**<별첨8>**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 4차산업혁명을 선도하는 CODE형 SW 인재 양성  Capstone Design(종합설계) 결과보고서 | | | | | | | | | |
| **교과목명** | | | **병렬프로그래밍** | | | **학부(과)명** | | **소프트웨어융합대학** | | | |
| **작 품 명** | | | Tensor Core 기반의 행렬곱 가속화 및 딥러닝 적용 방식 분석 | | | | | | | | |
| **팀 명** | | | Tencore | | | | | | | | |
| **지도교수** | | | 이정근 | | | | | | | | |
| **참여**  **학생** | | **대표** | 소속학과 | | 학번 | | 성명 | | 연락처 | | |
| **참여**  **학생** | | **대표** | 빅데이터 | | 20143148 | | 이호중 | | 010-6375-5471 | | |
| **참여**  **학생** | | **팀원** | 소속학과 | | 성명 | | 소속학과 | | 성명 | | |
| **참여**  **학생** | | **팀원** | 콘텐츠IT | | 한승탁 | | 컴퓨터공학과 | | 김용호 | | |
| **참여**  **학생** | | **팀원** | 콘텐츠IT | | 조준형 | |  | |  | | |
| **참여**  **학생** | | **팀원** |  | |  | |  | |  | | |
| **참여**  **학생** | | **팀원** |  | |  | |  | |  | | |
| **참여 분야** | | |  SW융합대학 □ SW융합/연계 전공 | | | | | | | | |
| **참여기업** | | | GeoMex Soft(지오맥스소프트) | | | | | | | | |
| **지식재산권** | | | **유형** | □ 특허 □ 실용신안 □ 상표 □ 디자인 □ 기타( ) | | | | | | | |
| **지식재산권** | | | **명칭(출원명)** | | | | | | | **번호** | |
| **지식재산권** | | |  | | | | | | |  | |
| **과제 목적** | | | | | | | | | | | |
| 최근 CCTV 시장에서 지능형 또는 인지형 CCTV에 대한 수요가 증가하고 있습니다. 우리 회사는 지리정보 시스템 및 관제전문 회사로 지능형 CCTV 개발에 많은 관심을 가지고 있고 개발에 임하고 있습니다.  이번 캡스톤 과제에서 학생들이 지능형 시스템에 활용되는 딥러닝 적용시 GPU 구조의 핵심 기술인 Tensor Core를 이용한 가속화 부분에 대해서 연구하여 기업이 이를 활용할 수 있도록 한다. | | | | | | | | | | | |
| **과제 내용** | | | | | | | | | | | |
| 1. 요약  NVIDIA에서 Volta micro 아키텍처부터 특별한 유닛인 Tensor Core을 소개하였습니다. Tensor Core은 4X4 행렬의 곱셈과 덧셈 연산을 클럭 사이클마다 수행합니다. Volta 아키텍처 다음 세대인 튜링 아키텍처 기반의 GPU인 NVIDIA GeForce RTX 2080 Ti(2080ti)는 544개의 Tensor Core와 4352개의 Cuda Core를 제공합니다. NVIDIA 에서는 CUDA: Warp Matrix Multilpy Accumlate(WMMA) API와 cuBLASS GEMM등을 Tensor Core를 이용하여 프로그래밍을 할 수 있도록 제공합니다.  Tencore팀은 여러가지 방법을 통하여 2080ti의 행렬 연산 속도와 결과 값을 비교하였습니다. 16비트 부동소수점으로 표현된 행렬을 Tensor Core와 Cuda Core로 연산했을 때 결과 값이 같았으며 Tensor Core의 행렬 연산속도가 더 빨랐습니다.  2. Tensor Core 소개  <그림1><Tensor Core의 FMA>  32비트의 부동소수점 연산을 수행하는 Cuda Core와 달리 Tensor Core는 16비트의 부동소수점으로 연산 한 뒤 32비트의 부동소수점으로 축적합니다. 16비트 연산은 32비트 연산과 비교하여 절반의 메모리 대역폭(Memory Bandwidth)과 메모리 사용량(Memory Footprint)를 요구합니다. 따라서 Tensor Core는 더 빠른 연산을 수행할 수 있습니다.  -Tensor Core프로그래밍(WMMA)  <그림2>  GPU에서 행렬의 연산은 글로벌 메모리, 공유메모리, 레지스터 단위로 나누어 실행되며  Fragment는 core에서 수행되는 단위이며 WMMA는 텐서코어에서 Fragment단위로 실행할 수 있도록 제공하고 합니다. 다음은 WMMA 이용하여 A\_matrix \* B\_matrix + C\_matrix 를 계산하는 예입니다. 처음에 Fragment를 다음과 같이 선언해야 합니다.  wmma::fragment<wmma::matrix\_a, M, N, K, half, wmma::col\_major> a\_frag;  wmma::fragment<wmma::matrix\_b, M, N, K, half, wmma::col\_major> b\_frag;  wmma::fragment<wmma::accumulator, M, N, K, float> acc\_frag;    앞에서부터 matrix\_a옵션은 행렬 곱해지는 행렬을 나타내며 곱해주는 행렬은 matrix\_b로 설정합니다. 결과가 축적되는 C\_matrix는 accumulator로 설정하여야 합니다. M,N과 K는 a\_matrix가 M X N행렬이며 B\_matrix가 N x K 행렬임을 선언해줍니다. 이때 M,N,K는 16의 배수이어야 합니다. half는 자료형을 설정하는 옵션이며 16비트 부동소수점 자료형을 뜻하고 float는 32비트 부동소수점을 뜻합니다. 따라서 Tensor Core 연산에 사용되는 자료형은 half로 결과가 누적되는 accumulator에 float설정합니다.  wmma::load\_matrix\_sync(a\_frag, A\_matrix, M);  wmma::load\_matrix\_sync(b\_frag, B\_matrix, K);  위와 같은 명령어를 통하여 앞서 선언된 a\_frag에 A\_matirx의 값을 넣어줍니다. 마지막 옵션은 행과 열의 크기입니다.  wmma::mma\_sync(acc\_frag, a\_frag, b\_frag, acc\_frag);.  wmma::store\_matrix\_sync(C\_matrix, acc\_frag, M, wmma::mem\_col\_major);  위와 같은 명령어를 통하여 명령어를 통하여 행렬 연산이 수행되며 C\_matrix에 값이 저장됩니다.  - Tensor Core 프로그래밍 (cublas)  cubals는 cublas\_v2.h을 이용합니다. cublasSetMathMode를 CUBLAS\_TENSOR\_OP\_MATH로 설정하면 Tensor Core 라이브러리를 사용할 수 있습니다. 반대로 CUBLAS\_DEFAULT\_MATH로 설정시에는 라이브러리를 사용할 수 없습니다. 연산은 cubalsGemmEx함수를 이용합니다. 이 함수는 연산되는 행렬 각각의 자료형을 선택할 수 있습니다. 이 함수에서 옵션을 CUBLAS\_GEMM\_DFALT\_TENSOR\_OP으로 설정할 시 Tensor Core로 동작합니다. 반대로 CUBLAS\_GEMM\_DFALT로 설정하면 Cuad Core로 동작합니다.  3. Tensor Core 실험  curand 함수를 통하여 임의의 값을 가지는 행렬을 생성하고 Tensor Core 와 Cuda Core를 이용하여 연산속도와 결과 값을 비교하였습니다. 실험을 위하여 GPU는 NVIDIA GeForce RTX 2080 Ti(2080ti)이며 CUDA 10.0 toolkit에 있는 cublas라이브러리를 사용하였습니다. 2080ti는 544개의 Tensor Core와 4352개의 Cuda Core를 가지고 있어 두 코어의 성능을 비교할 수 있었습니다. cublas 라이브러리는 코어 별 최적화된 행렬연산함수를 제공하기 때문에 코어 별 성능 비교에 사용했습니다.  <플롯1>  <행렬 크기 별 계산한 Tera flops(계산된 행 x 열 x 2 /milliseconds/1e9)>  행렬의 크기 별로 Tflops(계산된 행 x 열 x 2 /milliseconds/10^9)를 측정한 결과 모든 행렬에서 Tensor Core의 연산속도가 더 빨랐으며 행렬의 크기가 증가할수록 차이가 더 많이 발생했습니다.  연산 결과는 각각 16비트와 32비트 부동소수점으로 표현된 행렬을 코어별로 비교하였습니다. 그 결과 32비트 행렬의 값은 평균 0.002364546 표준편차 0.001353507의 오차가 발생하였습니다. 하지만 16비트로 표현된 행렬의 연산결과의 오차는 없었습니다.  4. 결론  32비트 부동소수점연산에서 Tensor core 와 CUDA core에서 연산 결과는 조금씩 오차가 발생합니다. 하지만 16비트 부동소수점연산에서는 결과 값이 같았습니다. flops로 측정한 연산 속도는 실험한 모든 행렬의 크기에서 Tensor Core에 속도가 더 빨랐으며 행렬의 크기가 커질수록 차이가 더 발생했습니다. | | | | | | | | | | | |
| **활용 방안 및 기대효과** | | | | | | | | | | | |
| 딥 러닝 학습은 많은 행렬 연산이 필요하여 시간이 오래 걸립니다. 그래서 신경망 개발에 많은 시간이 소요됩니다. 학습시간을 줄이는 것이 신경망 개발에 있어 중요한 문제입니다. 딥 러닝 학습은 16비트 부동소수점 행렬 연산으로 처리가 가능.[[1]](#footnote-1)합니다. 때문에 Tensor Core를 딥 러닝에 사용한다면 학습 시간을 줄일 수 있습니다. | | | | | | | | | | | |

1. Sharan Narang,et al,” MIXED PRECISION TRAINING”(ICLR 2018) [↑](#footnote-ref-1)