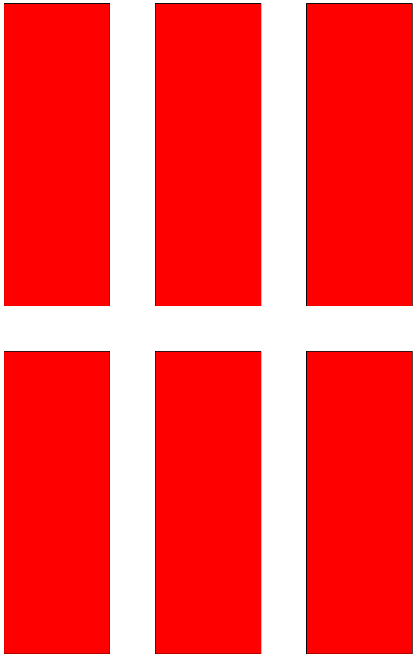
PHAS0102: Techniques of High-Performance Computing

GPUs and CPUs

- CPU = central processing unit
  - Designed to be good at everything a computer needs to do
- GPU = graphical processing unit
  - Designed to be good at processing 3D graphics

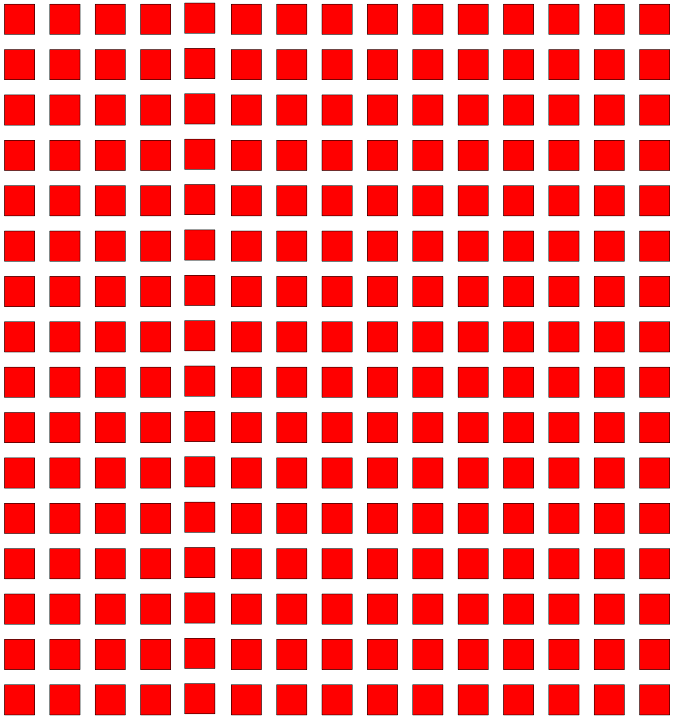
GPUs and CPUs

CPU



A few powerful cores

GPU



A lot of smaller cores

GPUs

- Can do many parallel flops at the same time
- Usually only fast for single precision computations
- Copying memory to a GPU can take time, and GPUs have limited storage

Common GPU manufacturers

- Nvidia
- AMD
- Intel

Programming on a GPU

- Cuda
  - Nvidia specific GPU API
  - Can be used from Python, C, C++, Fortran, Matlab, Julia, and more
- OpenCL
  - Open → can be used on all platforms
  - Can run on GPUs or CPUs
- SYCL
  - More modern open standard

Programming on a GPU

- OneApi
  - Developed by Intel
  - Cros platform
- OpenACC
  - Can be used from C, C++ and Fortran
  - Used on many large HPC systems
- OpenMP
  - Can be used from C, C++ and Fortran
  - First developed for CPUs but more recent versions also support GPUs

Programming on a GPU

- My personal recommendation

```
if you’re using an Nvidia GPU:
    Cuda
elif you’re using C++:
    SYCL
else:
    OpenCL
```

pycuda and **pyopencl**

- These two libraries allow you to use Cuda and OpenCL directly from Python.
- There are examples in the lecture notes.

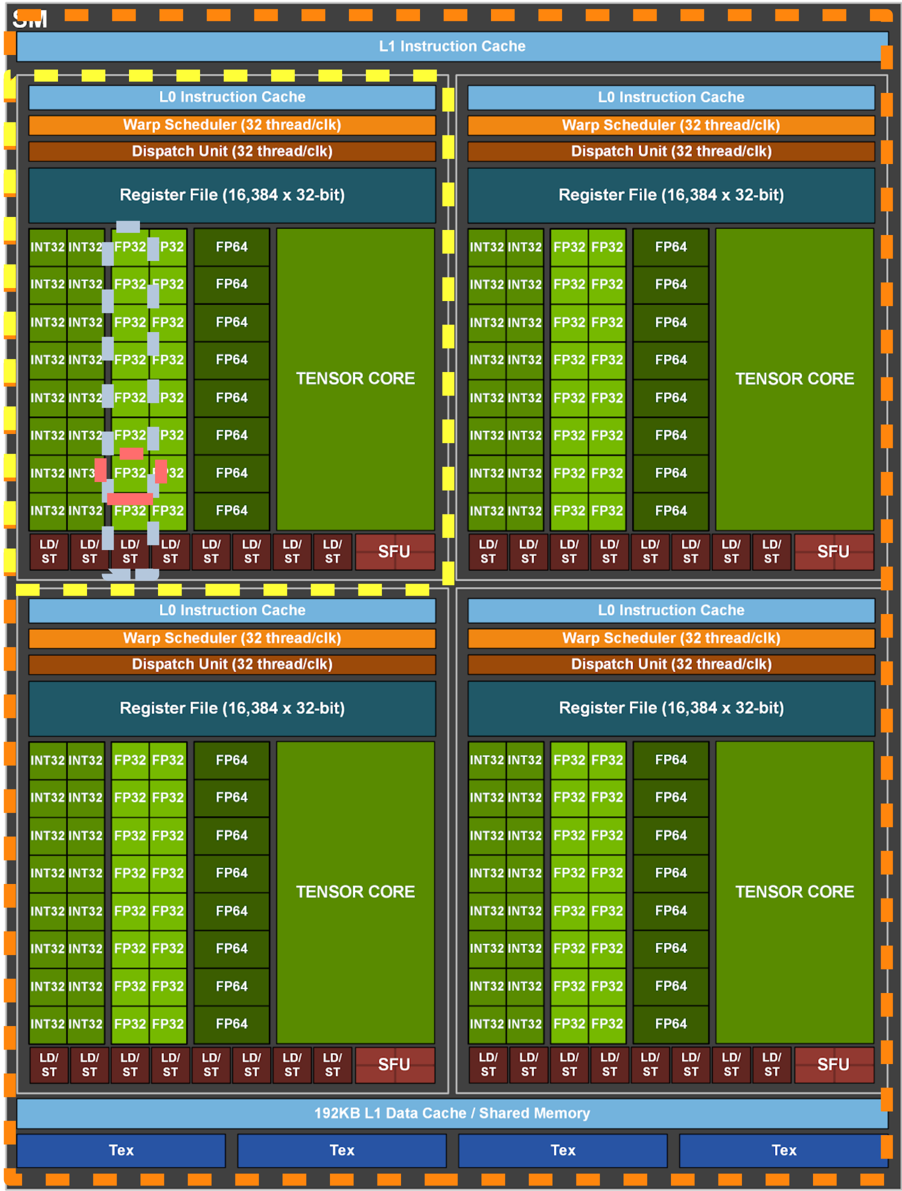
Using Cuda with Numba

```
from numba import cuda
```

[live Numba & CUDA demo]

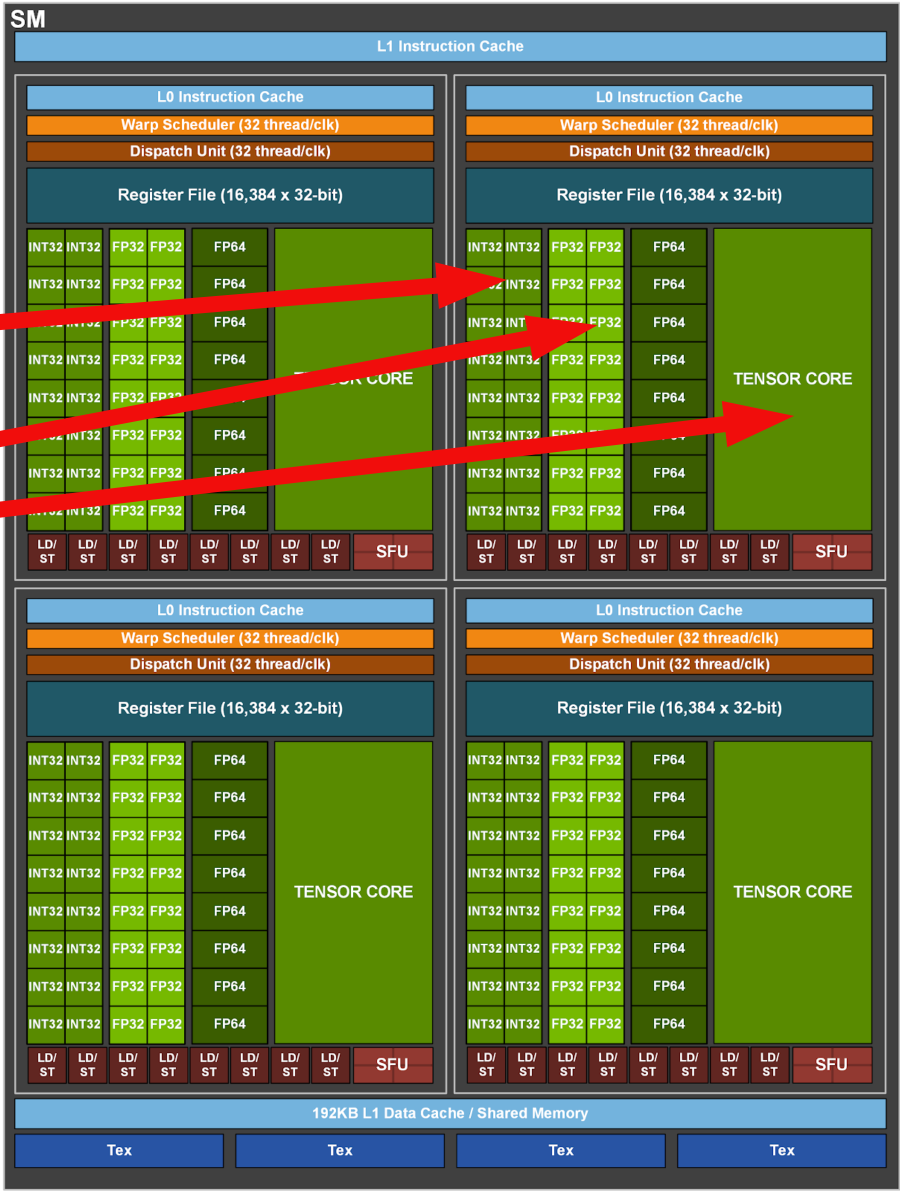
Cuda device model

- Streaming multiprocessor (SM)
  - GPUs are made up of multiple SMs
- Warps
  - A collection of blocks
  - Each thread in a warp must follow the same execution path
- Blocks
  - A collection of threads
- Thread
  - Threads are where calculations are actually done


SM
L1 Instruction Cache
L0 Instruction Cache
Warp Scheduler (32 thread/clk)
Dispatch Unit (32 thread/clk)
Register File (16,384 x 32-bit)
INT32
FP32
FP64
TENSOR CORE
LD/ST
SFU
192KB L1 Data Cache / Shared Memory
Tex

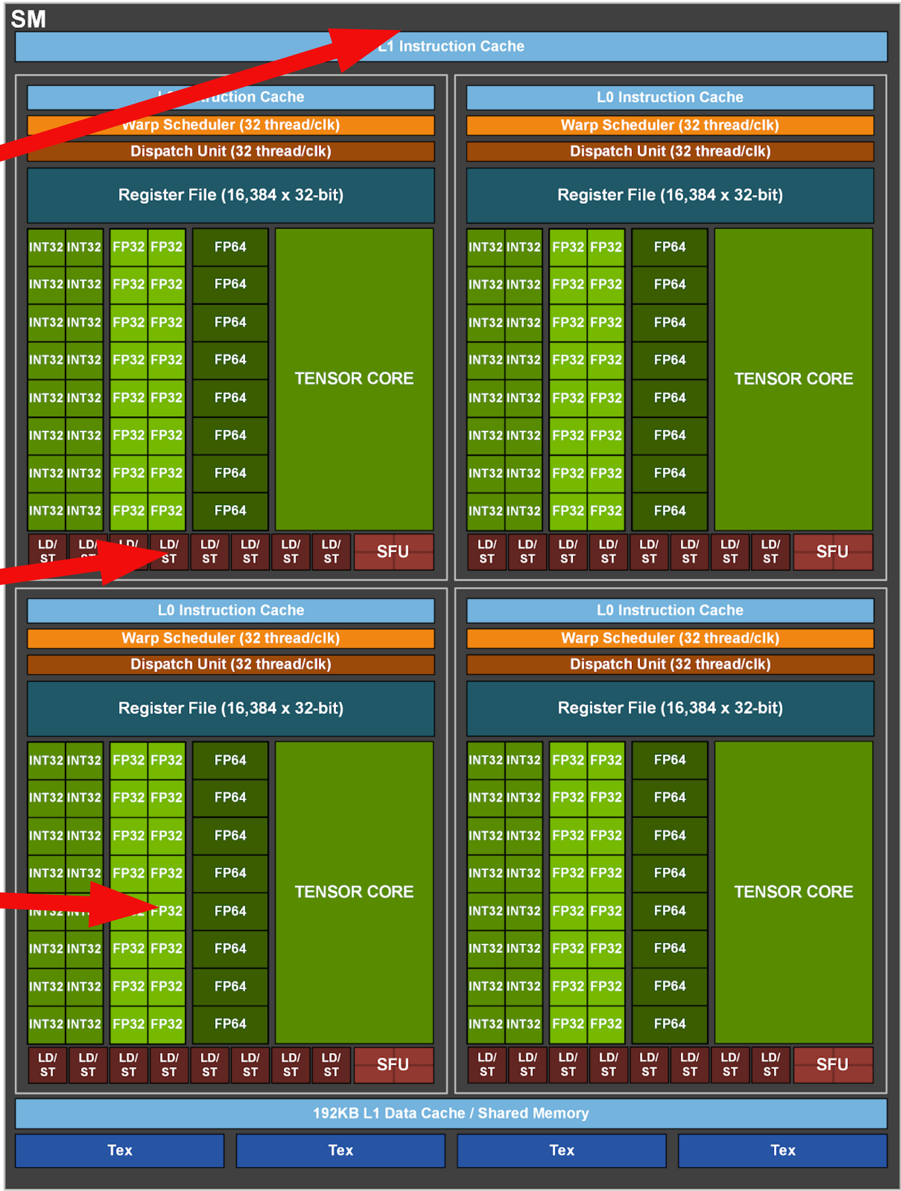
Cuda device model

- Threads for integer calculations
- Threads for float calculations
- Tensor threads

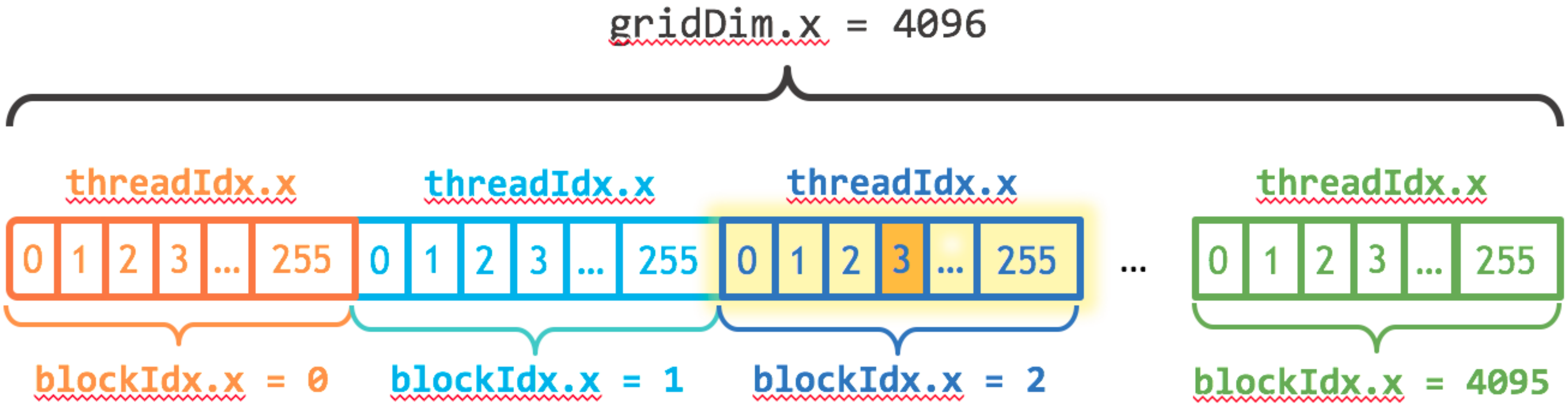

SM
L1 Instruction Cache
L0 Instruction Cache
Warp Scheduler (32 thread/clk)
Dispatch Unit (32 thread/clk)
Register File (16,384 x 32-bit)
INT32
FP32
FP64
TENSOR CORE
LD/ST
SFU
192KB L1 Data Cache / Shared Memory
Tex

Cuda device model

- Global memory
  - Access from threads to global memory is slow
- Shared memory
  - Shared within a block
- Private memory
  - Used by a thread during calculations


SM
L1 Instruction Cache
L0 Instruction Cache
Warp Scheduler (32 thread/clk)
Dispatch Unit (32 thread/clk)
Register File (16,384 x 32-bit)
INT32
FP32
FP64
TENSOR CORE
LD/ ST
SFU
192KB L1 Data Cache / Shared Memory
Tex

Thread numbering



gridDim.x = 4096

threadIdx.x

0 1 2 3 … 255

blockIdx.x = 0

threadIdx.x

0 1 2 3 … 255

blockIdx.x = 1

threadIdx.x

0 1 2 3 … 255

blockIdx.x = 2

…

threadIdx.x

0 1 2 3 … 255

blockIdx.x = 4095

```
index = blockIdx.x  * blockDim.x + threadIdx.x

index =     (2)     *    (256)   +     (3)      = 515
```

- In this example, threads are arranged in a line. Threads could also be arranged into a 2D or 3D array.