Parallel Machine Learning and Artificial Intelligence

Dr. Handan Liu

h.liu@northeastern.edu

Northeastern University

Content

- GPU and CUDA, and Deep Learning

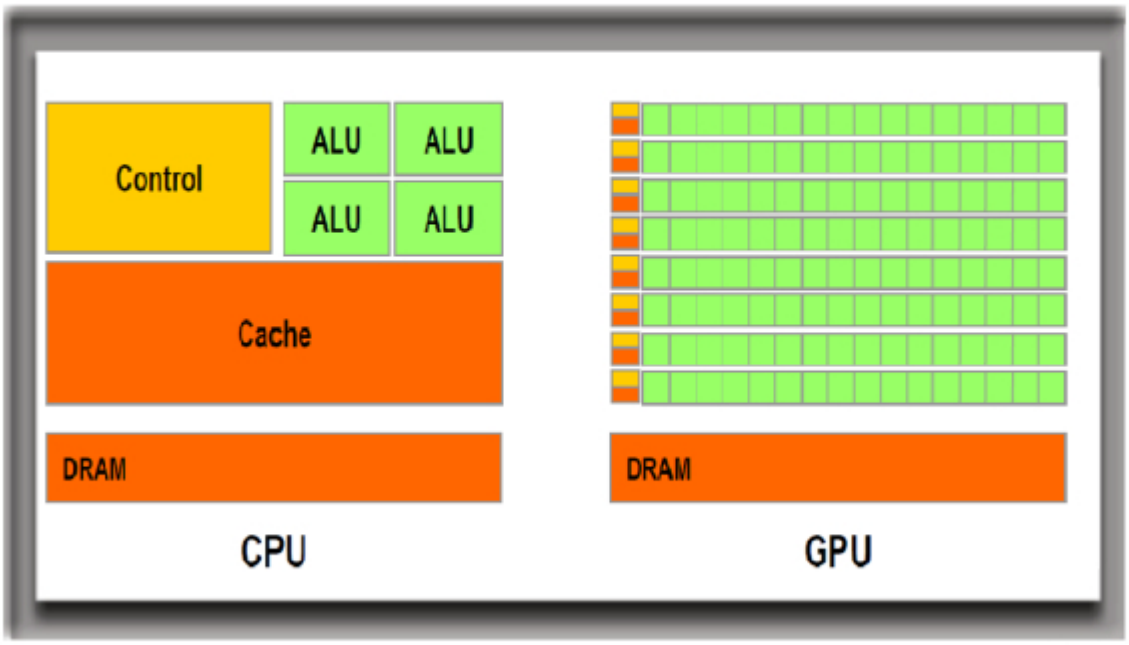
Graphics Processing Unit (GPU)

- A GPU is a processor that is good at handling *specialized* computations.

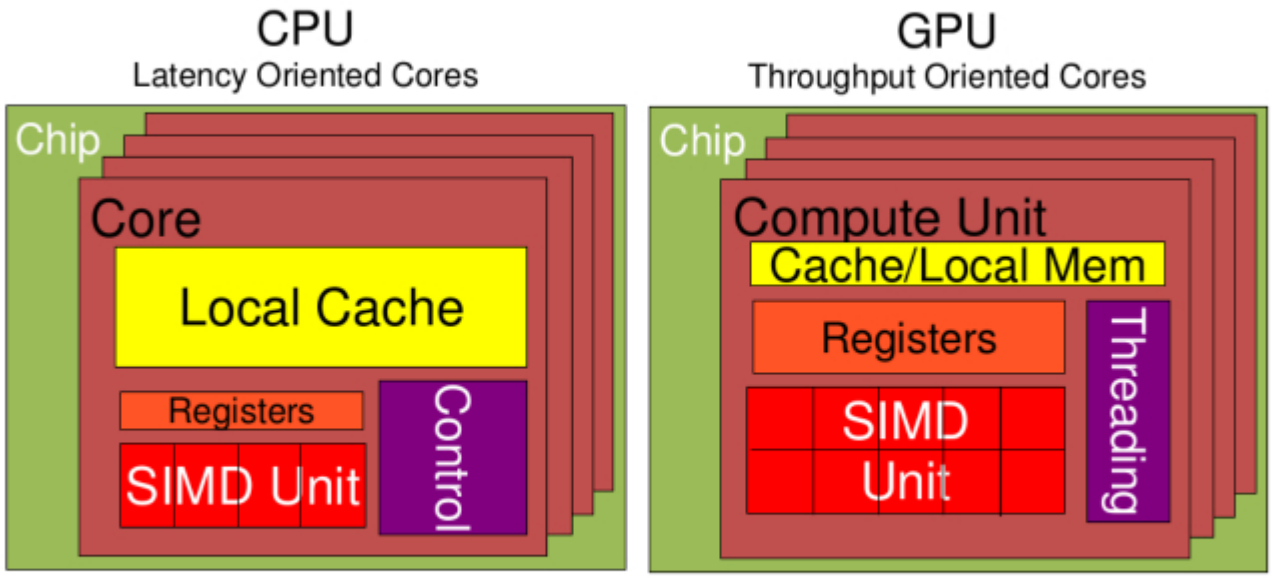
- GPU are specialized processors that are very good at solving some problems and not very good at solving others!

Isn't a GPU faster than a CPU?

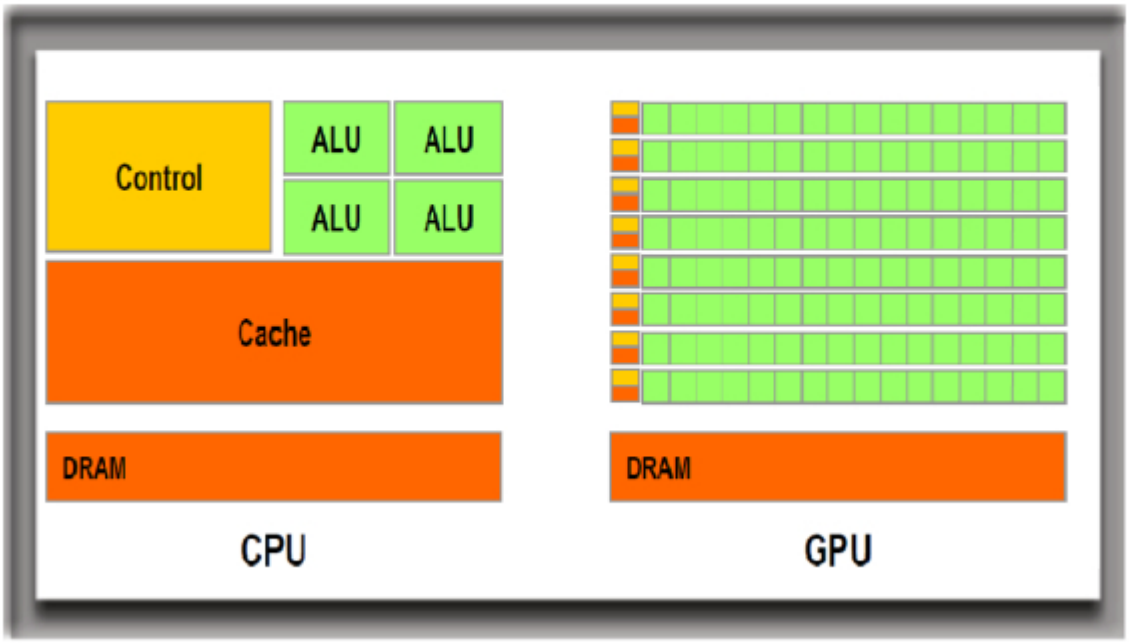
CPU vs GPU Processing



Courtesy of Nvidia CUDA

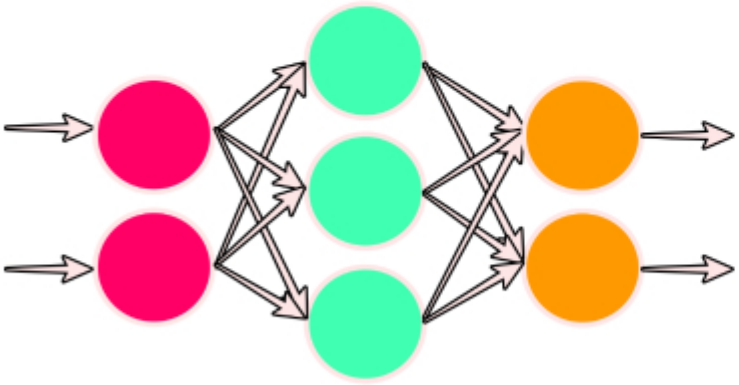


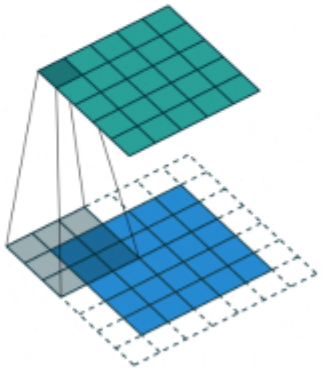
Difference



- What type of program is suitable for running on the GPU?
  1. Computationally intensive procedures.
  2. Easy parallel program.

Why does Deep Learning and Neural Networks use GPUs?





- Blue (bottom) - Input channel
- Shaded (on top of blue) - 3 x 3 convolutional filter
- Green (top) - Output channel

CUDA

- CUDA is a parallel computing platform and programming model
- Programming in CUDA is typically done in Fortran, C, or C++
  - Support for other low-level languages is available (e.g. OpenCL, OpenACC)
  - PyCUDA wrapper allows CUDA functionality from Python code (written in C++, so very little performance degradation)

- CUDA programming typically follows the following general workflow

CUDA Workflow

- Example CUDA code: Saxpy
  - Single-precision A*X Plus Y
- Split into **Host** and **Device** code
- General idea:
  - Allocate memory on both **Host** and **Device**
  - Initialize variables on **Host**
  - Copy data from **Host** to **Device**
  - Perform parallel computation on **Device**
  - Copy data back from **Device** to **Host**
  - Free memory on both **Host** and **Device**

```cpp
#include <stdio.h>

__global__
void saxpy(int n, float a, float *x, float *y)
{
  int i = blockIdx.x*blockDim.x + threadIdx.x;
  if (i < n) y[i] = a*x[i] + y[i];
}

int main(void)
{
  int N = 1<<20;
  float *x, *y, *d_x, *d_y;
  x = (float*)malloc(N*sizeof(float));
  y = (float*)malloc(N*sizeof(float));

  cudaMalloc(&d_x, N*sizeof(float));
  cudaMalloc(&d_y, N*sizeof(float));

  for (int i = 0; i < N; i++) {
    x[i] = 1.0f;
    y[i] = 2.0f;
  }

  cudaMemcpy(d_x, x, N*sizeof(float), cudaMemcpyHostToDevice);
  cudaMemcpy(d_y, y, N*sizeof(float), cudaMemcpyHostToDevice);

  // Perform SAXPY on 1M elements
  saxpy<<<(N+255)/256, 256>>>(N, 2.0f, d_x, d_y);

  cudaMemcpy(y, d_y, N*sizeof(float), cudaMemcpyDeviceToHost);

  float maxError = 0.0f;
  for (int i = 0; i < N; i++)
    maxError = max(maxError, abs(y[i]-4.0f));
  printf("Max error: %f\n", maxError);

  cudaFree(d_x);
  cudaFree(d_y);
  free(x);
  free(y);
}
```

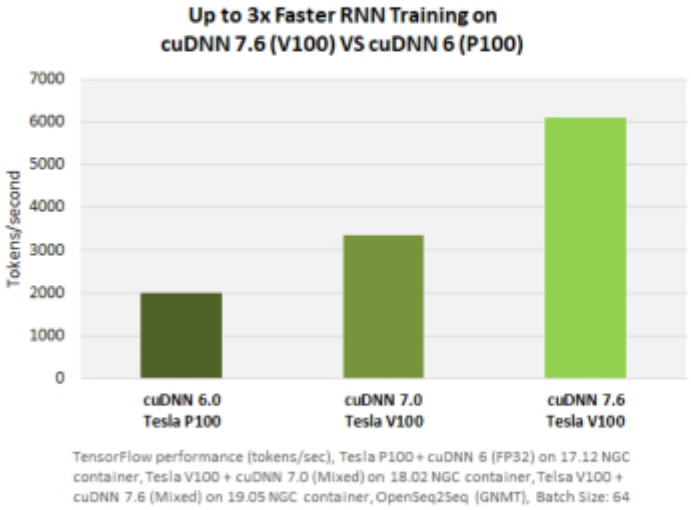
CUDA Toolkit – Underlying Libraries

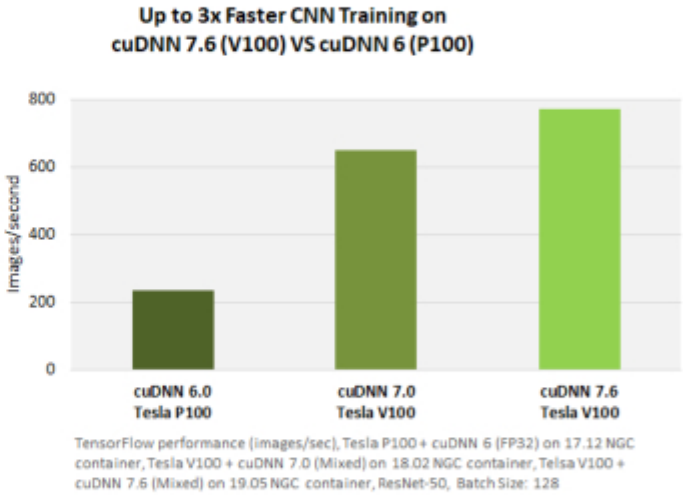
- cuBLAS – CUDA Basic Linear Algebra Subroutines library
- CUDART – CUDA Runtime library
- cuFFT – CUDA Fast Fourier Transform library
- cuRAND – CUDA Random Number Generation library
- cuSOLVER – CUDA based collection of dense and sparse direct solvers
- cuSPARSE – CUDA Sparse Matrix library
- NPP – NVIDIA Performance Primitives library
- nvGRAPH – NVIDIA Graph Analytics library
- NVML – NVIDIA Management Library
- NVRTC – NVIDIA Runtime Compilation library for CUDA C++
- nvJPEG – Hybrid JPEG Processing [new in cuda 10+]

DNN Library – cuDNN: Heart of Convolutions

- NVIDIA CUDA® Deep Neural Network library - cuDNN
- cuDNN is a GPU-accelerated library of primitives for deep neural networks.
- cuDNN provides highly tuned implementations for standard routines such as forward and backward convolution, pooling, normalization, and activation layers.
- cuDNN supports high-performance GPU acceleration.
- cuDNN accelerates widely used deep learning frameworks, including:
  - PyTorch, MXNet, TensorFlow, MATLAB, Caffe,Caffe2, Chainer, Keras, etc.

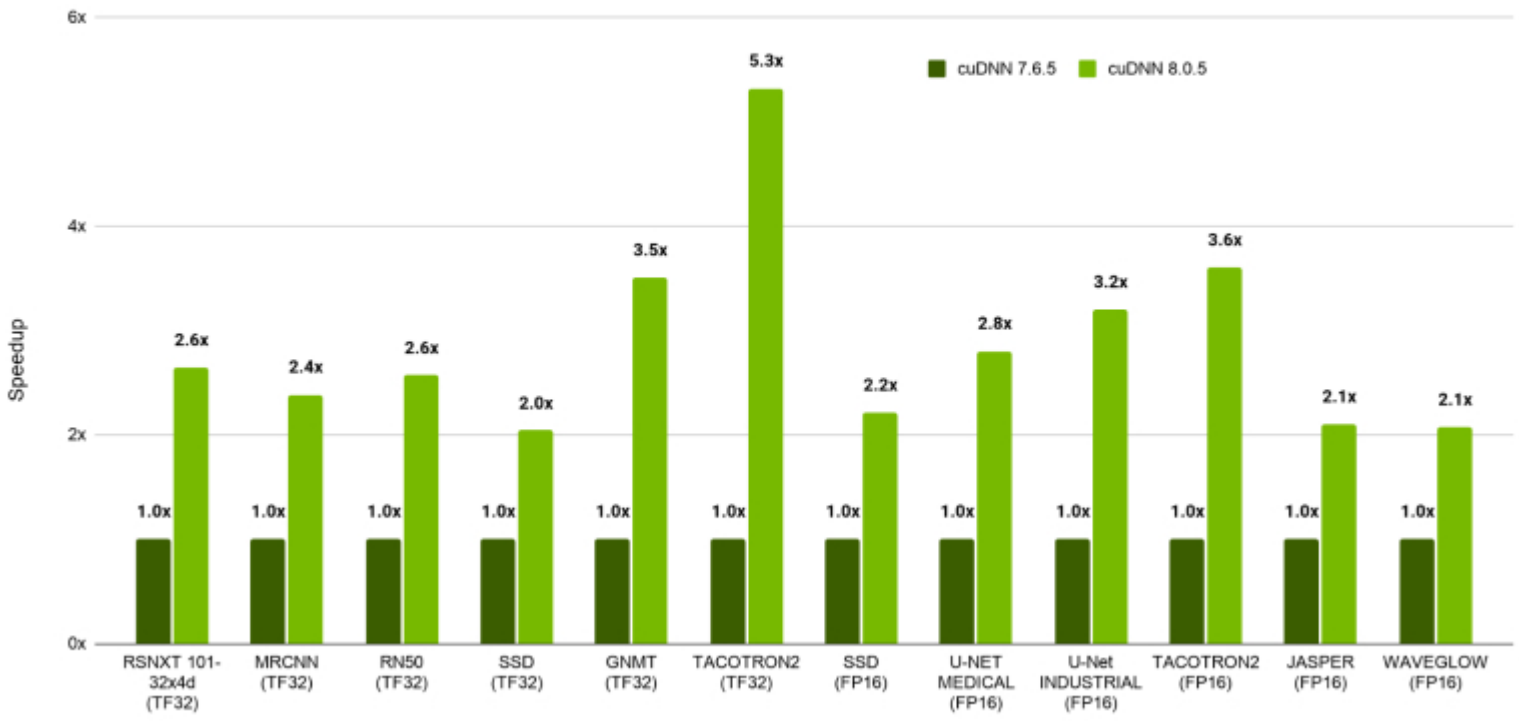
Where does the Performance come from?





Courtesy: https://developer.nvidia.com/cudnn

A100 OVER 5X FASTER THAN V100 WITH CUDNN 8

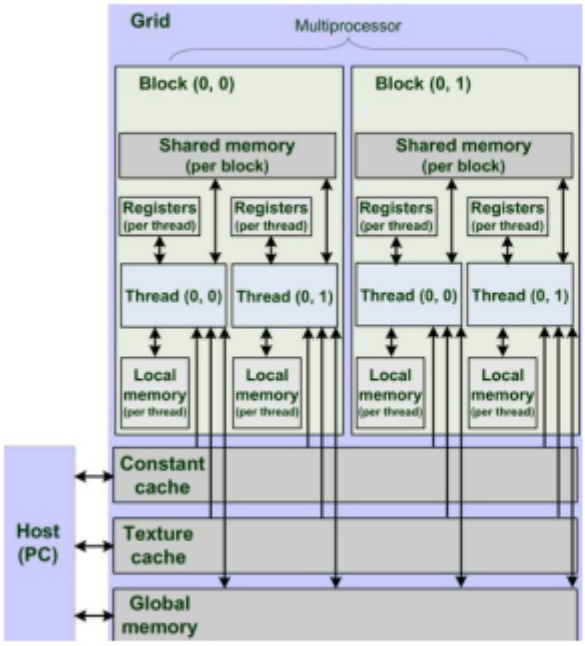


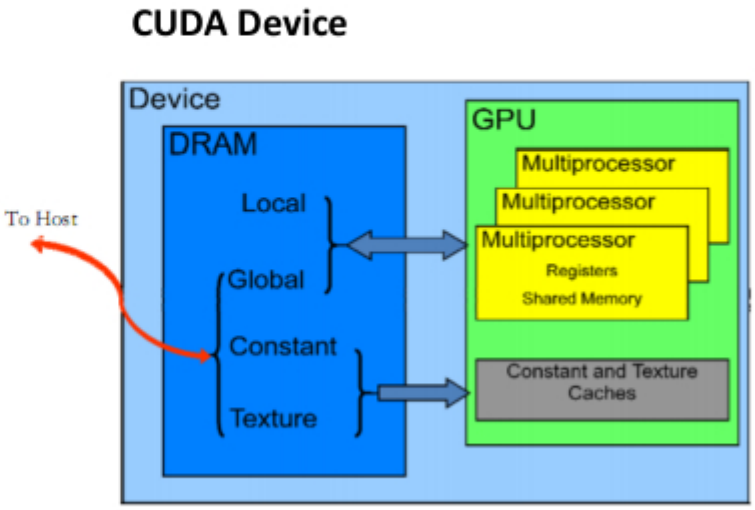
CUDA Programming Models

- Host and Device

- In the CUDA program architecture,
  - the main program is still executed by the CPU, and
  - when the data is processed in parallel, CUDA will compile the program into a program that the GPU can execute and transfer it to the GPU.

GPU Hardware and Memory Hierarchy





- Two types according to off or on chip:
  - Reside on the GPU chip are register and shared memory.
  - Local, Global, Constant, and Texture memory all reside off chip. Local, Constant, and Texture are all cached.
- In terms of speed, if all the various types of device memory were to race here's how the race would turn out:
  - 1st place: Register file
  - 2nd place: Shared Memory
  - 3rd place: Constant Memory
  - 4th: Texture Memory
  - Tie for last place: Local Memory and Global Memory

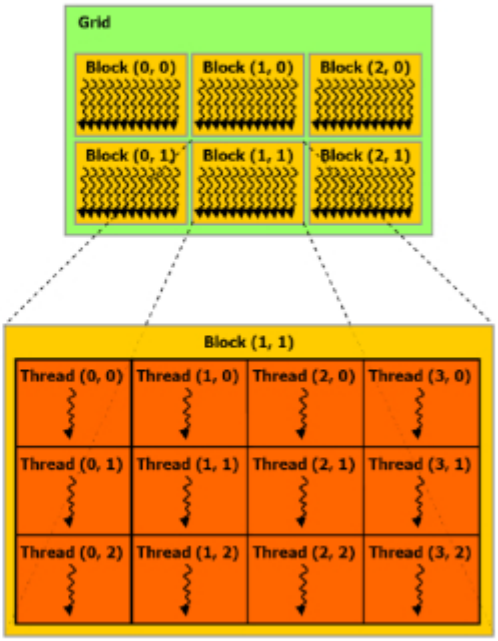
From a Hardware Perspective,

- SP: Streaming Processor, also known as the CUDA core
- SM: Streaming Multiprocessor, also called GPU core
- CUDA allocates the register and shared memory (scarce resource of SM) to all threads, which limits the parallelism capability.

From a Software Perspective,

- Thread — A CUDA parallel program will be executed in many threads.
- Block — Several threads are grouped into a block, and threads in the same block can be synchronized or communicated through shared memory.
- Grid — Multiple blocks will form a grid.
- Warp — The scheduling unit is fixed to 32.

SIMT: single instruction multiple threads.



Reservations

- CPU nodes
  - Reservation name: csye7105
  - 6 nodes
  - Commands:
    - ✔ $ srun -p reservation --reservation= csye7105 --time=01:00:00 --pty /bin/bash

- GPU nodes
  - Reservation name: csye7105-gpu
  - 3 nodes with multiple GPUs: maximum to 8 GPUs
  - Commands:
    - ✔ $ srun --reservation= csye7105-gpu --gres=gpu:1 --pty /bin/bash
    - ✔ $ srun -p gpu --gres=gpu:1 --pty /bin/bash

Getting Started with CUDA on the Cluster

- https://rc-docs.northeastern.edu/en/latest/using-discovery/workingwithgpu.html

- On GPU node, load cuda module and run commands to check the detailed information of GPU drivers.
  - Load CUDA module to check the details of GPU device / CUDA driver
  - CUDA versions on Discovery: 9.0, 9.2, 10.0, 10.2, 11.0
    - ✔ $ module avail cuda
    - ✔ $ module load cuda/10.0 (For examples)

NVIDIA System Management Interface

- The NVIDIA System Management Interface (nvidia-smi) is a command line utility, based on top of the NVIDIA Management Library (NVML), intended to aid in the management and monitoring of NVIDIA GPU devices for for each of NVIDIA's Tesla, Quadro, GRID and GeForce etc.
- NVSMI is a cross platform tool that supports all standard NVIDIA driver-supported platforms.
- Note that much of the functionality of NVSMI is provided by the underlying NVML C-based library.
  - $ nvidia-smi

- Stay safe!
- See you next class!

Next Lecture will Continue:

Parallel Deep Learning Overview

Data Structures of Deep Learning