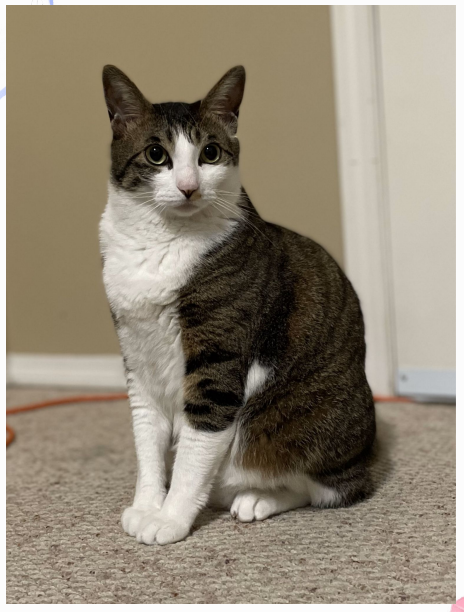
Introduction to Nvidia CUDA

Patrick Yi

Hello!

- My name is Jeongseop (Patrick) Yi
- Currently in 4B CFM at UWaterloo
- CUDA Background:
  - Heterogeneous Programming at POSTECH in Korea last Fall!
- Prev @ Manulife, Geotab, Ford



pjyi2147

patrickyi-0427

patrick@patrickyi.xyz

Workshop Prerequisites

Requirements:

- Google account for Google Colab and Drive
- Strong enthusiasm regarding the topic!

Nice to know:

- Basic C/C++ knowledge (pointers, malloc/free, int main(), etc)
- Basic Computer Architecture (namely Von Neumann Arch. - CPU/RAM)

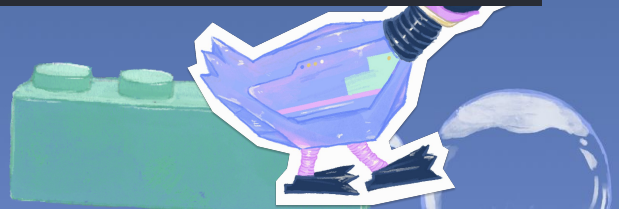
Agenda

1. What is CUDA? [5 min]
2. GPU paradigm [5 min]
3. CUDA Syntax [10 min]
4. Code [25 min]
5. Future topics + Q&A [5 min]

What is CUDA?

- Programming Language for Nvidia GPUs
- CUDA stands for Compute Unified Device Architecture
  - Is it really used? I doubt so…
- A dialect of C/C++, usually referred as CUDA C/C++
- Accelerator for “any” parallelizable tasks
- Heavily used in Machine Learning acceleration
  - PyTorch, Tensorflow, etc

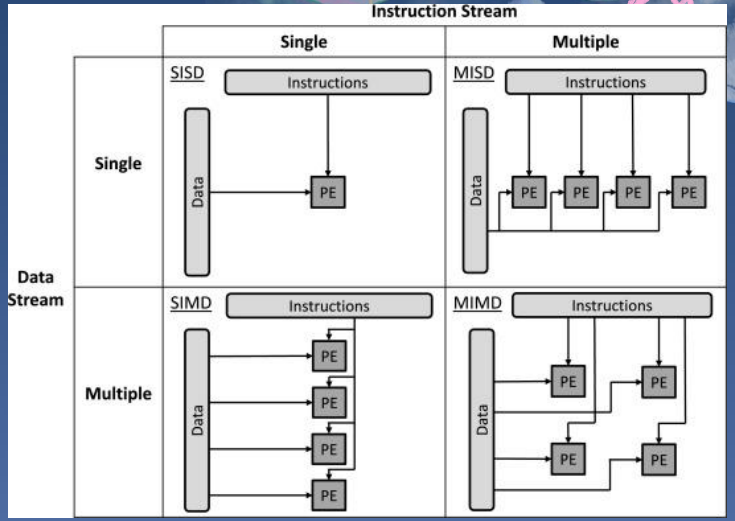
```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```



CUDA/GPU Programming paradigm

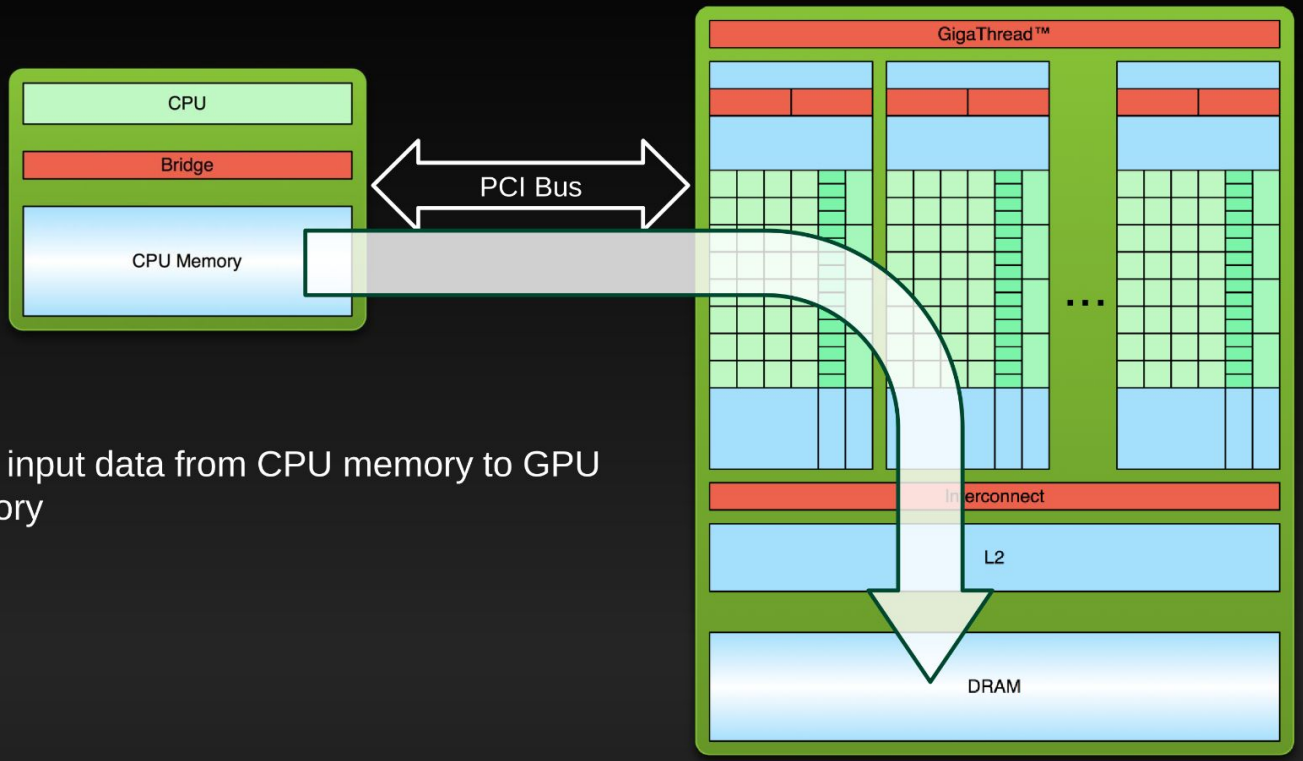
- Host (CPU) / Device (GPU) Mechanism
- SIMT (Single Instruction Multi Threads)

1. Allocate memory to CPU and GPU
2. Copy data from CPU to GPU
3. Load GPU program and execute
4. Copy results from GPU to CPU
5. Free memory



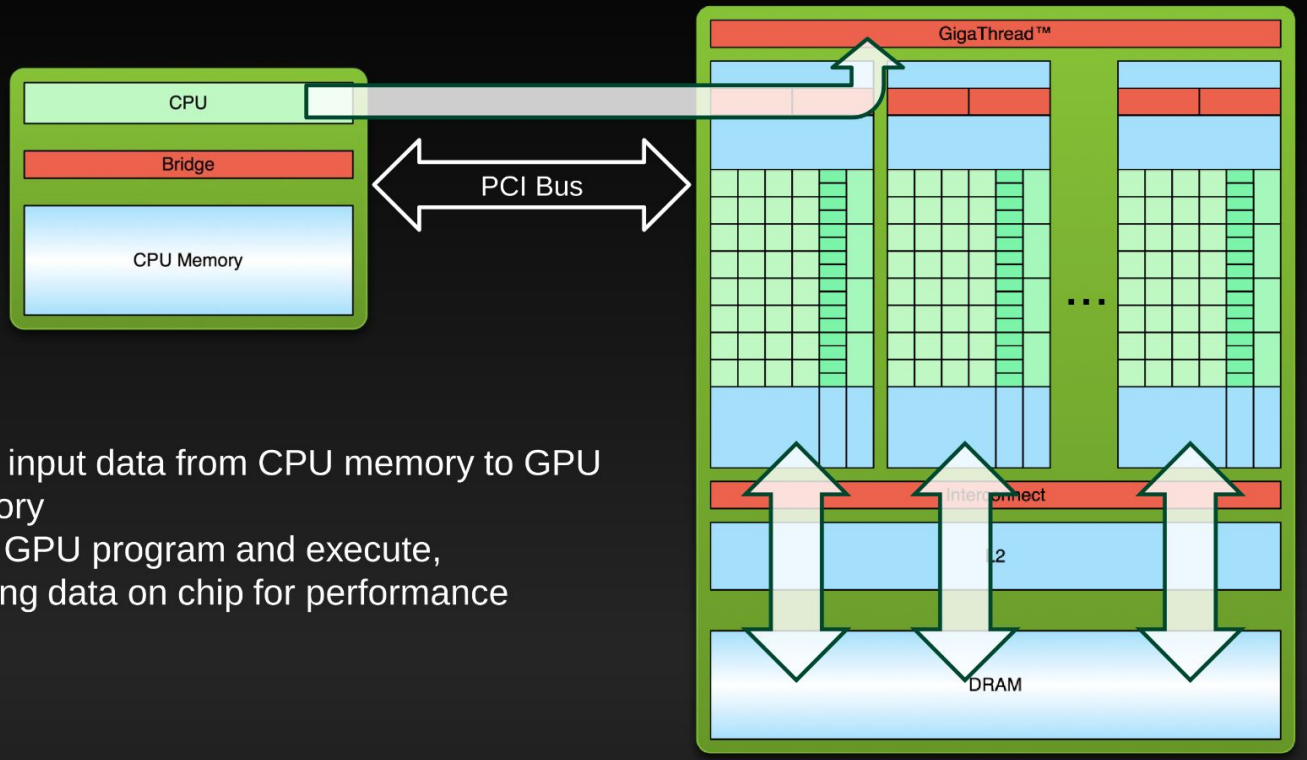
Flynn's Taxonomy

Simple Processing Flow



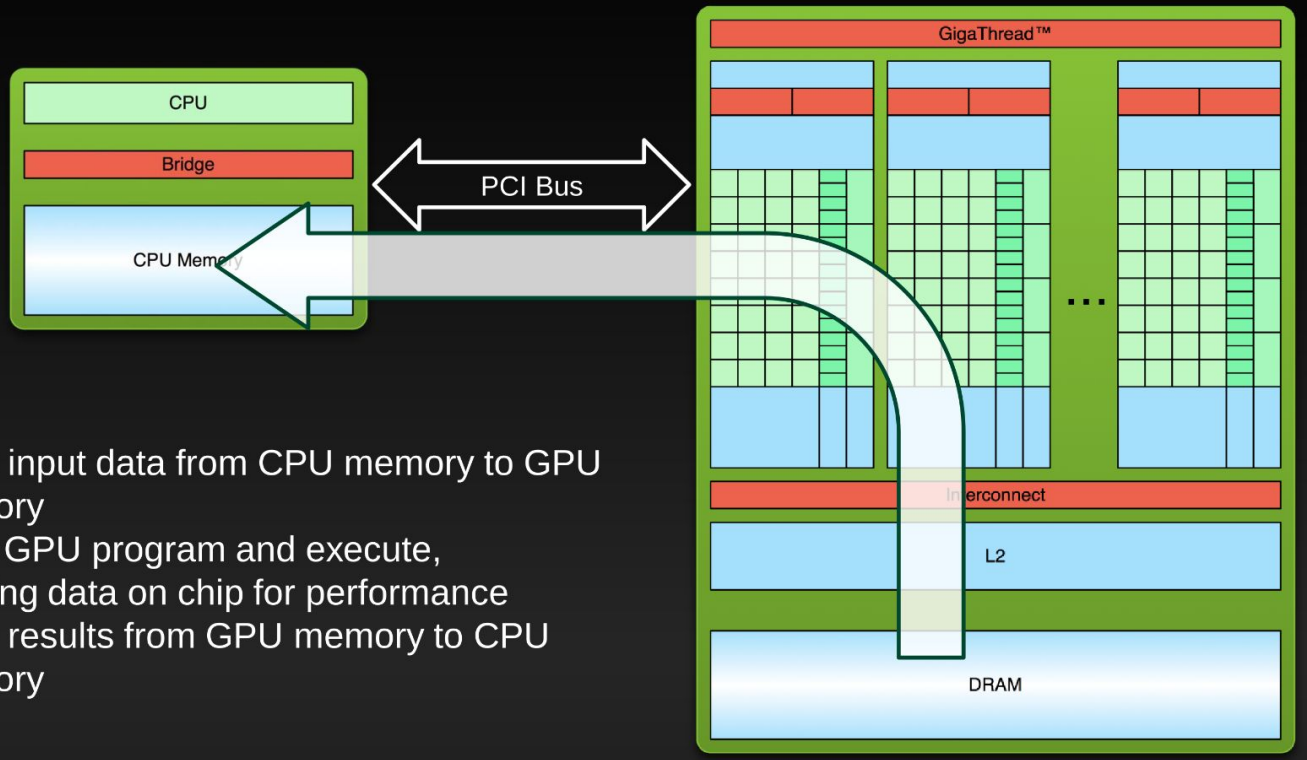
1. Copy input data from CPU memory to GPU memory

Simple Processing Flow



1. Copy input data from CPU memory to GPU memory
2. Load GPU program and execute, caching data on chip for performance

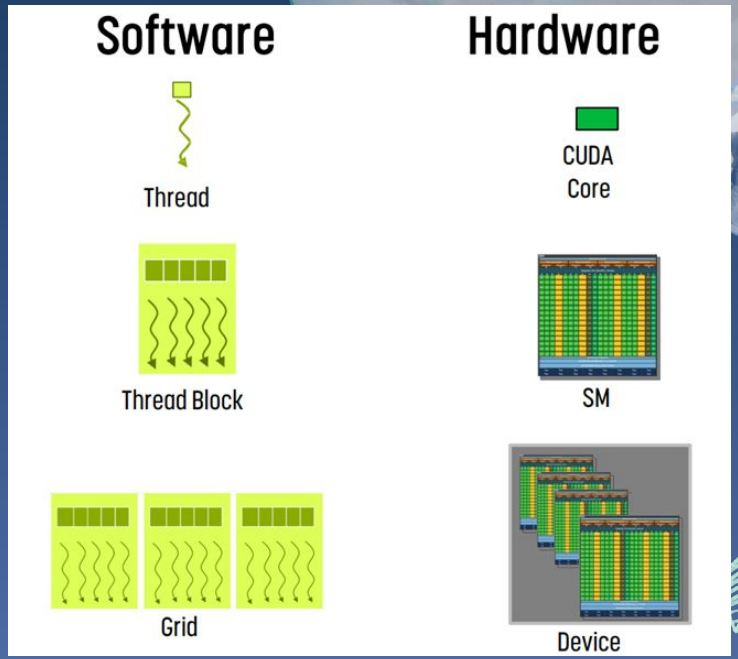
Simple Processing Flow

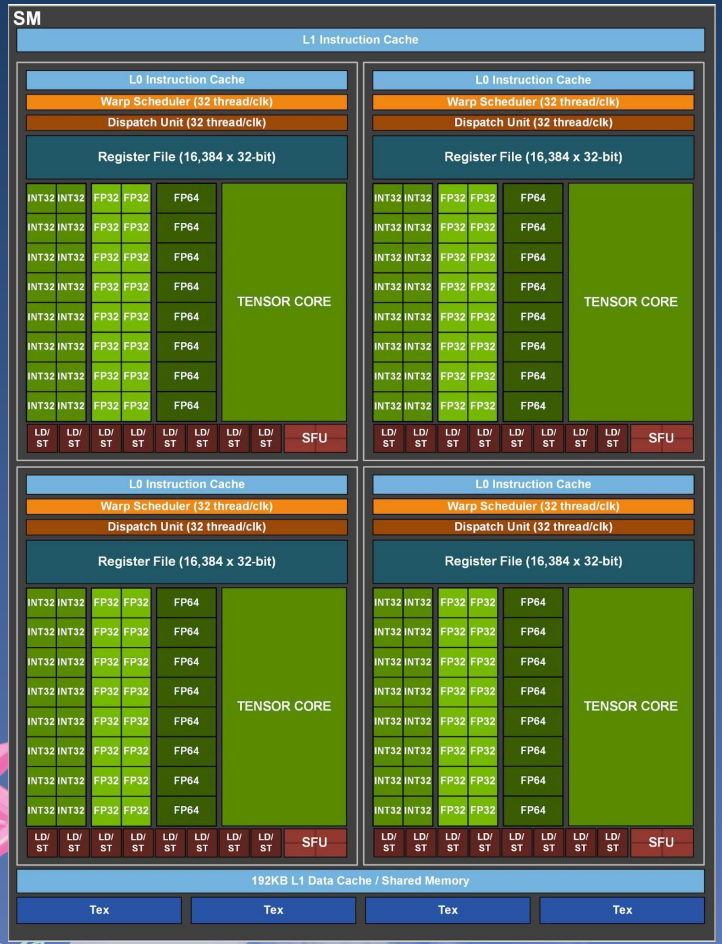


1. Copy input data from CPU memory to GPU memory
2. Load GPU program and execute, caching data on chip for performance
3. Copy results from GPU memory to CPU memory

CUDA/GPGPU Architecture

- Terminologies
  - Kernel: GPU functions (SW)
  - CUDA cores (HW)
  - Warps (HW)
  - SM (Streaming Multiprocessor, HW)
  - Thread (SW)
  - Thread Block (SW)
  - Grid (SW)
- Kernel reserved keywords
  - __global__ / __device__ / __shared__ / __host__
  - We will be using only __global__ today





GA100 Streaming Multiprocessor (SM)

GH100 Streaming Multiprocessor (SM)

GH100 Full GPU with 144 SMs

https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf

https://resources.nvidia.com/en-us-tensor-core?ncid=no-ncid

CUDA Kernel Syntax (1/2)

- Kernel Definition

```
__global__ void testkernel(float* param1, float* param2, float* ret) {
  // GPU Code here...
}
```

- Things to notice:
  - __global__ keyword indicates that this is a GPU kernel to the compiler
  - CUDA kernels usually have `void` as return type
    - Must have a return parameter/pointer
  - All parameters are pointers to DEVICE(GPU) memory
    - Must be initialized somewhere in Host(CPU) code!

CUDA Kernel Syntax (2/2)

- Kernel Execution

```
int main() {
  // CPU Code here...
  testkernel<<<BLOCK_NUM, BLOCK_SIZE>>>(d_float1, d_float2, d_ret);
  cudaDeviceSynchronize();
  // More CPU code
}
```

- Things to notice:
  - <<<>>> block
    - It corresponds to the number of thread block and thread
  - Parameters must reside in GPU memory
    - cudaMalloc, cudaMemcpy, cudaFree (Usages later)
  - GPU kernel call is asynchronous to the CPU code
    - cudaDeviceSynchronize

Let’s code!

Open up Google Colab!

What’s next? (1/2)

Memory Architectures

- Shared memory (SMEM, Can I share memory between threads faster?)
- Memory Coalescing (Why coalesced memory is important?)
- Asynchronous memory (Should GPU memory allocation block CPU?)
- Unified memory (Do I even need to care memory transfers?)

Parallel Algorithm

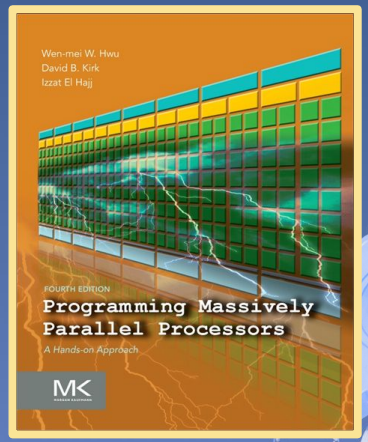
- 1D Reduction (max, min, sum, etc)
- SpMV (Sparse Matrix-Vector Multiplication)
- 1D/2D Convolution (CNN)

What’s next? (2/2)

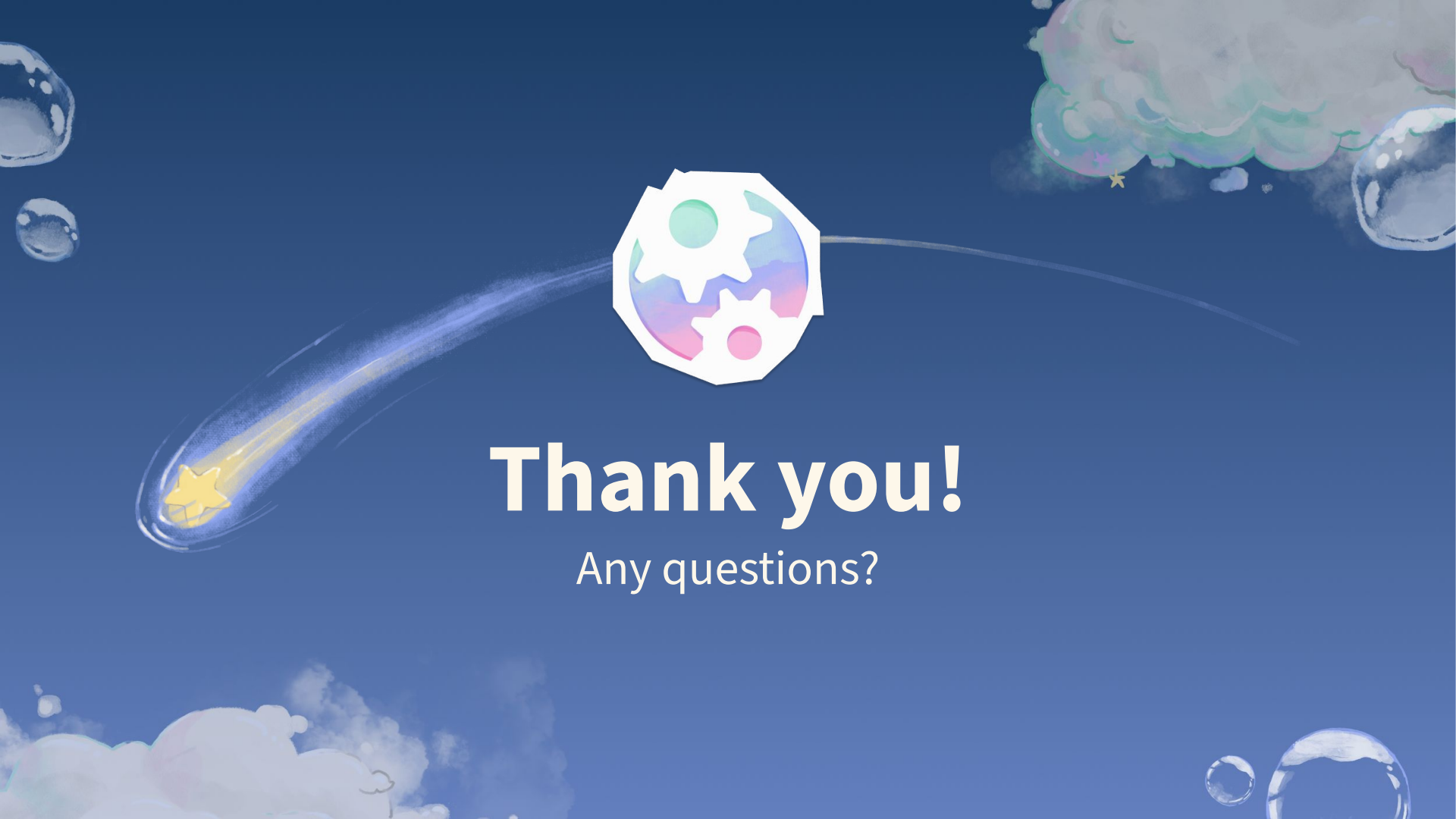
Nvidia Whitepapers:

- https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf
- https://resources.nvidia.com/en-us-tensor-core?ncid=no-ncid

Programming Massively Parallel Processors 3/4th Edition:



Answers for Google Colab exercises in `answer` branch


Thank you!
Any questions?