NVIDIA Tensor Core Programmability, Performance & Precision

1. Introduction

- 고차원 행렬 연산을 효율적으로 하기 위해서 Deep Learning application용 하드웨어 개발을 많이 한다.
  Ex) TPU, Volta(Tensor core)
- Tensor Core는 4x4 행렬 연산(A * B + C) 을 효율적으로 함
  half precision 곱셈, single precision 덧셈

1. Introduction

텐서 코어를 어떻게 잘 활용할 수 있을까?

- Suitable programming models for tensor cores
  성능과 표현력이 둘 다 좋은 프로그래밍 인터페이스
- Tensor core로 얻을 수 있는 성능 증가를 측정
  in various problem size and workloads
- Mixed precision을 쓸 때 일어나는 Loss of precision 측정 및 정확도를 높이는 테크닉 연구

2. Related Work

- MS: Catapult system(uses FPGAs)
- Movidius: Myriad 2 Vision Processing Unit
- Google: TPU
  (main engine is MAC matrix multiply unit 256x256 MACs)
- Intel: Neural Network Processor(NPP)
  새로운 메모리 아키텍쳐(캐쉬가 없고 데이터 흐름을 소프트웨어에서 제어)

3. Volta Architecture

- L1 cache + shared memory
- Mixed precision tensor core

- 6 GPU Processing Clusters
  - 7 Texture Processing Clusters
  - 14 Streaming Multiprocessors
- SFU: sin, cosine, reciprocal square root
- Half precision, half bandwidth, faster, high throughput

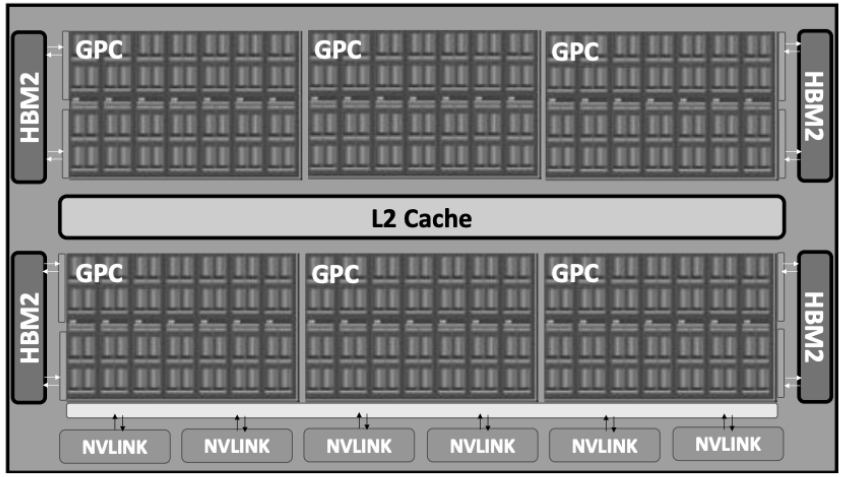

Fig. 1: Volta GV100 GPU architecture features six GPCs and 16 GB HBM2. Adapted from [18].

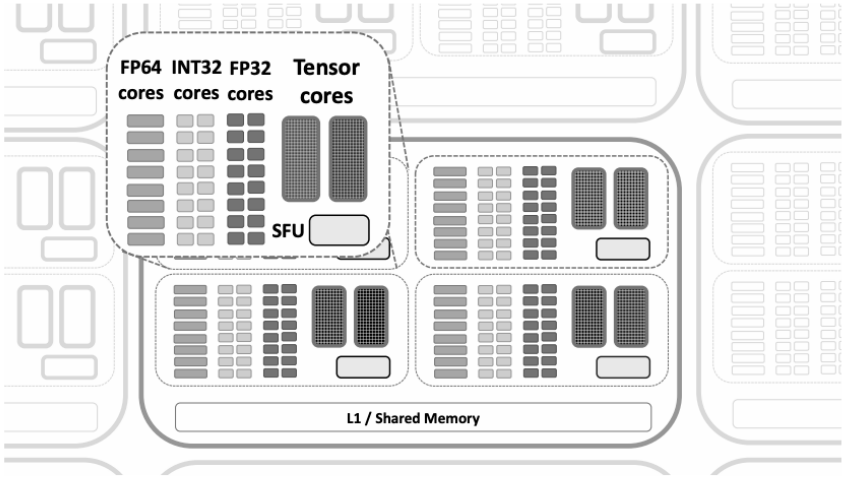

Fig. 2: Simplified diagram of the Volta SM architecture. The NVIDIA Tesla V100 uses 80 SMs.

4. Programming NVIDIA Tensor Core

- 4x4 행렬의 행렬곱, 합 연산만 가능
- BLAS GEMM 연산 표현 가능
  General Matrix to Matrix Multiplication

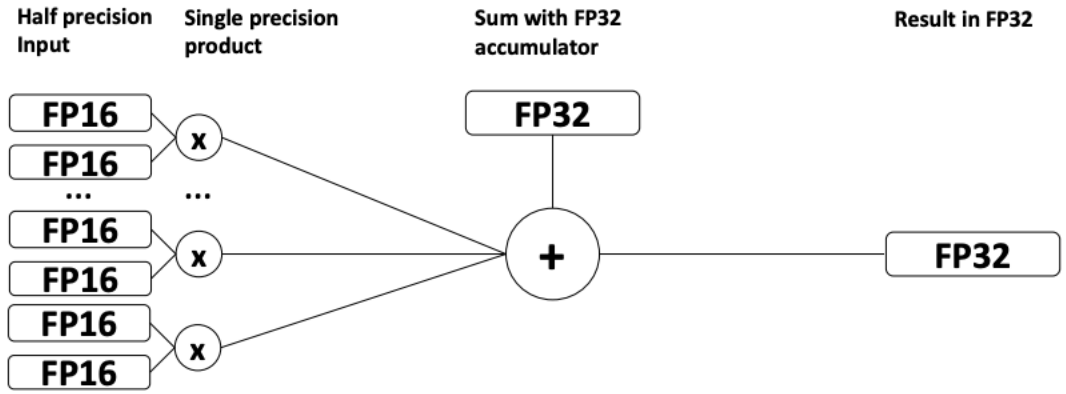

Fig. 3: FMAs in NVIDIA Tensor Cores.

4. Programming NVIDIA Tensor Core

- The lowest level interface
  CUDA 9 Warp Matrix Multiply and Accumulation(WMMA) api
  현재 직접 Tensor Core를 프로그램 할 수 있는 유일한 수단
- 16x16 행렬 곱, 합 연산 지원
- 최근 CUDA 9 에서는 정사각 행렬이 아니어도 가능
- 큰 행렬곱만을 지원하는 이유는 하드웨어보다 많은 스레드를 실행시키는 CUDA 철학과 관련있음

4. Programming NVIDIA Tensor Core

```
// 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);
}
```

4. Programming NVIDIA Tensor Core

다양한 사이즈의 행렬 곱에 사용할 수 있는 행렬 연산 방법(api)

- Tiled Matrix Multiply with CUDA 9 WMMA
  큰 행렬을 나누어 계산해서 합친다
- NVIDIA CUTLASS
  CUDA C++ templated header-only library GEMM 연산용
  다양한 precision 지원
- NVIDIA cuBLAS
  선형대수 연산 라이브러리, Tensor core용 GEMM routine 제공

4. Programming NVIDIA Tensor Core

- 많은 어플리케이션이 작은 사이즈의 행렬 연산을 Parallel하게 함
- cuBLAS를 쓰는게 가장 편하다
  batched sgemm은 지원하지만 gemm은 지원하지 않는다

5. Precision Loss

- Mixed Precision이 반올림하면서 큰 에러발생 가능성
- 행렬에 곱해지는 값은 이전 iteration에 비해 값이 매우 작고 계산 결과도 작다 그래서 half precision
- 하지만 덧셈을 하면 훨씬 큰 값이 된다 그래서 precision loss가 클 수 있어서 single precision
- 보통 뉴럴넷이 어느정도 까지는 precision loss에 큰 영향을 받지는 않는다
- Exception of mentecarlo code 같은 경우는 반올림 에러에 민감하다

5. Precision Loss

- Limited range: 5 bits of exponent
  maximum 65,504, floating-point +-65,504
- Decreasing precision with increasing value range intervals
  큰 수에서 정확도 손실이 매우 큼
  어떤 제곱수 사이에서 1024개의 값만 표현할 수 있음
  2^0 2^1 사이에 1024개
  2^10 2^11 사이에 1024개

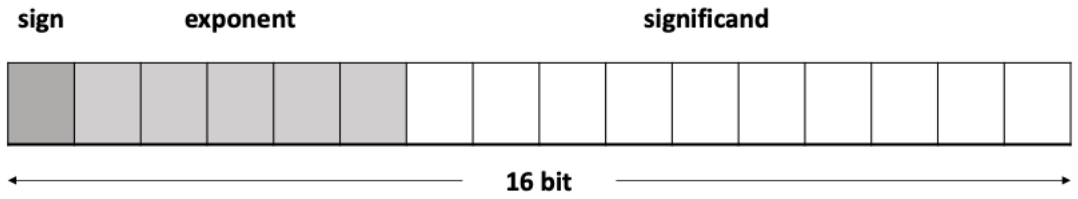

Fig. 4: Half precision floating-point number representation.

5. Precision Loss

Precision loss 줄이기

- 메모리를 더 사용
- 한 번의 추가적인 행렬 연산
- Precision refinement(iterative precision refinement)

$$R_A = A_{single} - A_{half}.$$

$$A_{single}B_{half} = (A_{single} - A_{half} + A_{half})B_{half} = (R_A + A_{half})B_{half} = R_AB_{half} + A_{half}B_{half}.$$

$$A_{single}B_{single} = (R_A + A_{half})(R_B + B_{half}) = R_AR_B + A_{half}R_B + R_AB_{half} + A_{half}B_{half}.$$

5. Precision Loss

- CUDA core vs Tensor core

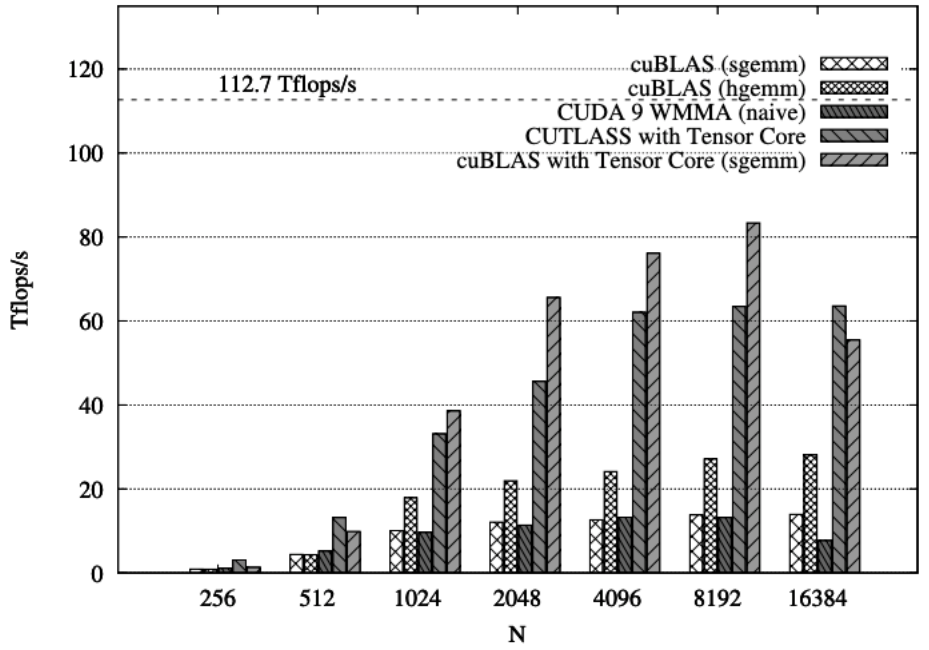

Fig. 6: GEMM performance without Tensor Cores in single and half precision (white bars) and with Tensor Cores using naive implementation with CUDA 9 WMMA, CUTLASS and cuBLAS (grey bars) varying with matrix size N.

Performance Experiment

- 행렬이 8192x8192일 때 Tensor Core 최고 성능
- Mixed precision일 때는 이론 성능 74%정도
  single and half precisio보다 6배 3배 성능
- CUTLASS가 cuBLAS보다 성능이 더 좋음
  다양한 tiling 설정 테스트 가능
- WMMA 구현은 sgemm에 비해 성능향상 없음
  shared memory를 사용 했을 때 5배 높은 성능 향상

Performance Experiment

Batched Gemm

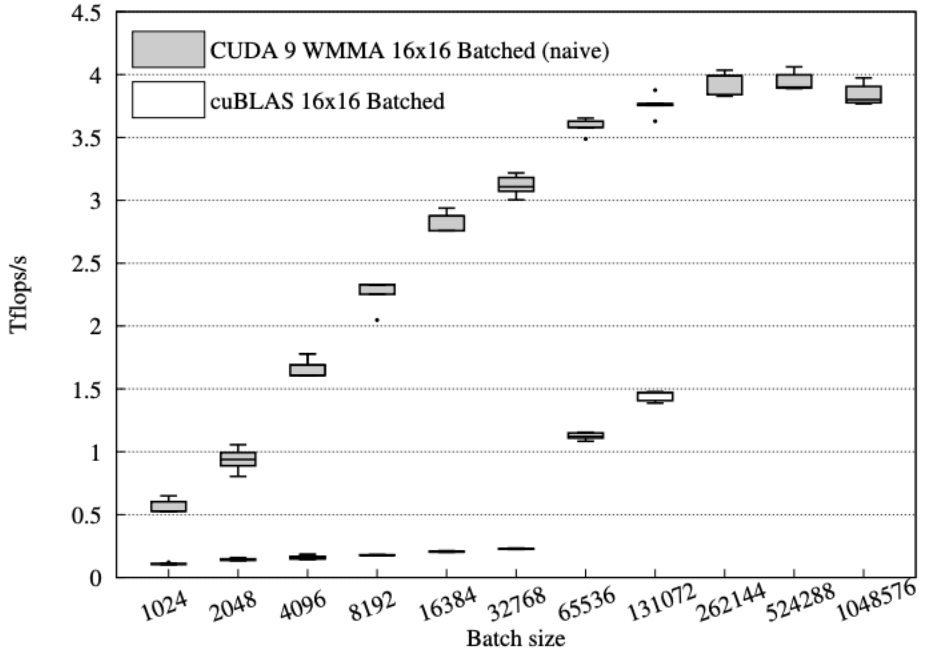

Fig. 7: Performance of cuBLAS batched `sgemm` on CUDA cores, and CUDA 9 WMMA implementation performing *batch size* 16×16 matrix multiplies. The cuBLAS batched `sgemm` cannot run for more than 131,072 multiplications as they require more memory than the available one on the Tesla V100 GPU.

Precision loss Experiment

quantify the decrease of precision loss

- error matrix e as e = (Chalf − Csingle)

$$\|e\|_{Max} = \max(|e_{i,j}|).$$

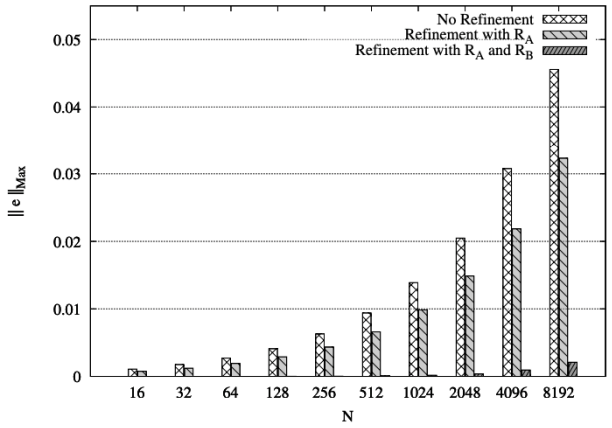

Fig. 8: Error in half precision (no refinement, white bars), using precision refinement with R_A, and precision refinement with both R_A and R_B varying the matrix size N.

Precision loss Experiment

- 에러가 행렬의 크기가 커질수록 커짐
  곱, 덧셈연산 횟수가 행렬 크기(NxN) N^2 배기 때문에

Precision loss Experiment

Computational Cost

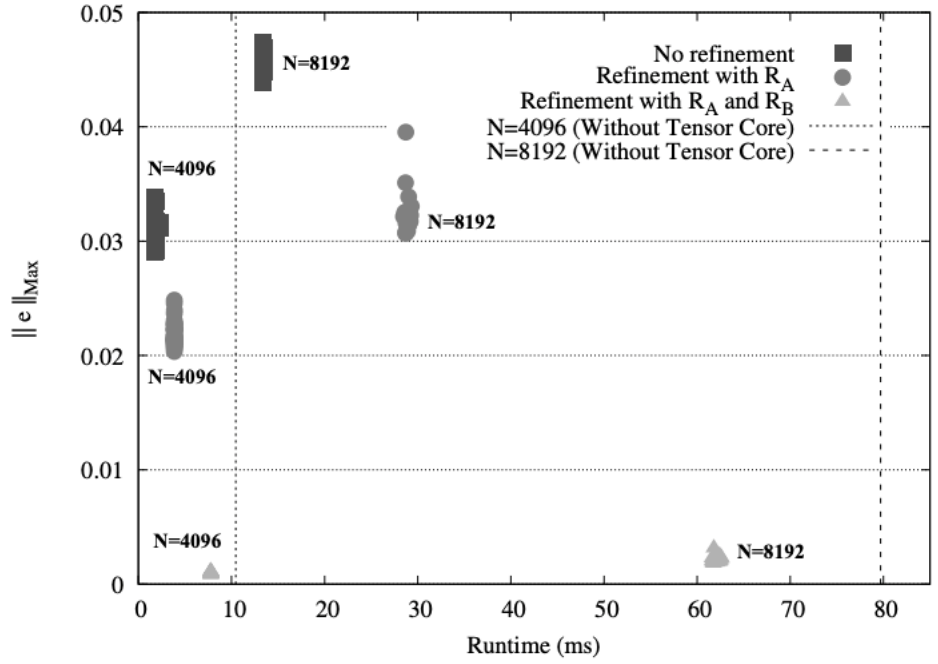

Fig. 9: Scatter plot with $\|e\|_{Max}$ on x axis and runtime on the y axis for `GEMM` with no refinement (squares), refinement with only R_A (circles), and with both R_A and R_B (triangles) for $N = 8,192$ and $N = 4,096$. The two dashed lines represent the execution time for `sgemm` without Tensor Cores.

Precision loss Experiment

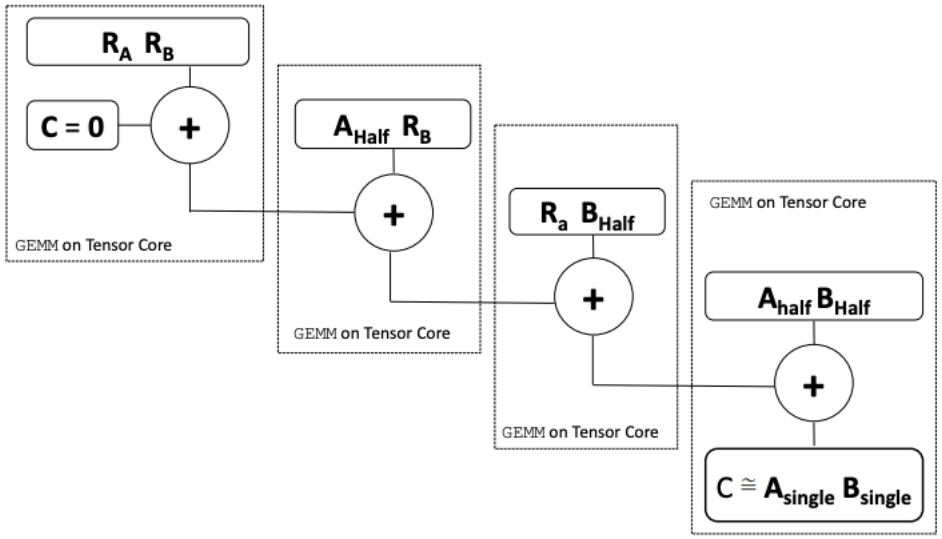

Fig. 5: Implementation of precision refinement using four pipelined GEMM on Tensor Cores.

Conclusion

- Tensor Core는 큰 행렬연산을 할 때 6배
  작은 행렬연산을 병렬로 할 때 2.5 ~ 12배 빠름
- Programmability
  WMMA api, CUTLASS, cuBLAS
- Performance
  Tensor Core > CUDA core (batched matrix, large matrix)
  memory traffic을 최소화 하는 데이터 배치 방법도 고려해야 함
- Precision
  Precision loss는 matrix가 커질 수록 크다