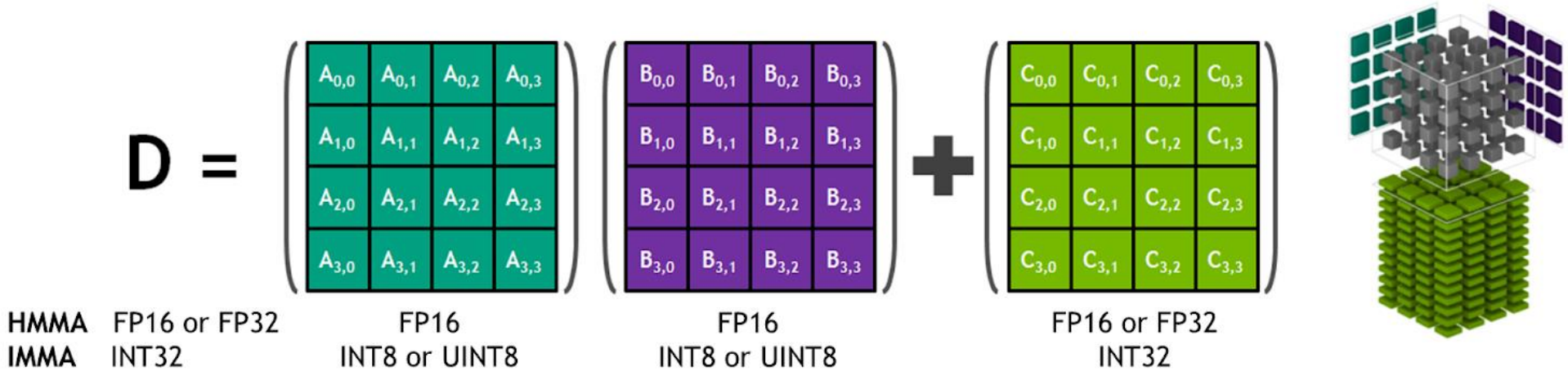
GPU에서의 딥러닝 가속 방법: 텐서 코어

- **행렬 곱셈에 특화된 전용 하드웨어**
  - **텐서 코어**를 사용하여 **4~8배** 실행시간 개선 가능 [CUDA 9]
  - 실제는 **약 2배 정도** 실행시간 개선 가능



GPU에서의 딥러닝 가속 방법: 텐서 코어

- **4x4 크기의 행렬 곱셈에 특화된 전용 장치**
  - 4x4 크기의 행렬 곱셈에는 64번의 곱셈, 56번의 덧셈 필요
  - **텐서 코어 연산 성능 (GFLOPS): (64+56) * 1.7GHz = 204GFLOPS***
  - Volta (FP16), Turing (rINT4, INT8, FP16)



*FLOPS (**FL**oating point **O**perations **P**er **S**econd): 컴퓨터가 초당 몇번의 부동소수점 연산이 가능한지를 표현하는 지표

GPU에서의 딥러닝 가속 방법: 텐서 코어

- Mixed-precision computation
  - Input data에 FP32가 아닌 FP16으로도 충분한 학습 성능을 얻는 것이 가능
  - 곱셈 연산에서는 FP16 (Activation/Weight), 덧셈 연산에서는 FP32를 사용
    - **Key factor of accuracy**: bit-width of accumulation [Wang, NIPS18]

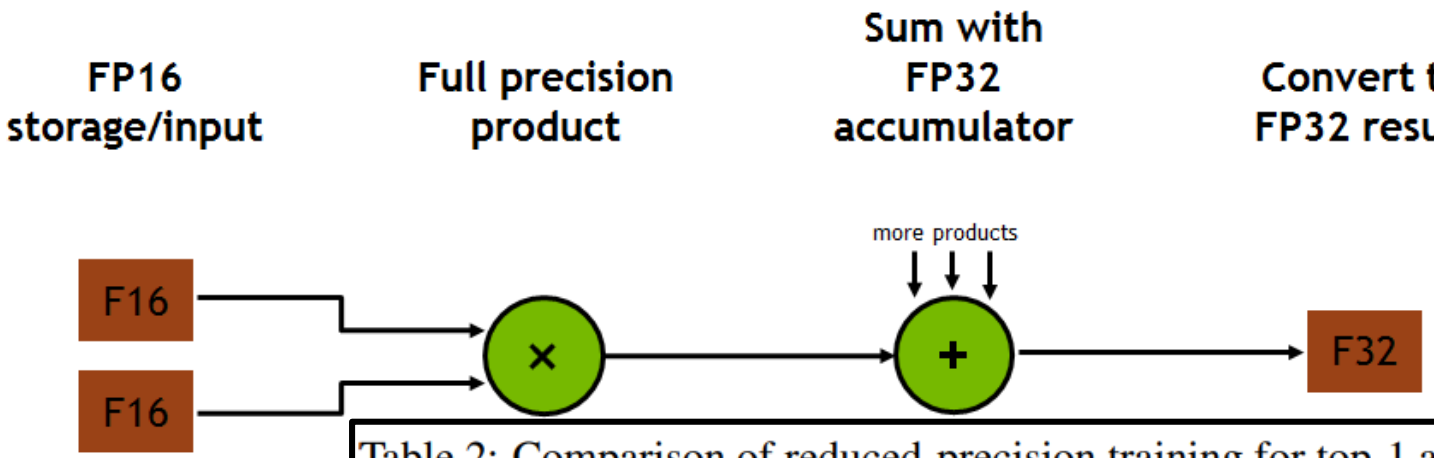


Table 2: Comparison of reduced-precision training for top-1 accuracy (%) for AlexNet (ImageNet)

Reduced Precision Training Scheme	Bit-Precision W	x	dW	dx	acc	$FP32$	Reduced Precision
DoReFa-Net [22]	1	2	32	6	32	55.9	46.1
WAGE [19]	2	8	8	8	32	N/A	51.6
DFP [4]	16	16	16	16	32	57.4	56.9
MPT [16]	16	16	16	16	32	56.8	56.9
Proposed *FP8* training	**8**	**8**	**8**	**8**	**16**	**58.0**	**57.5**

GPU에서의 딥러닝 가속 방법: 텐서 코어

GPU 내부의 텐서 코어

- **Streaming multiprocessor (SM)는 8개의 텐서 코어를 포함**
  - SM 내부의 Sub-Core당 2개의 텐서 코어
- **Titan V는 80개의 SM을 포함**
  - 640 (80*8) Tensor Cores는 FP16 계산에서 **125TFLOPS** 가능
  - CUDA 코어는 15.7TFLOPS

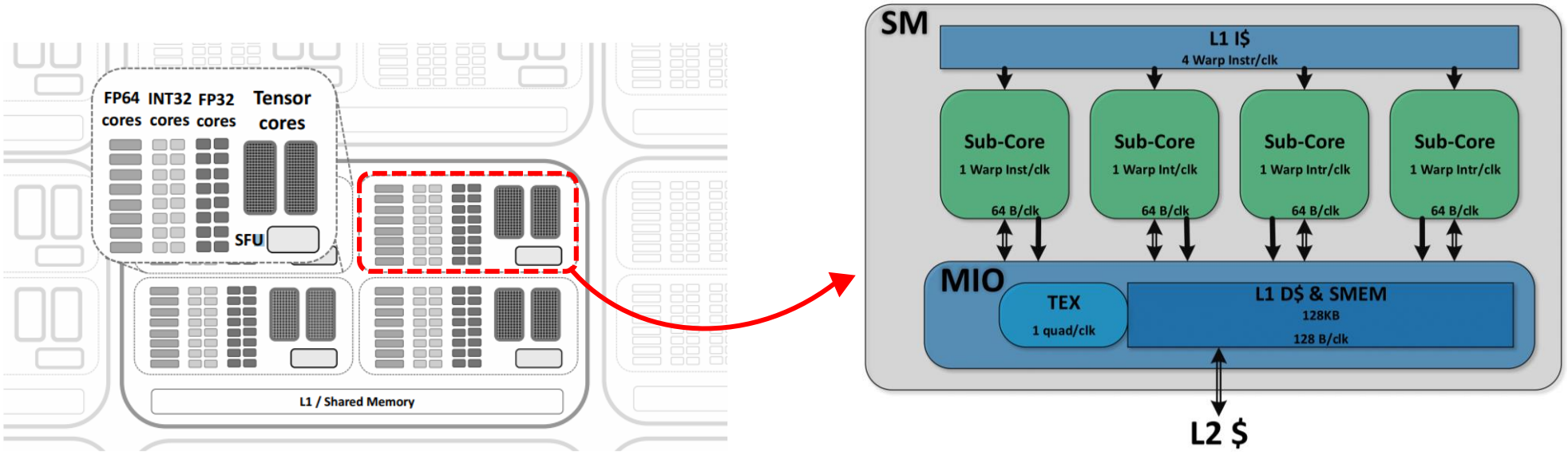
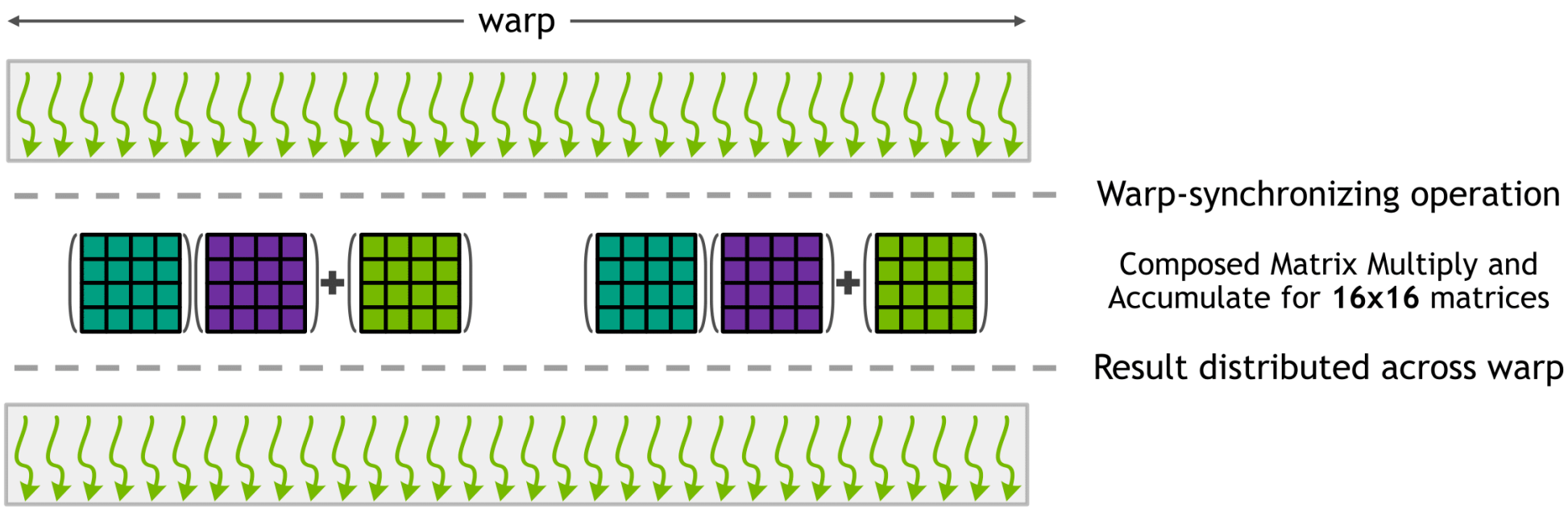


Fig. 2: Simplified diagram of the Volta SM architecture. The NVIDIA Tesla V100 uses 80 SMs.

GPU에서의 딥러닝 가속 방법: 텐서 코어

- **텐서 코어를 사용하는 방법 (어려운 방법)**
  - **Warp Matrix Multiply and Accumulate (WMMA) API**
    - CUDA에서 사용가능한 방법 (Warp 단위로 행렬의 로드, 스토어, 계산)
    - 4x4 크기의 행렬이 가능한 텐서 코어를 사용, **16x16 크기의 행렬 계산**



GPU에서의 딥러닝 가속 방법: 텐서 코어

- Warp Matrix Multiply and Accumulate (WMMA) API
  - wmma fragment: GPU 내부 레지스터에 계산에 사용할 행렬의 값 로드
  - fill_fragment: 특정 값으로 초기화
  - load_matrix_sync/store_matrix_sync
  - mma_sync: 행렬 곱셈 수행

Listing 1: CUDA 9 WMMA provides a direct way to calculate 16x16 matrix matrix-multiply-and-accumulate using a CUDA Warp (32 threads).

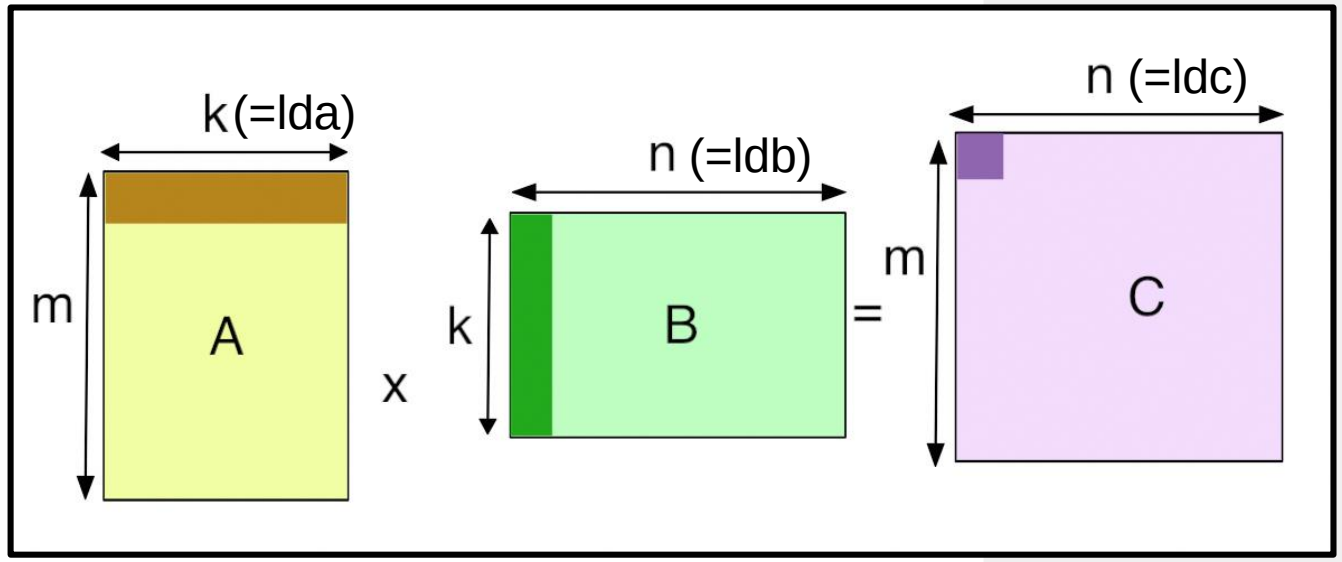
```
// Calculate AB with NVIDIA Tensor Cores
// Kernel executed by 1 Warp (32 Threads)
__global__ void tensorOp(float *D, half *A, half *B) {
        // 1. Declare the fragments
        wmma::fragment<wmma::matrix_a, M, N, K, half, wmma::col_major> Amat;
        wmma::fragment<wmma::matrix_b, M, N, K, half, wmma::col_major> Bmat;
        wmma::fragment<wmma::accumulator, M, N, K, float, void> Cmat;
        // 2. Initialize the output to zero
        wmma::fill_fragment(Cmat, 0.0f);
        // 3. Load the inputs into the fragments
        wmma::load_matrix_sync(Amat, A, M);
        wmma::load_matrix_sync(Bmat, B, K);
        // 4. Perform the matrix multiplication
        wmma::mma_sync(Cmat, Amat, Bmat, Cmat);
        // 5. Store the result from fragment to global
        wmma::store_matrix_sync(D,Cmat, M, wmma::mem_col_major);
}
```

GPU에서의 딥러닝 가속 방법: 텐서 코어

텐서 코어를 사용하는 방법 (쉬운 방법)

CUDA Basic Linear Algebra Subprogram (cuBLAS)

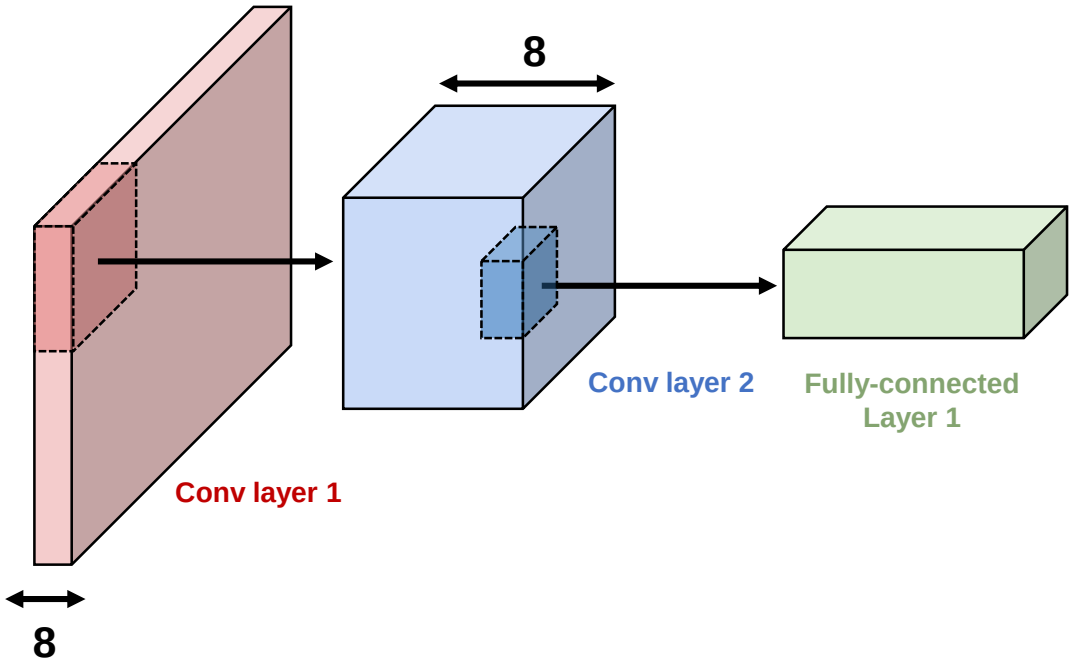
- **cublasGemmEx API**
  - 행렬을 연산 알고리즘 (GemmAlgo_t: CUBLAS_GEMM_DEFAULT_TENSOR_OP)
  - **다음의 파라미터는 반드시 지켜져야 함**
    - 8의 배수 (k, lda, ldb, ldc), 4의 배수 (m)




```
cublasStatus_t cublasGemmEx(cublasHandle_t handle,
                            cublasOperation_t transa,
                            cublasOperation_t transb,
                            int m,
                            int n,
                            int k,
                            const void    *alpha,
                            const void     *A,
                            cudaDataType_t Atype,
                            int lda,
                            const void     *B,
                            cudaDataType_t Btype,
                            int ldb,
                            const void    *beta,
                            void           *C,
                            cudaDataType_t Ctype,
                            int ldc,
                            cudaDataType_t computeType,
                            cublasGemmAlgo_t algo)
```

GPU에서의 딥러닝 가속 방법: 텐서 코어

- 텐서 코어를 사용하는 방법 (쉬운 방법)
  - CUDA Deep Neural Network library (cuDNN)
    - cudnnConvolutionForward API
      - Math type: CUBLAS_GEMM_DEFAULT_TENSOR_OP
      - Input and Output channel (8의 배수)



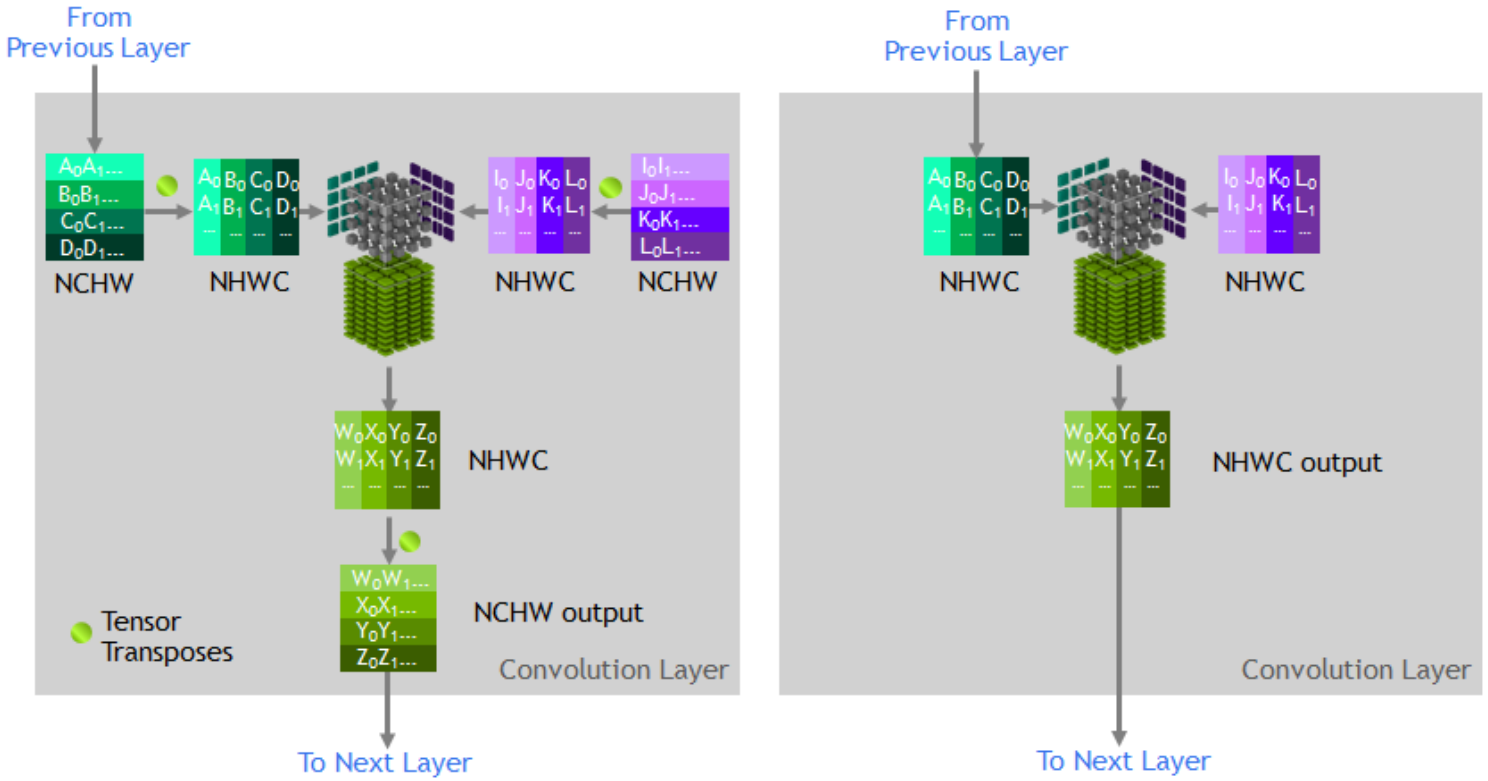

```
// Set the compute data type (below as CUDNN_DATA_FLOAT):
checkCudnnErr( cudnnSetConvolutionNdDescriptor(cudnnConvDesc,
                                               convDim,
                                               padA,
                                               convstrideA,
                                               dilationA,
                                               CUDNN_CONVOLUTION,
                                               CUDNN_DATA_FLOAT) );

// Set the math type to allow cuDNN to use Tensor Cores:
checkCudnnErr( cudnnSetConvolutionMathType(cudnnConvDesc, CUDNN_TENSOR_OP_MATH) );

// Choose a supported algorithm:
cudnnConvolutionFwdAlgo_t algo = CUDNN_CONVOLUTION_FWD_ALGO_IMPLICIT_PRECOMP_GEMM;

// Allocate your workspace:
checkCudnnErr( cudnnGetConvolutionForwardWorkspaceSize(handle_, cudnnIdesc,
                                                       cudnnFdesc, cudnnConvDesc,
                                                       cudnnOdesc, algo, &workSpaceSize) );

if (workSpaceSize > 0) {
    cudaMalloc(&workSpace, workSpaceSize);
}

// Invoke the convolution:
checkCudnnErr( cudnnConvolutionForward(handle_, (void*)(&alpha), cudnnIdesc, devPtrI,
                                       cudnnFdesc, devPtrF, cudnnConvDesc, algo,
                                       workSpace, workSpaceSize, (void*)(&beta),
                                       cudnnOdesc, devPtrO) );
```

목차

- 딥러닝과 행렬 곱셈
- 딥러닝 가속을 위한 다양한 가속 시스템
- GPU에서의 딥러닝 가속 방법: CUDA 코어/텐서 코어
- **정리**

정리

- **강력한 행렬 연산 장치: 텐서 코어**
  - CUDA 코어보다 더 빠르게 행렬로 이뤄진 레이어 계산 가능
  - **하지만, 능사는 아니다 !**
    - **Pruning 사용 불가능, 파라미터의 크기가 지켜지지 않는 다면 동작하지 않음**
    - **텐서의 저장 포맷에 따라 추가적인 변환 과정이 필요 (N: batch, H: height, W: width, C: # of channel of image)**



감사합니다

Reference

- **[CUDA 9]**
  - https://devblogs.nvidia.com/programming-tensor-cores-cuda-9/
- **[Wang, NIPS18]**
  - Training Deep Neural Networks with 8-bit Floating Point Numbers