Cuda 강의

Introduction: GPU? Manycore?

인공지능 : 추론 / 학습(GPU)

Manycore : 한 칩에 코어가 여러 개 들어있는 것.

멀티 코어 : 클럭을 올리다 보니 발열이 되어서 문제가 됨(파워 효율이 안 좋아서)

매니 코어 : 클럭을 올리지 않고, 여러 코어를 병렬적으로 사용하면서 발열이 최소화됨

코어는 여러 개 만드는데 병렬(분산)처리가 어렵다.

딥러닝, 빅데이터, GPU의 발전

RTX – 가정용

서버 – 데이터 센터

RTX, 서버용의 차이는 메모리이다.

GPU(그래픽 연산만 사용하는 유닛) -> GPGPU(모든 곳에 사용할 수 있는 그래픽 유닛)

GPU = SIMT라고도 불린다.

Sequentail. Vs Data Parallel vs Thread

Vector SIMD(1개의 명령어로 여러 개 데이터를 처리한다) – intel, amd에서 사용(Vector processing)

* 컴파일러 옵션 설정 + cpu가 사용가능한지 확인 필요

딥러닝에서는 고성능의 GPU비트가 필요하지 않는다.

딥러닝에서 Vector processing이 빛을 낼 수도 있다.

CUDA 코어는 심플한 연산들이 포함된 것

통신은 Global memory로 한다.

SIM

SIMT

Process vs Thread?

Cash memory : CPU안에 있는 메모리

시간적 접근성

CPU : Latency Optimizer, GPU : Throught Optimizer

Parallel Programming on a GPU

For(i=0; i<100; i++)

C[i] = A[i] + B[i];

* GPU는 독립적인 코드를 좋아한다
* 1코어에 10개씩 나눈다.

For(i=0; i<10; i++)

C[CoreIndex \* 10 + i] = A[CoreIndex \* 10 + i] + B[CoreIndex \* 10 + i]

C[10 \* i + CoreIndex] => 가능하지만 메모리 접근이 비효율적이다.(시간적 접근성이 떨어진다)

핵심 : 어떤 코어가 어떤 데이터를 처리할지 할당해줘야 한다.

Cuda 코어를 Stream Processer라고 부르고, Cuda 코어 여러개를 Stream muti processer이라고 불린다.

Total Thread = block수 \* block당 thread 수

WARP : Thread의 묶음

Cord index 대신에 Block index, Thread index로 접근해서 매핑 시켜준다.

C[blockIndex \* 10 + threadIndex] = A[blockIndex \* 10 + threadIndex] + B[blockIndex \* 10 + threadIndex]

Parallel Programming on a Manycore

GPU 연산 기본 절차

1.cudaMalloc

2.cudaMemcpy

=> device -> host / host -> device

3.Kernel launch

=> Kernel은 C언어로 구현해야한다.

4.cudaMemcpy

5.cudaFree

GPU Architecture

* Single Instruction Multiple Thread(SIMT)
* Manycore Vector Processing
* Throught-Oriented Accelerating Architecture

Graphics memory : higher bandwidth

Fermi GPU : 16SM(Stream Muti Processing) = 512 Cores

Kepler GPU : 16SMX = 2880 cores (SMX = 192 CUDA cores)

Pascal GPU : 56SM

Maxwell GPU

Volta CPU : tensor core 도입

* Tensor core : 4x4 행렬을 곱해주고 더해주는 작업을 1번에 진행
* 딥러닝을 위해 Tensor core가 도입

Turing GPU

Tmpere GPU : bfloat16

Ada lovelace GPU

Hopper GPU : Transformer engine

Tesla GPU

Jetson TX1 and TX2

Jetson Xavier

Jetson Orin

For(i=0; i<n; i++) y[i] = a\*x[i] + y[i];

Int I = threadIdx.x + blockDim.x \* blockIdx.x;

If(i<n)y[i] = a\*x[i] + y[i];

For문을 threadIdx와 blockIdx를 사용하면 시간 복잡도를 줄일 수 있다.