Introduction to GPU programming (part 2)

By Kevin Fotso

Recap

Last time we learned
what was a GPU and
why it is needed

Recap

Last time we learned what was a GPU and why it is needed

- The architecture of a GPU

Recap

Last time we learned what was a GPU and why it is needed

- The architecture of a GPU
- Fundamentals of heterogenous computing

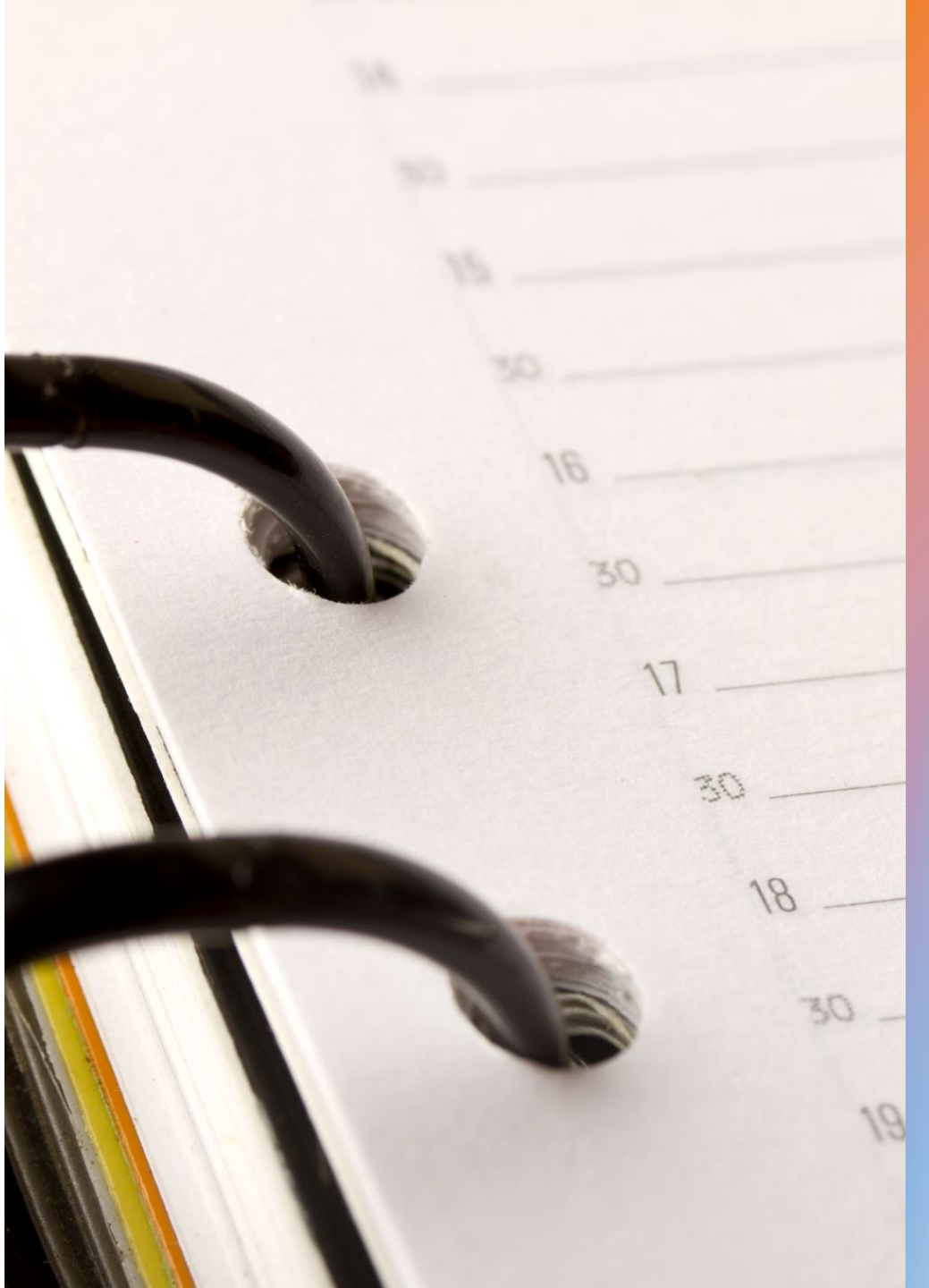
Recap

Last time we learned what was a GPU and why it is needed

- The architecture of a GPU
- Fundamentals of heterogenous computing
- Fundamental terms related with cuda

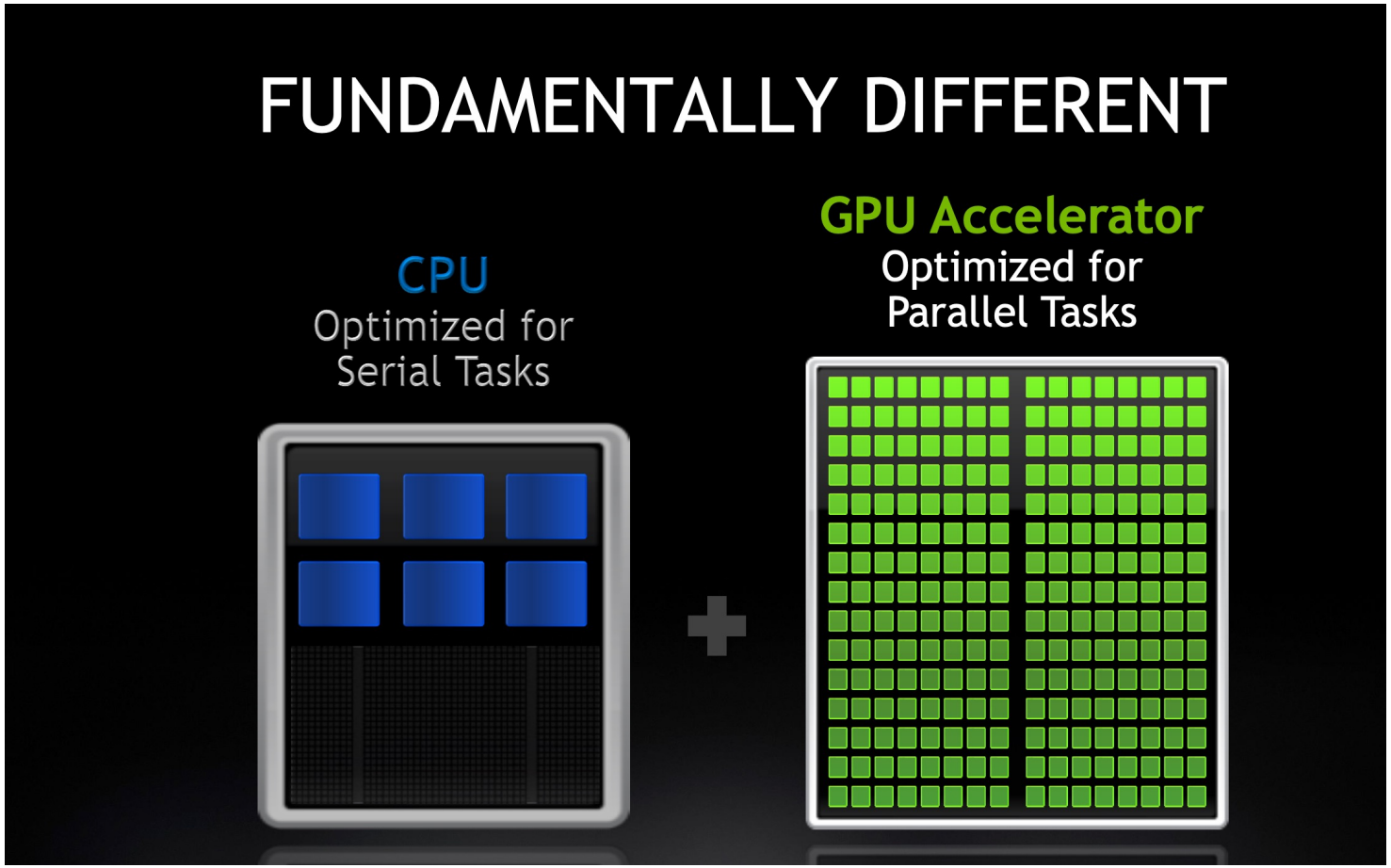
Today

- We will do a short recap
- Then learn about numba
- Finally about cupy.



I - RECAP

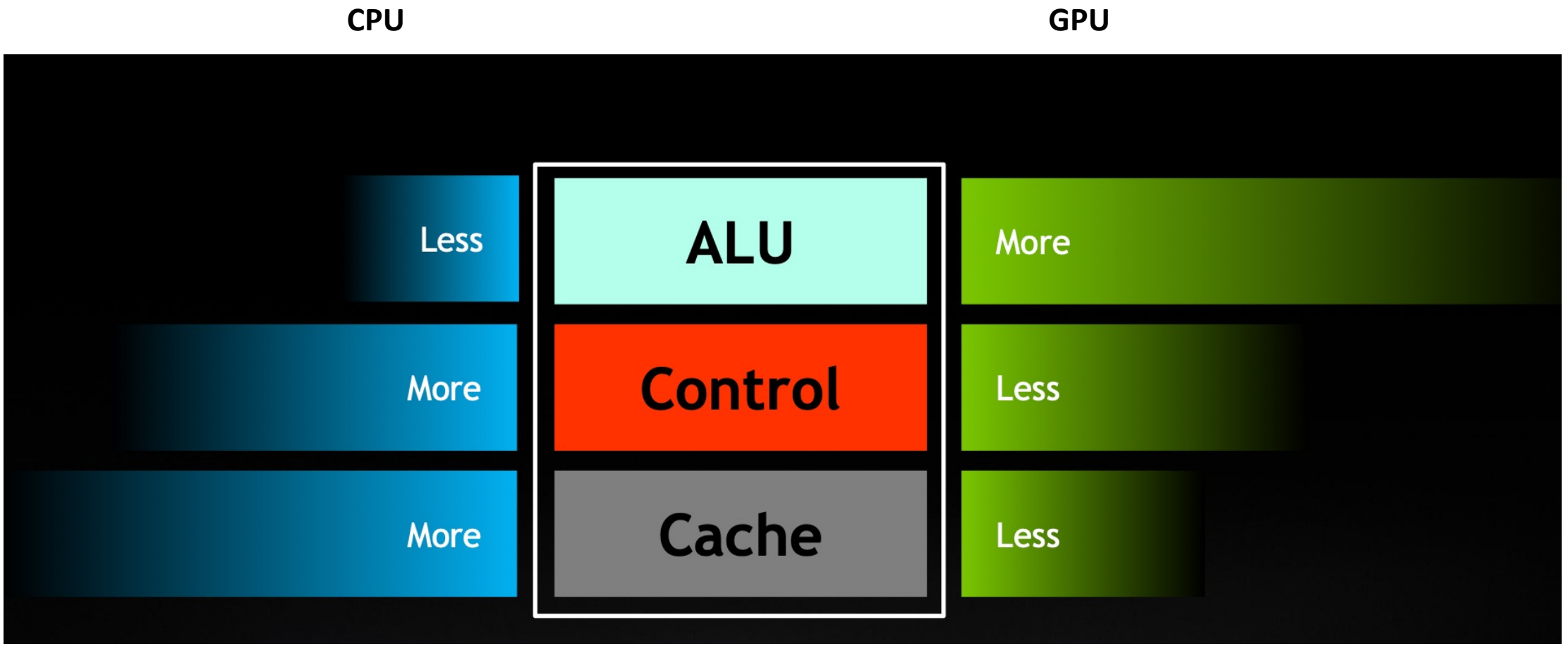
CPUs vs GPUs comparison



GPU follow SIMT pattern (Single Instruction multiple threads)

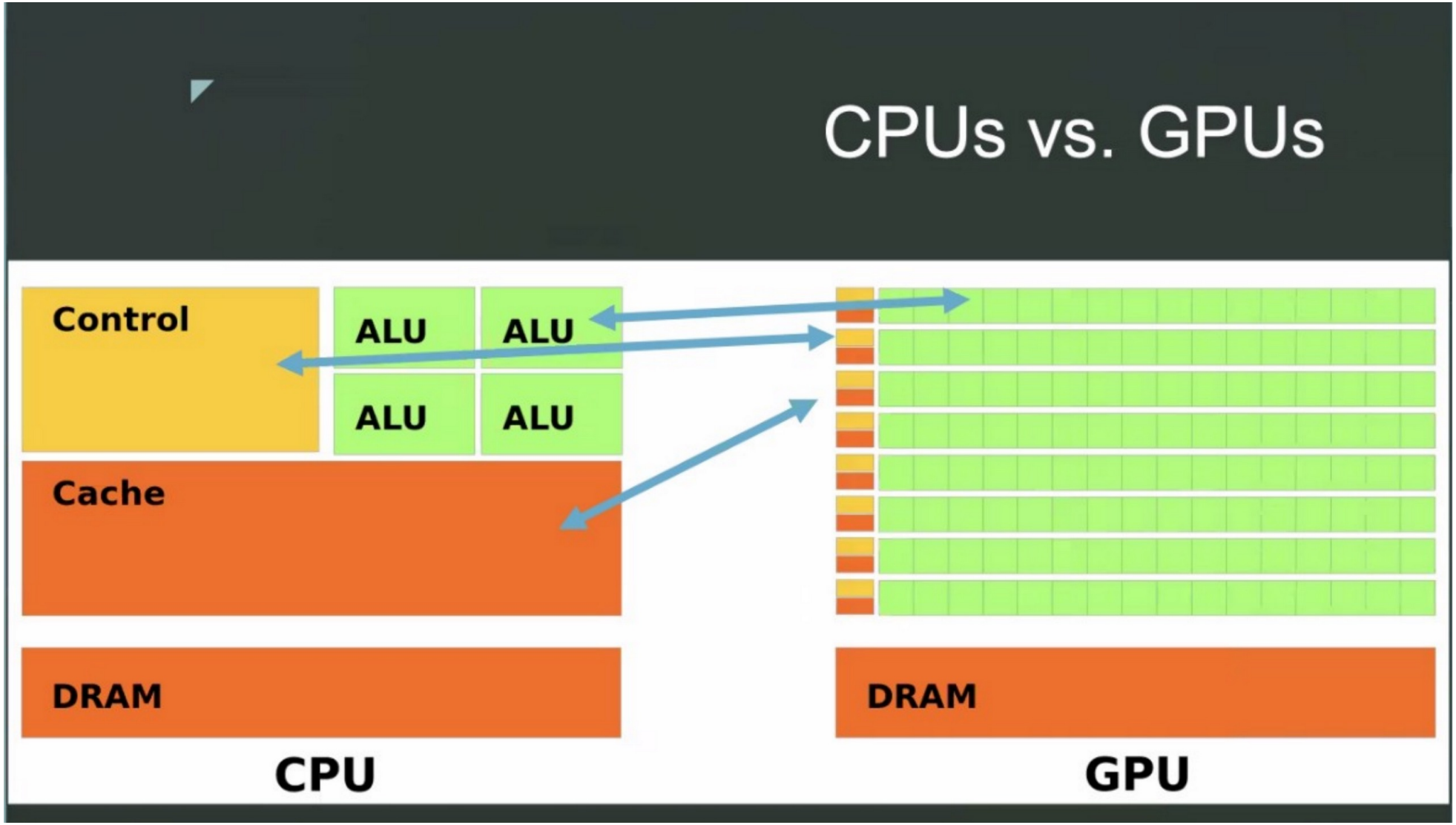
Source: Openhackaton @ Princeton 2023 "Introduction to GPU computing

CPUs vs GPUs comparison



Source: Openhackaton @ Princeton 2023 "Introduction to GPU computing

CPU vs GPU comparison



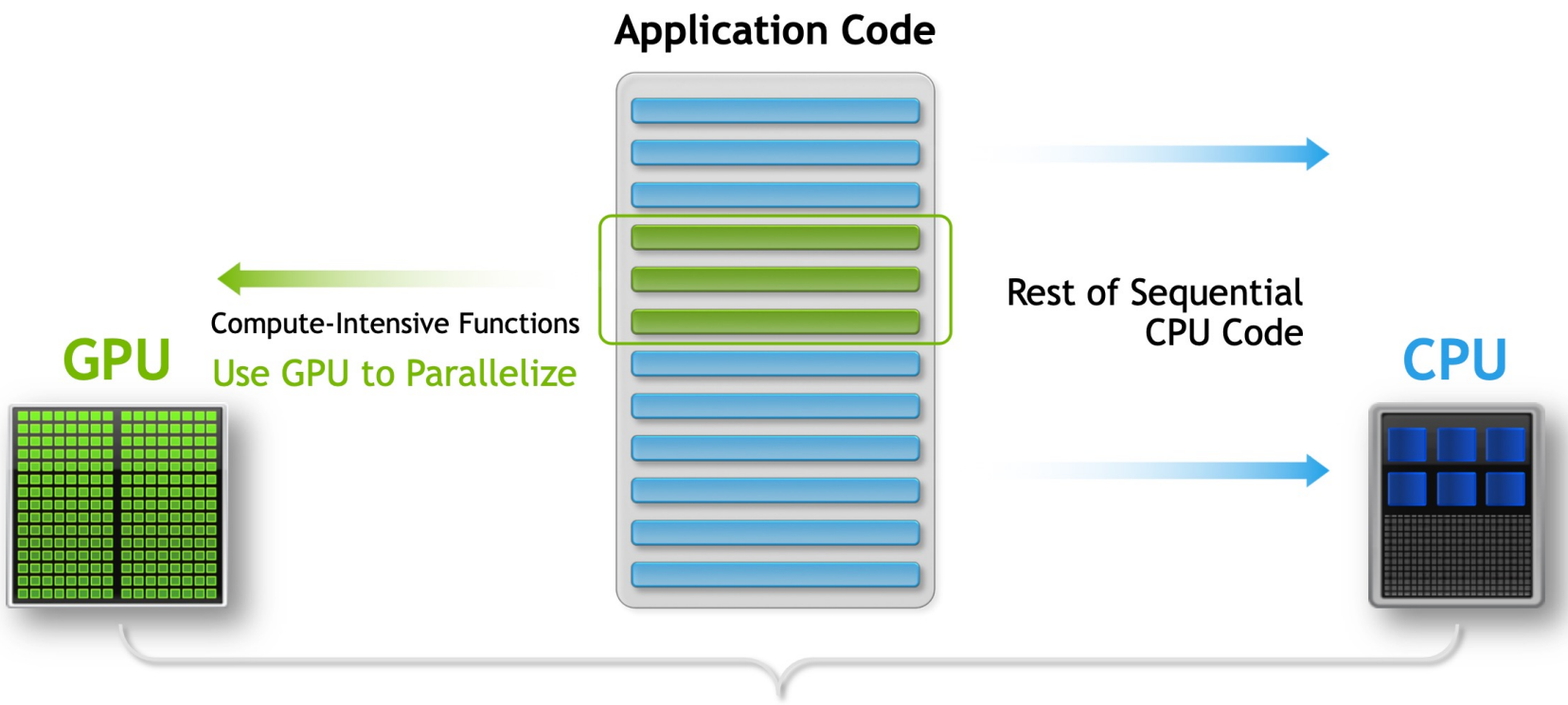
Source: Jayce Dowell Presentation on GPU University of New Mexico, Fall 2021

CPU vs GPU programming model

GPU: Matrix operations. (e.g. FFT)

CPU:

OS, security etc …

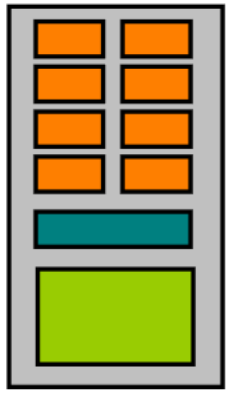


Source: https://www.olcf.ornl.gov/cuda-training-series/ ; CUDA C++

NVIDIA A100 specs

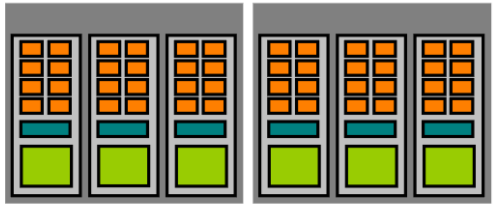
Scalar Processor

cuda core. **64 per SM**



Multiprocessor

streaming multiprocessor
: **108 SMs**

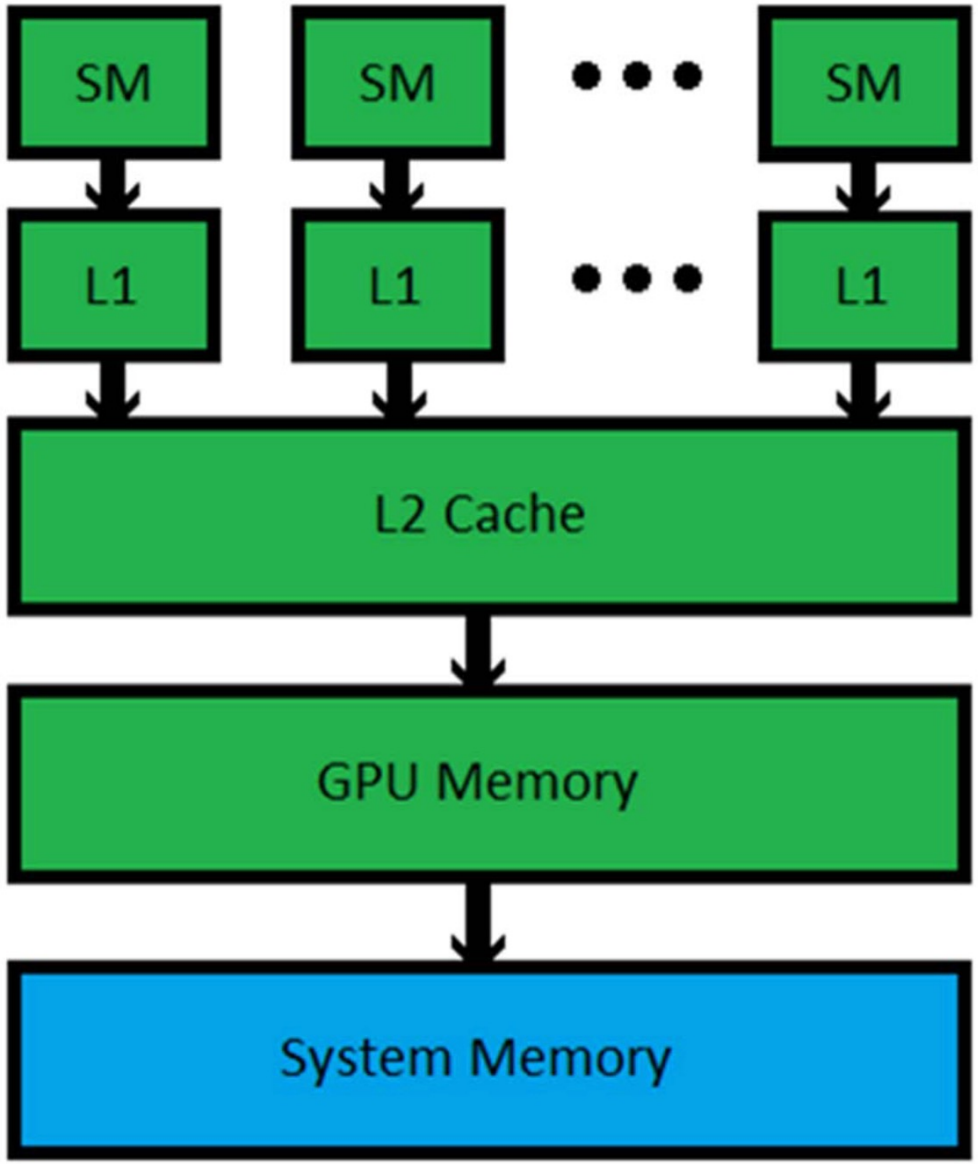


Device

108 SMs per device

Source: https://www.olcf.ornl.gov/wp-content/uploads/2019/12/03-CUDA-Fundamental-Optimization-Part-1.pdf

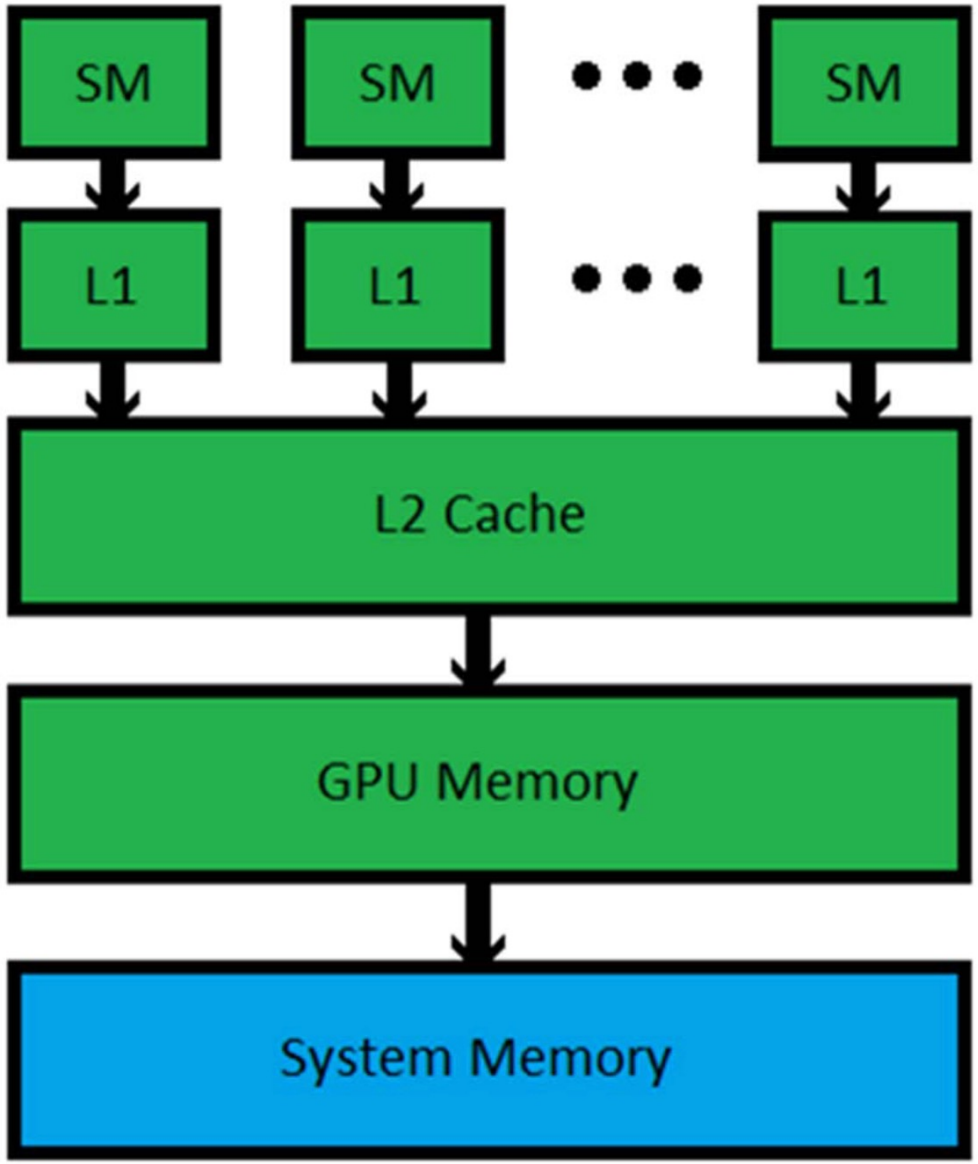
NVIDIA GPU memory hierarchy



- Registers and local memory (local memory is slower)

Source: http://meseec.ce.rit.edu/551-projects/spring2015/3-2.pdf; https://people.duke.edu/~ccc14/sta-663/CUDAPython.html

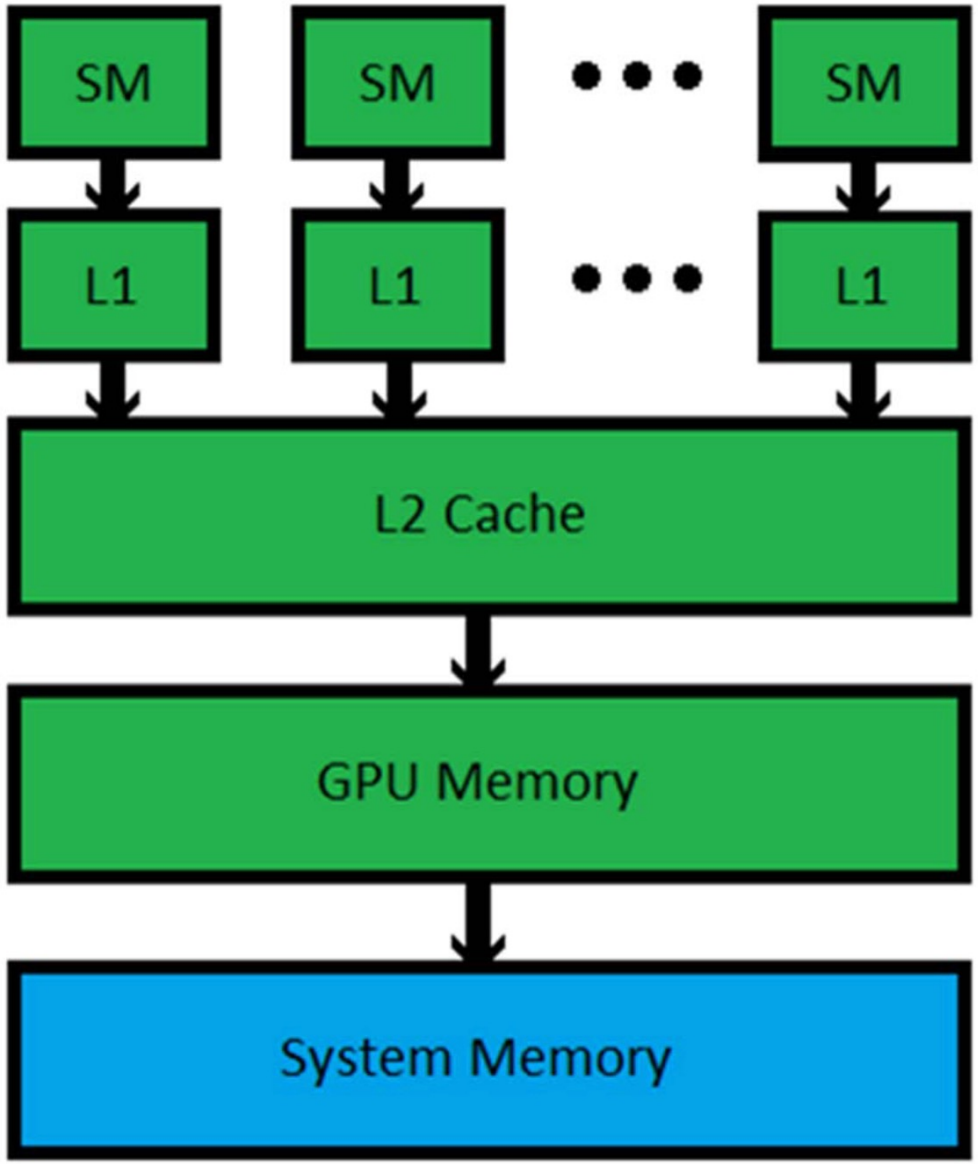
NVIDIA GPU memory hierarchy



- Registers and local memory (local memory is slower)
- Shared memory is accessible by all threads in the SM

Source: http://meseec.ce.rit.edu/551-projects/spring2015/3-2.pdf; https://people.duke.edu/~ccc14/sta-663/CUDAPython.html

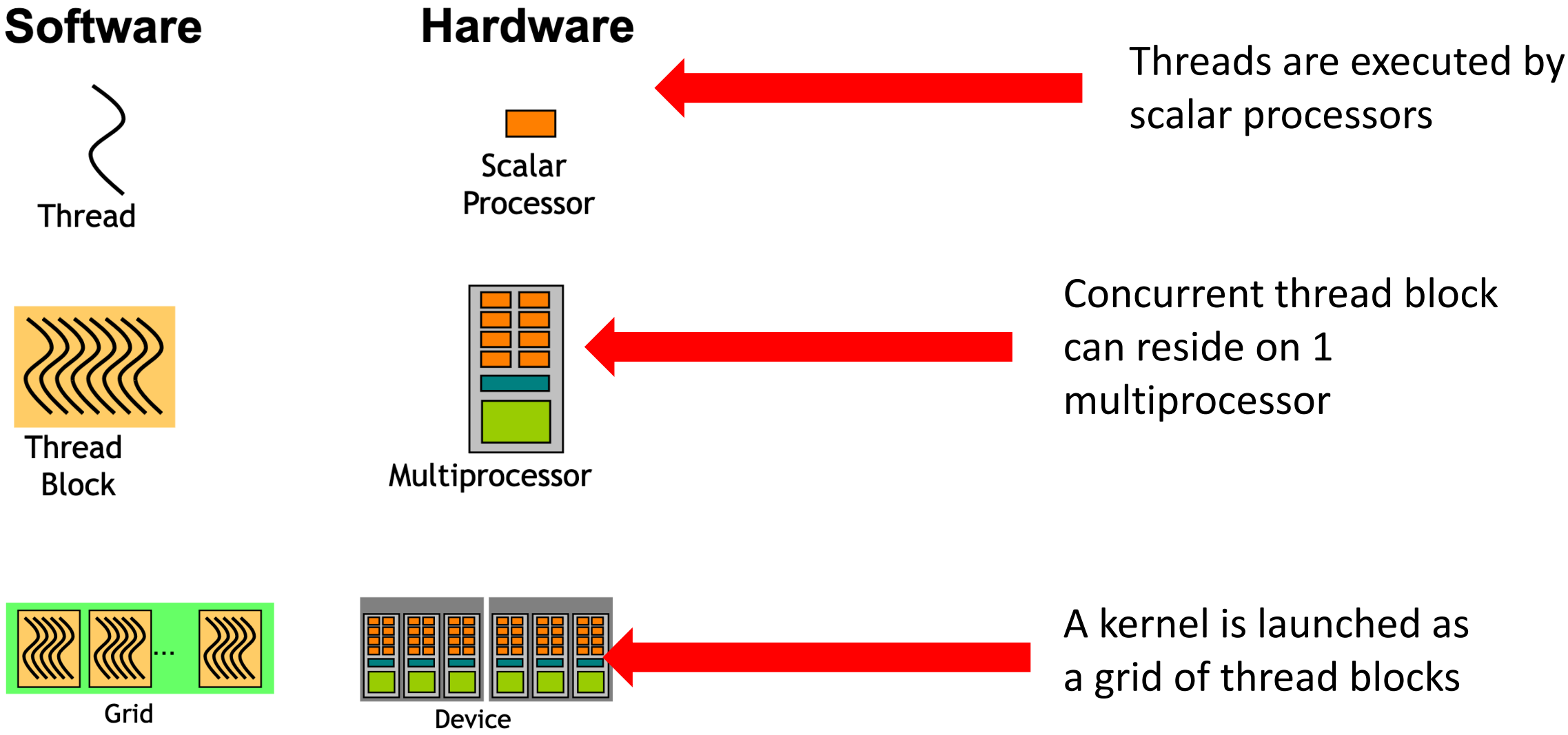
NVIDIA GPU memory hierarchy



- Registers and local memory (local memory is slower)
- Shared memory is accessible by all threads in the SM
- Global memory is accessible by all blocks and the CPU

Source: http://meseec.ce.rit.edu/551-projects/spring2015/3-2.pdf; https://people.duke.edu/~ccc14/sta-663/CUDAPython.html

GPU software programming



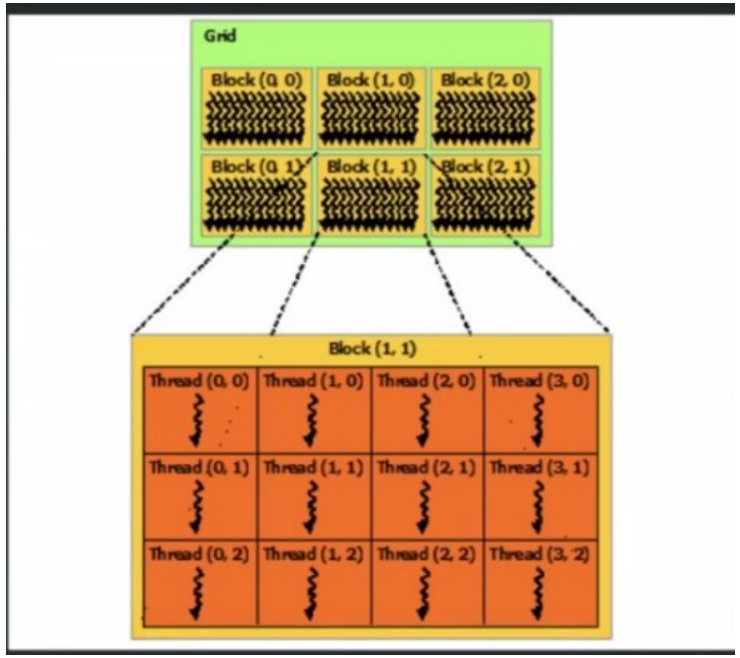
Source: https://www.olcf.ornl.gov/wp-content/uploads/2019/12/03-CUDA-Fundamental-Optimization-Part-1.pdf

CUDA terminology

* A kernel is the operation you want to run on your data which will be shipped to the GPU

* Kernels are launched in grid of blocks

* Grids and blocks can be 1D, 2D or 3D



Source: Jayce Dowell Presentation on GPU University of New Mexico, Fall 2021

II - Numba

What is numba?

- It is a just in time compiler (JIT) that translates python code into fast machine code.

What is numba?

- It is a just in time compiler (JIT) that translates python code into fast machine code.
- It means that it compiles the code during the execution of the program rather than before execution

What is numba?

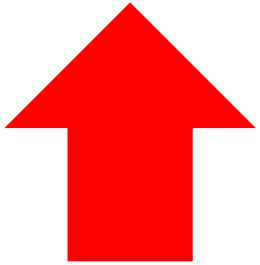
- It is a just in time compiler (JIT) that translates python code into fast machine code.
- It means that it compiles the code during the execution of the program rather than before execution
- It can be used on GPUs with cuda. It will allow to decorate functions that will be compiled into CUDA kernel

What is numba?

- It is a just in time compiler (JIT) that translates python code into fast machine code.
- It means that it compiles the code during the execution of the program rather than before execution
- It can be used on GPUs with cuda. It will allow to decorate functions that will be compiled into CUDA kernel
- Kernel written in numba have direct access to numpy arrays.

How to install it on Alpine?

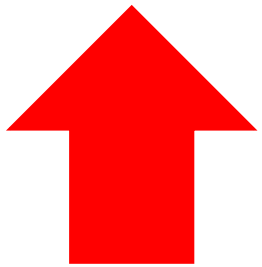
```
[kfotso@xsede.org@login-ci2 ~]$ acompile --ntasks=4
```



Requesting the resources to install

How to install it on Alpine?

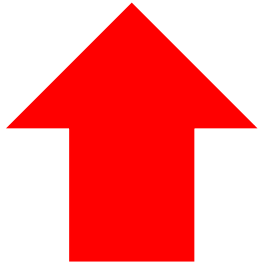
```
[kfotso@xsede.org@login-ci2 ~]$ acompile --ntasks=4

[kfotso@xsede.org@c3cpu-a2-u34-2 ~]$ ml anaconda
```



Load anaconda

How to install it on Alpine?

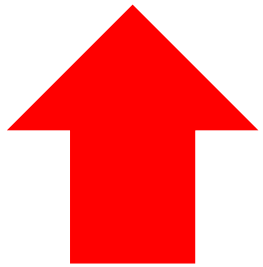
```
[kfotso@xsede.org@login-ci2 ~]$ acompile --ntasks=4

[kfotso@xsede.org@c3cpu-a2-u34-2 ~]$ ml anaconda

conda create --name numba_env python=3.10 numba cudatoolkit
```



Create the environment

How to install it on Alpine?

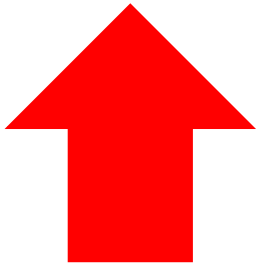
```
[kfotso@xsede.org@login-ci2 ~]$ acompile --ntasks=4

[kfotso@xsede.org@c3cpu-a2-u34-2 ~]$ ml anaconda

conda create --name numba_env python=3.10 numba cudatoolkit

exit
```



Release the resources once done

Next we verify the installation

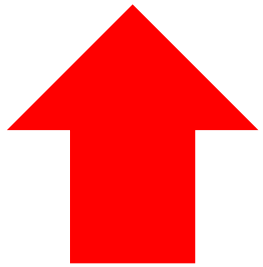
```
[kfotso@xs e.org@login-ci2 ~]$ sinteractive --partition=atesting_a100 --qos=testin
g --time=00:05:00 --gres=gpu:1 --ntasks=2
```



Request resource on debug gpu partition

Next we verify the installation

```
[kfotso@xsede.org@login-ci2 ~]$ sinteractive --partition=atesting_a100 --qos=testin
g --time=00:05:00 --gres=gpu:1 --ntasks=2

[kfotso@xsede.org@c3cpu-a2-u34-2 ~]$ ml anaconda

(base) [kfotso@xsede.org@c3cpu-a2-u34-2 ~]$ conda activate numba_env
```



Load anaconda and activate the ENV

Next we verify the installation

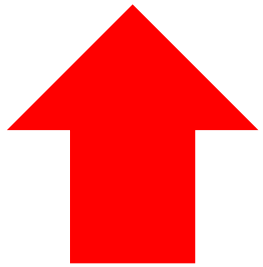
```
[kfotso@xsede.org@login-ci2 ~]$ sinteractive --partition=atesting_a100 --qos=testin
g --time=00:05:00 --gres=gpu:1 --ntasks=2

[kfotso@xsede.org@c3cpu-a2-u34-2 ~]$ ml anaconda

(base) [kfotso@xsede.org@c3cpu-a2-u34-2 ~]$ conda activate numba_env

(numba_env) [kfotso@xsede.org@c3gpu-a9-u31-1 ~]$ python
Python 3.10.13 (main, Sep 11 2023, 13:44:35) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
```

Call python

Next, we verify the installation

```
(numba_env) [kfotso@xsede.org@c3gpu-a9-u31-1 ~]$ python
Python 3.10.13 (main, Sep 11 2023, 13:44:35) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>> from numba import cuda
>>> cuda.is_available()
True
```



Verify cuda

GPU monitoring in general on Alpine

- To check the nodes where the job is running

```
de.org@login                                  72723]$ squeue --me
  JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
3481362     aa100 horovod_ kfotso@x  R       2:34      2 c3gpu-a9-u31-1,c
3479946  acompile acompile kfotso@x  R    2:03:48      1 c3cpu-c9-u5-2
3481778 atesting_ sinterac kfotso@x  R       7:56      1 c3gpu-a9-u29-1
de.org@login                                  72723]$ ssh c3gpu-a9-u29-1^C
```

We have a GPU node that we can ssh to: c3gpu-a9-u29-1

GPU monitoring in general on Alpine

- ssh to a that node on a different screen and run:

nvidia-smi -l

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.85.12    Driver Version: 525.85.12    CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA A100 80G...  On   | 00000000:25:00.0 Off |                    0 |
| N/A   33C    P0    80W / 300W |   1869MiB / 81920MiB |     26%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   1  NVIDIA A100 80G...  On   | 00000000:81:00.0 Off |                    0 |
| N/A   32C    P0    81W / 300W |   1869MiB / 81920MiB |     25%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A    569301      C   python                           1866MiB |
|    1   N/A  N/A    569302      C   python                           1866MiB |
+-----------------------------------------------------------------------------+
```

We can confirm that we are using 2 GPUs running in this example

GPU monitoring in general

- ssh to a that node on a different screen and run:

nvidia-smi -l

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.85.12    Driver Version: 525.85.12    CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA A100 80G...  On   | 00000000:25:00.0 Off |                    0 |
| N/A   33C    P0    80W / 300W |   1869MiB / 81920MiB |     26%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   1  NVIDIA A100 80G...  On   | 00000000:81:00.0 Off |                    0 |
| N/A   32C    P0    81W / 300W |   1869MiB / 81920MiB |     25%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A    569301      C   python                           1866MiB |
|    1   N/A  N/A    569302      C   python                           1866MiB |
+-----------------------------------------------------------------------------+
```

Percentage GPU usability. That is the portion of time 1 or more kernels were used

GPU monitoring

- ssh to a that node on a different screen and run:

nvidia-smi -l

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.85.12    Driver Version: 525.85.12    CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA A100 80G...  On   | 00000000:25:00.0 Off |                    0 |
| N/A   33C    P0    80W / 300W |   1869MiB / 81920MiB |     26%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   1  NVIDIA A100 80G...  On   | 00000000:81:00.0 Off |                    0 |
| N/A   32C    P0    81W / 300W |   1869MiB / 81920MiB |     25%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A    569301      C   python                           1866MiB |
|    1   N/A  N/A    569302      C   python                           1866MiB |
+-----------------------------------------------------------------------------+
```

How much memory is used.

GPU monitoring

- nvidia-smi has more parameters that can be used. Cheatsheet here: https://www.docs.arc.vt.edu/usage/gpumon.html

```
[kfotso@xsede.org@c3gpu-a9-u29-1 ~]$ nvidia-smi --query-gpu=timestamp,name,pci.bus_id,driver_version,temperatur
e.gpu,utilization.gpu,utilization.memory --format=csv -l 5
timestamp, name, pci.bus_id, driver_version, temperature.gpu, utilization.gpu [%], utilization.memory [%]
2023/10/23 21:42:45.228, NVIDIA A100 80GB PCIe, 00000000:25:00.0, 525.85.12, 32, 0 %, 0 %
2023/10/23 21:42:45.230, NVIDIA A100 80GB PCIe, 00000000:81:00.0, 525.85.12, 31, 0 %, 0 %
2023/10/23 21:42:50.233, NVIDIA A100 80GB PCIe, 00000000:25:00.0, 525.85.12, 33, 7 %, 0 %
2023/10/23 21:42:50.234, NVIDIA A100 80GB PCIe, 00000000:81:00.0, 525.85.12, 32, 5 %, 0 %
2023/10/23 21:42:55.236, NVIDIA A100 80GB PCIe, 00000000:25:00.0, 525.85.12, 34, 25 %, 2 %
2023/10/23 21:42:55.237, NVIDIA A100 80GB PCIe, 00000000:81:00.0, 525.85.12, 32, 23 %, 1 %
2023/10/23 21:43:00.238, NVIDIA A100 80GB PCIe, 00000000:25:00.0, 525.85.12, 34, 25 %, 2 %
2023/10/23 21:43:00.240, NVIDIA A100 80GB PCIe, 00000000:81:00.0, 525.85.12, 33, 25 %, 1 %
2023/10/23 21:43:05.241, NVIDIA A100 80GB PCIe, 00000000:25:00.0, 525.85.12, 34, 25 %, 2 %
2023/10/23 21:43:05.243, NVIDIA A100 80GB PCIe, 00000000:81:00.0, 525.85.12, 33, 24 %, 1 %
2023/10/23 21:43:10.244, NVIDIA A100 80GB PCIe, 00000000:25:00.0, 525.85.12, 35, 26 %, 2 %
2023/10/23 21:43:10.246, NVIDIA A100 80GB PCIe, 00000000:81:00.0, 525.85.12, 33, 25 %, 2 %
2023/10/23 21:43:15.247, NVIDIA A100 80GB PCIe, 00000000:25:00.0, 525.85.12, 35, 26 %, 2 %
2023/10/23 21:43:15.248, NVIDIA A100 80GB PCIe, 00000000:81:00.0, 525.85.12, 33, 26 %, 2 %
```

Numba for universal functions

What are universal functions (ufuncs) ?

- They operate on ndarrays in an element by element fashion.

Source: https://thedatafrog.com/en/articles/boost-python-gpu/; https://numpy.org/doc/stable/reference/ufuncs.html

What are universal functions (ufuncs) ?

- They operate on ndarrays in an element by element fashion.

- Written in compiled C code therefore much faster than native Python

Source: https://thedatafrog.com/en/articles/boost-python-gpu/; https://numpy.org/doc/stable/reference/ufuncs.html

Let's see an example of a ufunc for a log function

```
import numpy as np
import math
from numba import vectorize
import time

# Code to demonstrate basic vectorization on GPU
# Source: https://thedatafrog.com/en/articles/boost-python-gpu/

@vectorize(['float32(float32)'], target='cuda')
def log_calc_gpu(x):
    return math.log(x)


# We define the vector when we will perform the log
total_points = int(1e8)
array = np.arange(1, total_points, dtype=np.float32)

# Calling the function and timing it
start = time.time() #Start
log_calc_gpu(array)
end = time.time() # End

# Print duration
print("Log calculation took {0} seconds)".format(end-start))
```

Vectorize allows to work on an array elementwise in parallel, which means using multiple threads

Source: https://thedatafrog.com/en/articles/boost-python-gpu/; https://numpy.org/doc/stable/reference/ufuncs.html

Let's see an example of a ufunc for a log function

```
import numpy as np
import math
from numba import vectorize
import time

# Code to demonstrate basic vectorization on GPU
# Source: https://thedatafrog.com/en/articles/boost-python-gpu/

@vectorize(['float32(float32)'], target='cuda')
def log_calc_gpu(x):
    return math.log(x)


# We define the vector when we will perform the log
total_points = int(1e8)
array = np.arange(1, total_points, dtype=np.float32)

# Calling the function and timing it
start = time.time() #Start
log_calc_gpu(array)
end = time.time() # End

# Print duration
print("Log calculation took {0} seconds)".format(end-start))
```

We tell python that we want it to be processed on the GPU

Source: https://thedatafrog.com/en/articles/boost-python-gpu/; https://numpy.org/doc/stable/reference/ufuncs.html

Let's see an example of a ufunc for a log function

```
import numpy as np
import math
from numba import vectorize
import time

# Code to demonstrate basic vectorization on GPU
# Source: https://thedatafrog.com/en/articles/boost-python-gpu/

@vectorize(['float32(float32)'], target='cuda')
def log_calc_gpu(x):
    return math.log(x)


# We define the vector when we will perform the log
total_points = int(1e8)
array = np.arange(1, total_points, dtype=np.float32)

# Calling the function and timing it
start = time.time() #Start
log_calc_gpu(array)
end = time.time() # End

# Print duration
print("Log calculation took {0} seconds)".format(end-start))
```

We define the return type as float32 and the input type as float32. 'return_type(input_type)'

Source: https://thedatafrog.com/en/articles/boost-python-gpu/; https://numpy.org/doc/stable/reference/ufuncs.html

Let's see an example of a ufunc for a log function

```
import numpy as np
import math
from numba import vectorize
import time

# Code to demonstrate basic vectorization on GPU
# Source: https://thedatafrog.com/en/articles/boost-python-gpu/

@vectorize(['float32(float32)'], target='cuda')
def log_calc_gpu(x):
    return math.log(x)


# We define the vector when we will perform the log
total_points = int(1e8)
array = np.arange(1, total_points, dtype=np.float32)

# Calling the function and timing it
start = time.time() #Start
log_calc_gpu(array)
end = time.time() # End

# Print duration
print("Log calculation took {0} seconds)".format(end-start))
```

Calling the vecotorize function

Source: https://thedatafrog.com/en/articles/boost-python-gpu/; https://numpy.org/doc/stable/reference/ufuncs.html

Log function assessment

```
(numba_env) [kfotso@xsede.org@c3gpu-a9-u31-1 other_example]$ python log_calc.py
Log calculation took 0.24405527114868164 seconds)
(numba_env) [kfotso@xsede.org@c3gpu-a9-u31-1 other_example]$ python log_calc_gpu.py
Log calculation took 0.17553949356079102 seconds)
```

We can see that the GPU was slightly faster but what happen if we reduce the number of points to 10 millions?

Log function assessment

```
(numba_env) [kfotso@xsede.org@c3gpu-a9-u31-1 other_example]$ python log_calc_gpu.py
Log calculation took 0.07612371444702148 seconds)
(numba_env) [kfotso@xsede.org@c3gpu-a9-u31-1 other_example]$ python log_calc.py
Log calculation took 0.024480581283569336 seconds)
```

The CPU was faster when my vector was 10 millions data points

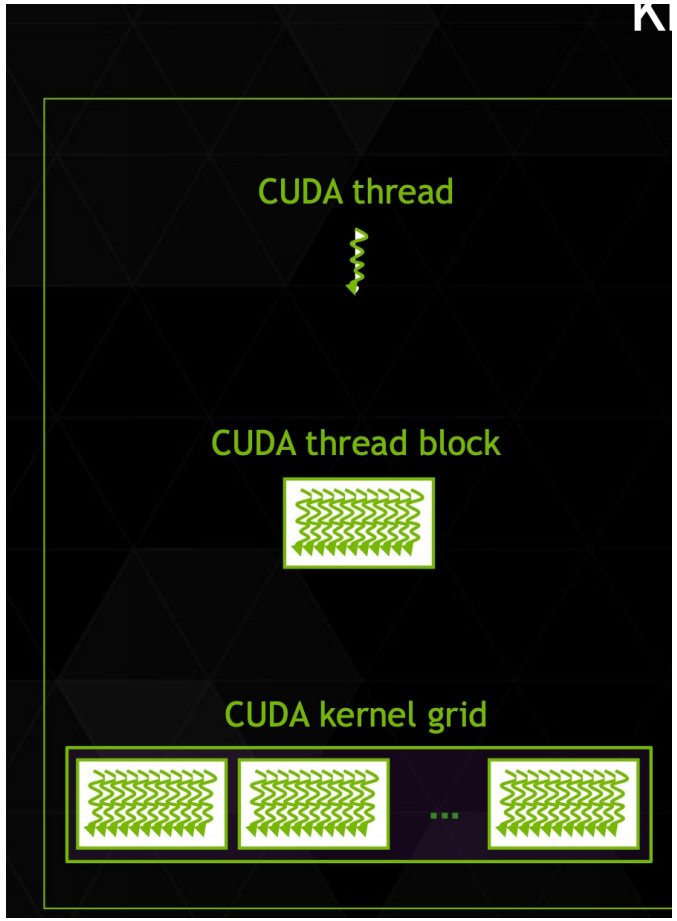
Log function assessment

- The array size needs to be sufficiently large to take advantage of the GPU.

- Otherwise most of the time will be wasted shipping the data to the GPU and then then results back to the host.

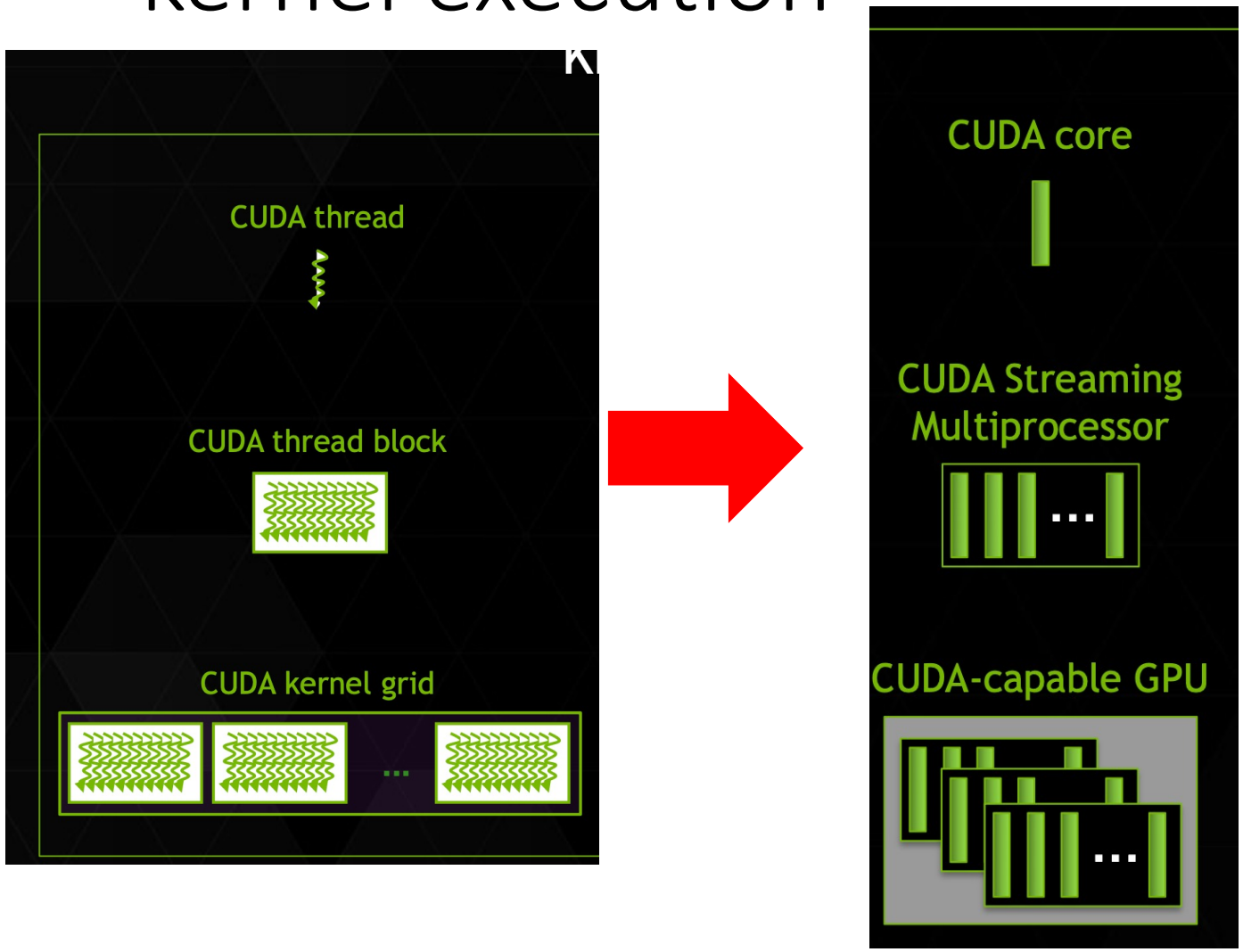
Source: https://thedatafrog.com/en/articles/boost-python-gpu/; https://numpy.org/doc/stable/reference/ufuncs.html

Let’s learn how to write a cuda kernel now: kernel execution



Source: Openhackaton Python numba Princeton hackaton

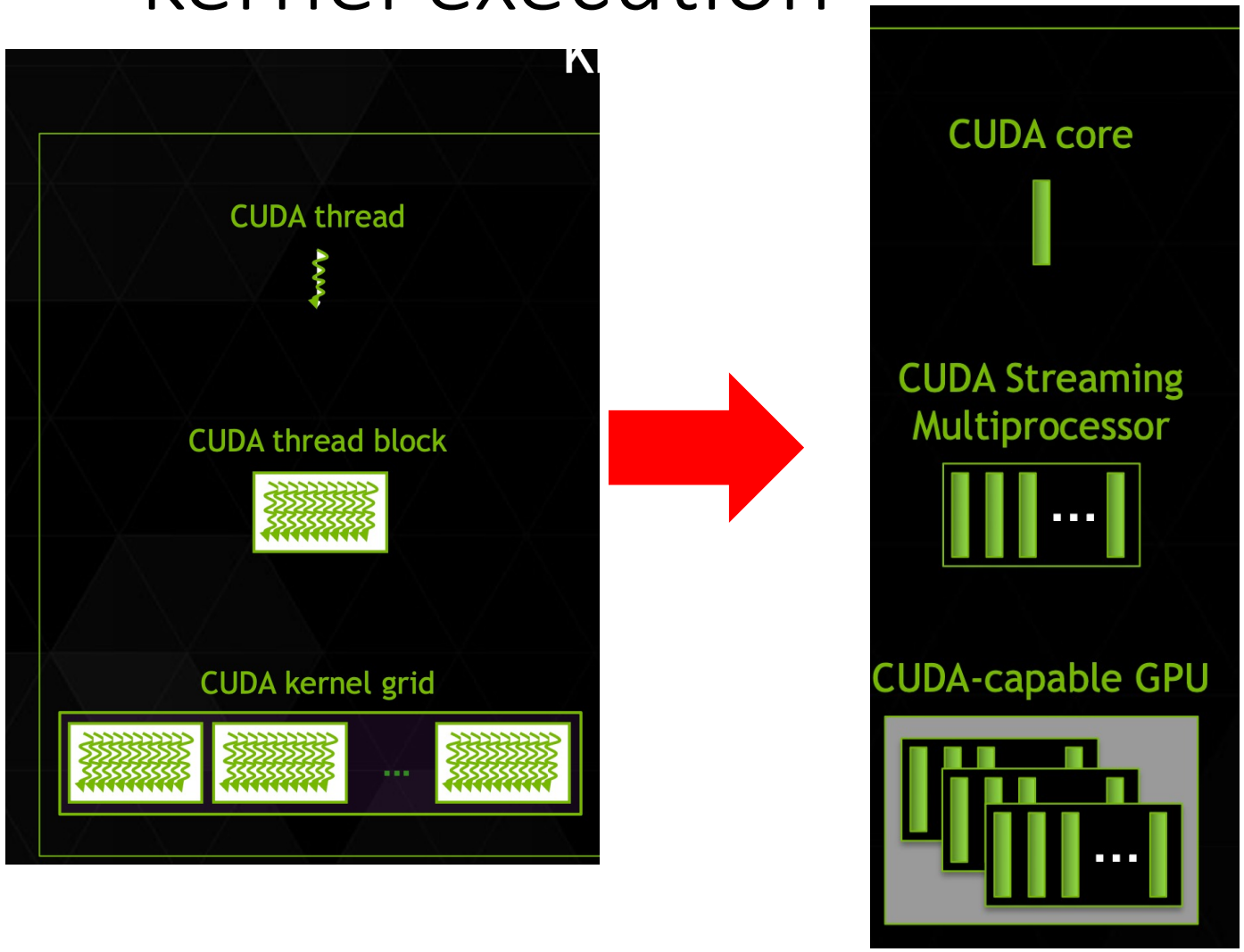
Let’s learn how to write a cuda kernel now: kernel execution



Source: Openhackaton Python numba Princeton hackaton

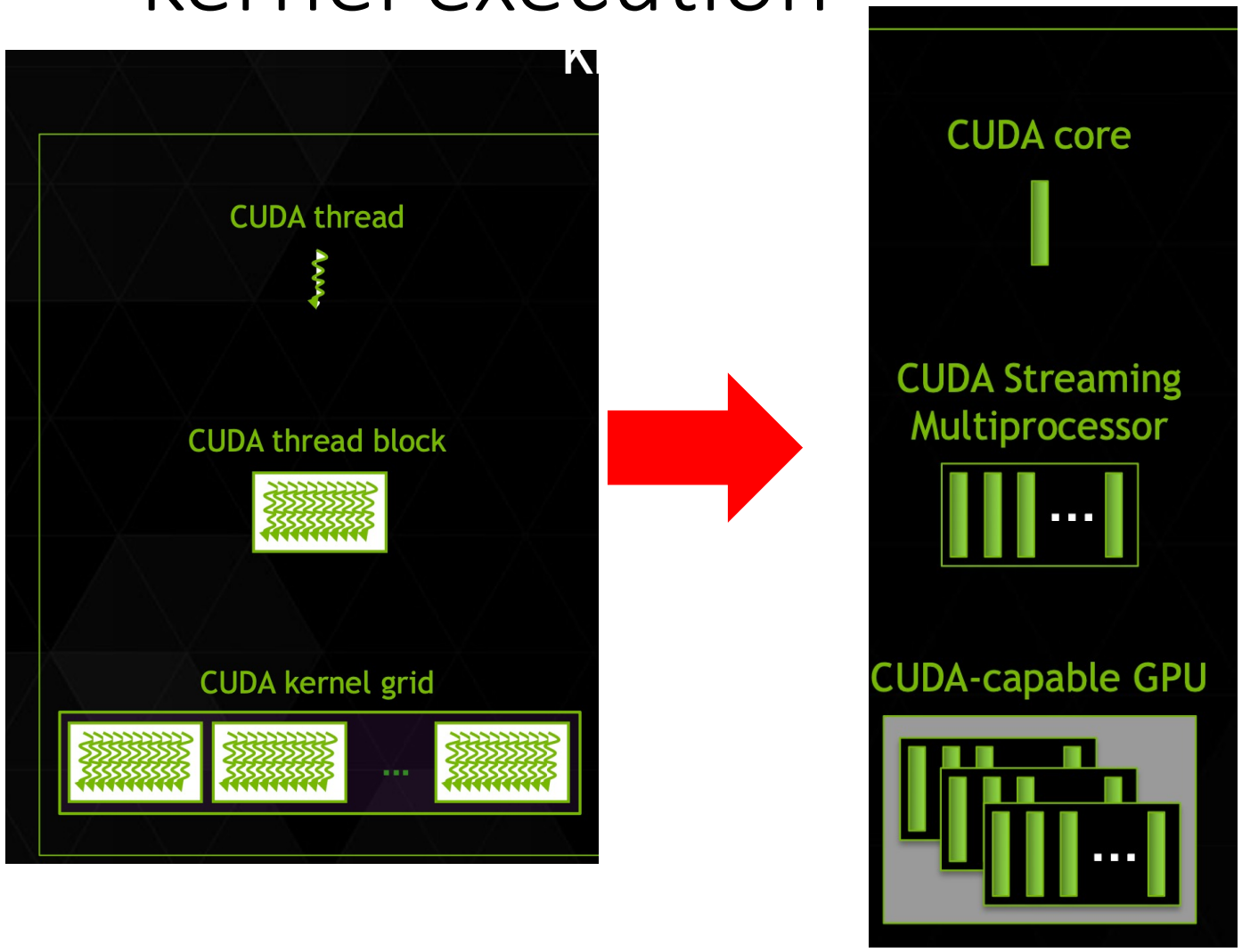
Let’s learn how to write a cuda kernel now: kernel execution

* Each thread is executed by a core



Source: Openhackaton Python numba Princeton hackaton

Let’s learn how to write a cuda kernel now: kernel execution

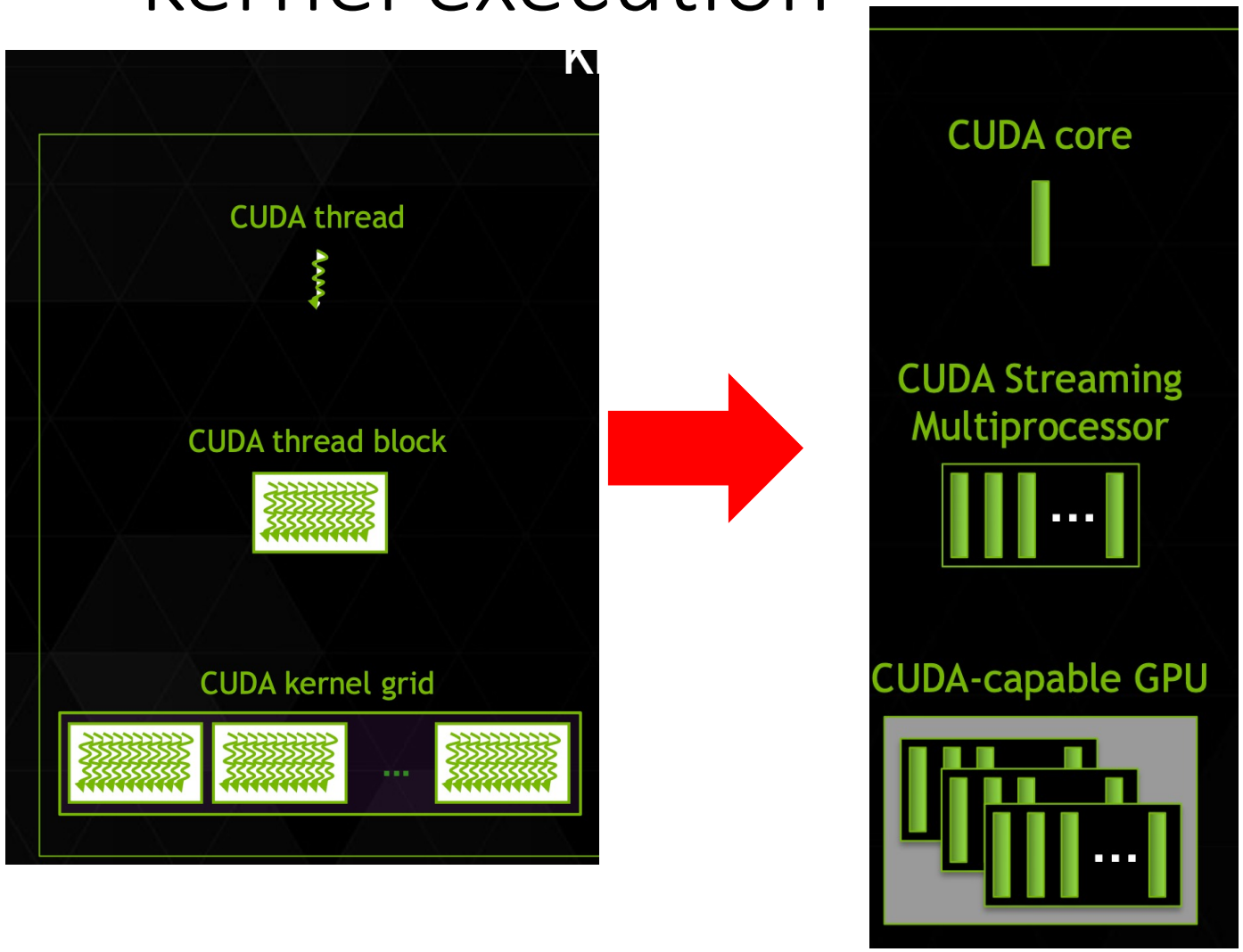


* Each thread is executed by a core

* A block is executed by an SM

Source: Openhackaton Python numba Princeton hackaton

Let’s learn how to write a cuda kernel now: kernel execution



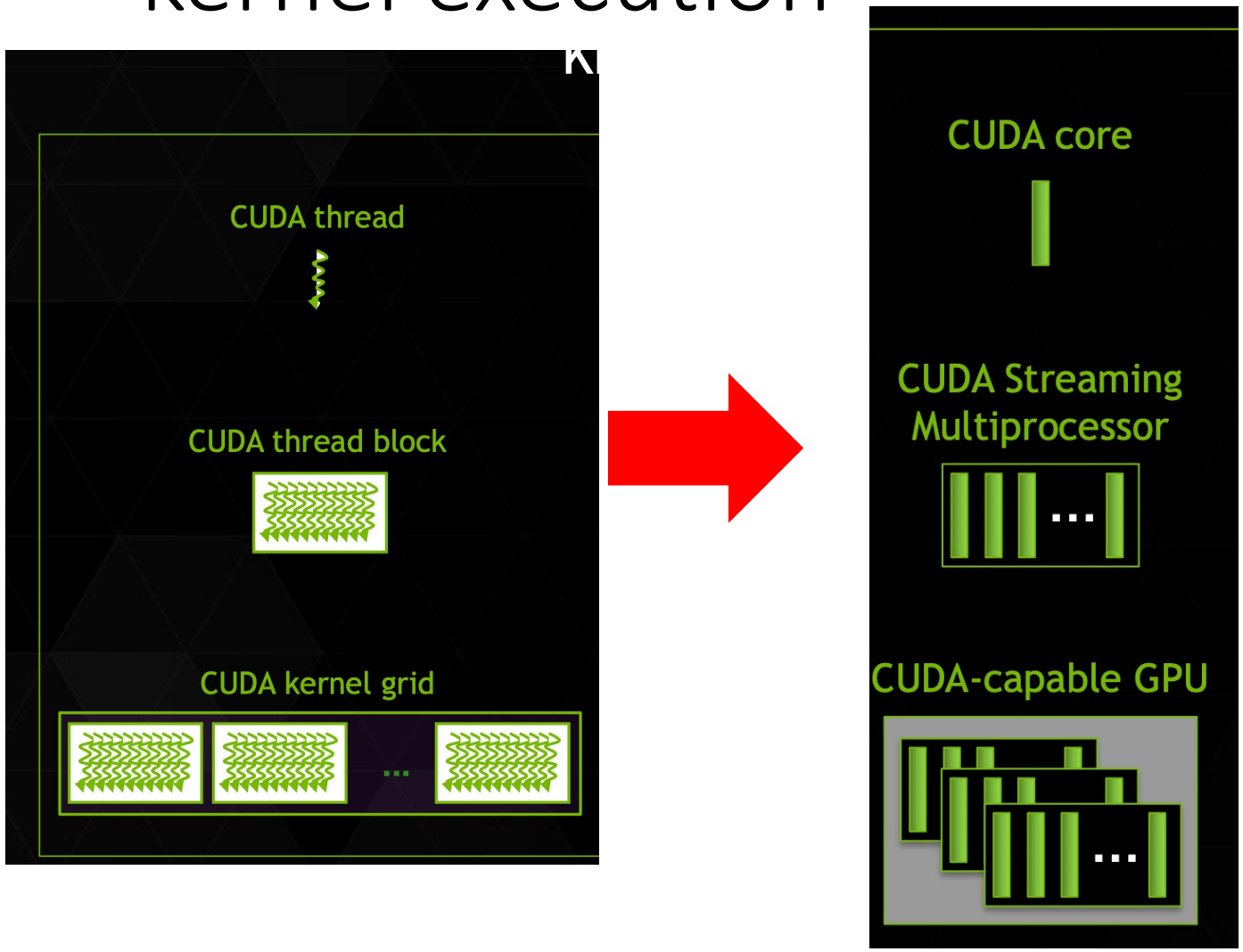
* Each thread is executed by a core

* A block is executed by an SM

* Several concurrent block can reside on 1 SM depending on the block. Trade off between block memory requirement and available resources

Source: Openhackaton Python numba Princeton hackaton

Let’s learn how to write a cuda kernel now: kernel execution



* Each thread is executed by a core

* A block is executed by an SM

* Several concurrent block can reside on 1 SM depending on the block. Trade off between block memory requirement and available resources

* Each kernel is executed on a device.

* Multiple kernels can execute on a device at a time

Source: Openhackaton Python numba Princeton hackaton 2023

Communication

- Thread may need to cooperate for memory accesses

Source: Openhackaton Python numba Princeton hackaton 2023

Communication

- Thread may need to cooperate for memory accesses
- Or in order to share results they will need to cooperate using the shared memory (L1)

Source: Openhackaton Python numba Princeton hackaton 2023

Communication

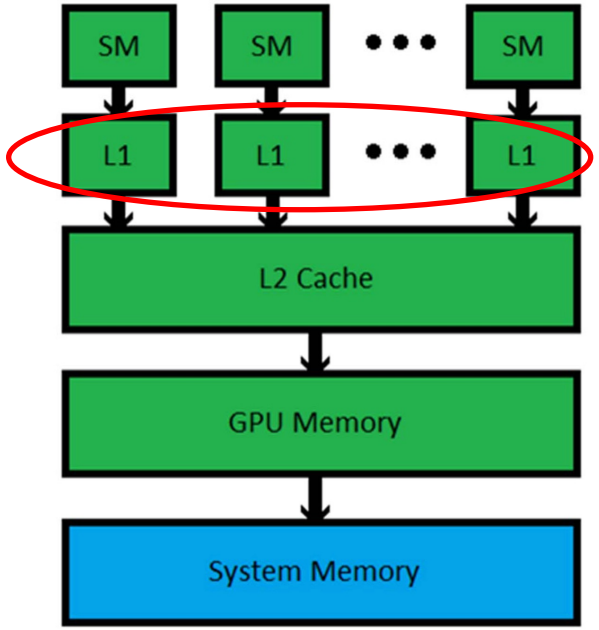
- Thread may need to cooperate for memory accesses
- Or in order to share results they will need to cooperate using the shared memory (L1)

- It is accessible by all threads within the block

Source: Openhackaton Python numba Princeton hackaton 2023

Communication

- Thread may need to cooperate for memory accesses
- Or in order to share results they will need to cooperate using the shared memory (L1)
- It is accessible by all threads within the block

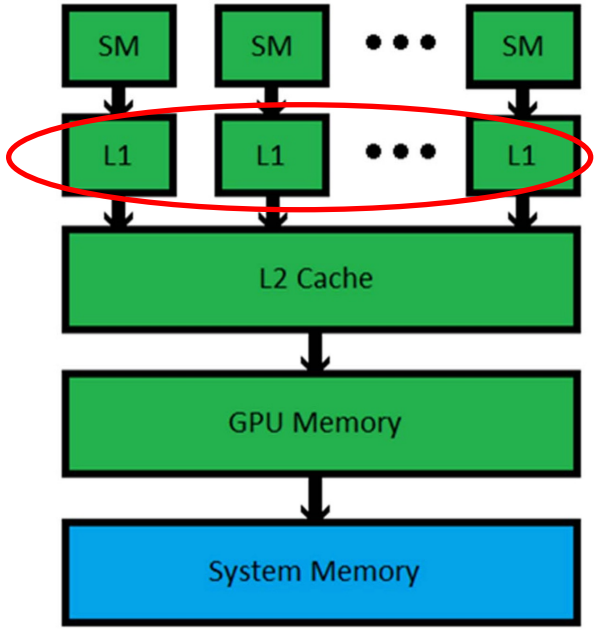


Source: Openhackaton Python numba Princeton hackaton 2023

Communication

- Thread may need to cooperate for memory accesses
- Or in order to share results they will need to cooperate using the shared memory (L1)

- It is accessible by all threads within the block
- In Numba deal with the global, shared and local memory.



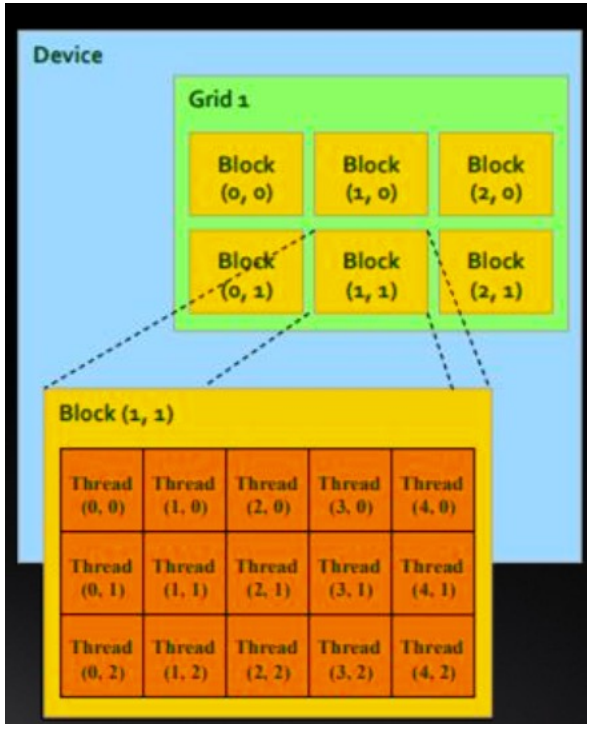
Source: Openhackaton Python numba Princeton hackaton 2023

Communication

- Thread may need to cooperate for memory accesses
- Or in order to share results they will need to cooperate using the shared memory (L1)

- It is accessible by all threads within the block

- Each thread and block have a threadID

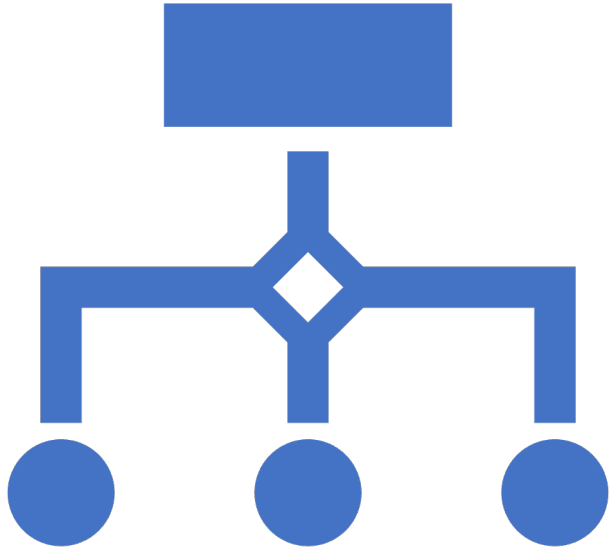


Source: Openhackaton Python numba Princeton hackaton 2023

CUDA kernel overview

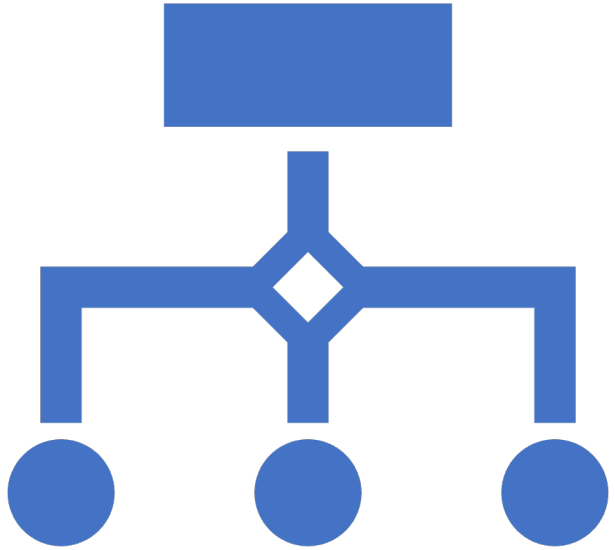
What is a kernel?

- A GPU function invoked from the GPU that specifies the grid parameters.
- Grid parameters are block per thread and thread per block.



What is a kernel?

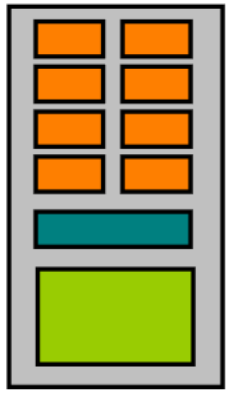
- A GPU function invoked from the GPU that specifies the grid parameters.
- Grid parameters are block per thread and thread per block.
- One the kernel has been called the kernel function code is executed by every thread once



NVIDIA A100 specs

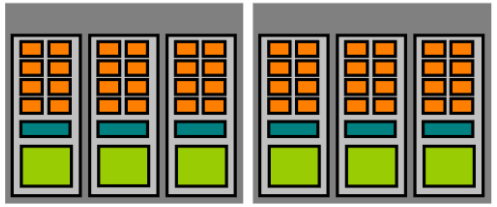
Scalar Processor

cuda core. **64 per SM**



Multiprocessor

streaming multiprocessor
: **108 SMs**



Device

108 SMs per device

Source: https://www.olcf.ornl.gov/wp-content/uploads/2019/12/03-CUDA-Fundamental-Optimization-Part-1.pdf

Cuda kernel

Make sure to import cuda

```
import numba.cuda as cuda
import numpy as np

# Source:  Openhackaton Python numba Princeton hackaton 2023

N = 500000
threadsperblock = 128

@cuda.jit('void(float32[:], float32[:], float32[:])')
def cu_substraction(array_A,array_B,result):

    """ Here, we define the kernel """

    bx = cuda.blockIdx.x
    bw = cuda.blockDim.x
    tx = cuda.threadIdx.x

    tid = tx + bx * bw

    if tid < N:
        result[tid] = array_A[tid] - array_B[tid]


# Defining the array.
array_A = np.arange(N, dtype = np.float32)
array_B = np.arange(N, dtype = np.float32)
result = np.zeros(N, dtype = np.float32)

blockpergrid = N + (threadsperblock - 1) // threadsperblock


cu_substraction[blockpergrid, threadsperblock](array_A, array_B, result)
```

Cuda kernel

```
import numba.cuda as cuda
import numpy as np

# Source:  Openhackaton Python numba Princeton hackaton 2023

N = 500000
threadsperblock = 128

@cuda.jit('void(float32[:], float32[:], float32[:])')
def cu_substraction(array_A,array_B,result):

    """ Here, we define the kernel """

    bx = cuda.blockIdx.x
    bw = cuda.blockDim.x
    tx = cuda.threadIdx.x

    tid = tx + bx * bw

    if tid < N:
        result[tid] = array_A[tid] - array_B[tid]


# Defining the array.
array_A = np.arange(N, dtype = np.float32)
array_B = np.arange(N, dtype = np.float32)
result = np.zeros(N, dtype = np.float32)

blockpergrid = N + (threadsperblock - 1) // threadsperblock


cu_substraction[blockpergrid, threadsperblock](array_A, array_B, result)
```

Size of my array

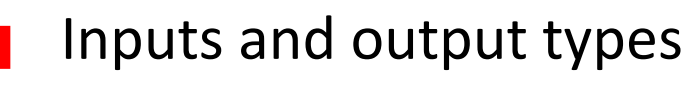
Cuda kernel

```python
import numba.cuda as cuda
import numpy as np

# Source:  Openhackaton Python numba Princeton hackaton 2023

N = 500000
threadsperblock = 128

@cuda.jit('void(float32[:], float32[:], float32[:])')
def cu_substraction(array_A,array_B,result):

    """ Here, we define the kernel """

    bx = cuda.blockIdx.x
    bw = cuda.blockDim.x
    tx = cuda.threadIdx.x

    tid = tx + bx * bw

    if tid < N:
        result[tid] = array_A[tid] - array_B[tid]


# Defining the array.
array_A = np.arange(N, dtype = np.float32)
array_B = np.arange(N, dtype = np.float32)
result = np.zeros(N, dtype = np.float32)

blockpergrid = N + (threadsperblock - 1) // threadsperblock


cu_substraction[blockpergrid, threadsperblock](array_A, array_B, result)
```

I define the number of threads per block, here based on the architecture for simplicity

Cuda kernel

```python
import numba.cuda as cuda
import numpy as np

# Source:  Openhackaton Python numba Princeton hackaton 2023

N = 500000
threadsperblock = 128

@cuda.jit('void(float32[:], float32[:], float
def cu_substraction(array_A,array_B,result):

    """ Here, we define the kernel """

    bx = cuda.blockIdx.x
    bw = cuda.blockDim.x
    tx = cuda.threadIdx.x

    tid = tx + bx * bw

    if tid < N:
        result[tid] = array_A[tid] - array_B[tid]


# Defining the array.
array_A = np.arange(N, dtype = np.float32)
array_B = np.arange(N, dtype = np.float32)
result = np.zeros(N, dtype = np.float32)

blockpergrid = N + (threadsperblock - 1) // threadsperblock


cu_substraction[blockpergrid, threadsperblock](array_A, array_B, result)
```



Inputs and output types

Cuda kernel

```python
import numba.cuda as cuda
import numpy as np

# Source:  Openhackaton Python numba Princeton hackaton 2023

N = 500000
threadsperblock = 128

@cuda.jit('void(float32[:], float32[:], float32[:])')
def cu_substraction(array_A,array_B,result):

    """ Here, we define the kernel """

    bx = cuda.blockIdx.x
    bw = cuda.blockDim.x
    tx = cuda.threadIdx.x

    tid = tx + bx * bw

    if tid < N:
        result[tid] = array_A[tid] - array_B[tid]


# Defining the array.
array_A = np.arange(N, dtype = np.float32)
array_B = np.arange(N, dtype = np.float32)
result = np.zeros(N, dtype = np.float32)

blockpergrid = N + (threadsperblock - 1) // threadsperblock


cu_substraction[blockpergrid, threadsperblock](array_A, array_B, result)
```

bx is the block index within the grid

Cuda kernel

```python
import numba.cuda as cuda
import numpy as np

# Source:  Openhackaton Python numba Princeton hackaton 2023

N = 500000
threadsperblock = 128

@cuda.jit('void(float32[:], float32[:], float32[:])')
def cu_substraction(array_A,array_B,result):

    """ Here, we define the kernel """

    bx = cuda.blockIdx.x
    bw = cuda.blockDim.x
    tx = cuda.threadIdx.x

    tid = tx + bx * bw

    if tid < N:
        result[tid] = array_A[tid] - array_B[tid]


# Defining the array.
array_A = np.arange(N, dtype = np.float32)
array_B = np.arange(N, dtype = np.float32)
result = np.zeros(N, dtype = np.float32)

blockpergrid = N + (threadsperblock - 1) // threadsperblock


cu_substraction[blockpergrid, threadsperblock](array_A, array_B, result)
```

Dimension of the block.
blockDim.x is a 1D block.

A 2D block would have been:
bx = blockDim.x,
by = blockDim.y

Cuda kernel

```python
import numba.cuda as cuda
import numpy as np

# Source:  Openhackaton Python numba Princeton hackaton 2023

N = 500000
threadsperblock = 128

@cuda.jit('void(float32[:], float32[:], float32[:])')
def cu_substraction(array_A,array_B,result):

    """ Here, we define the kernel """

    bx = cuda.blockIdx.x
    bw = cuda.blockDim.x
    tx = cuda.threadIdx.x

    tid = tx + bx * bw

    if tid < N:
        result[tid] = array_A[tid] - array_B[tid]


# Defining the array.
array_A = np.arange(N, dtype = np.float32)
array_B = np.arange(N, dtype = np.float32)
result = np.zeros(N, dtype = np.float32)

blockpergrid = N + (threadsperblock - 1) // threadsperblock


cu_substraction[blockpergrid, threadsperblock](array_A, array_B, result)
```

tx is the threadIDx within the block

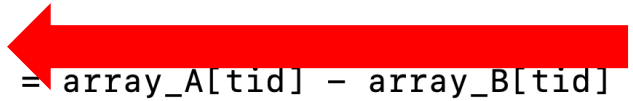
Cuda kernel

```python
import numba.cuda as cuda
import numpy as np

# Source:  Openhackaton Python numba Princeton hackaton 2023

N = 500000
threadsperblock = 128

@cuda.jit('void(float32[:], float32[:], float32[:])')
def cu_substraction(array_A,array_B,result):

    """ Here, we define the kernel """

    bx = cuda.blockIdx.x
    bw = cuda.blockDim.x
    tx = cuda.threadIdx.x

    tid = tx + bx * bw

    if tid < N:
        result[tid] = array_A[tid] - array_B[tid]


# Defining the array.
array_A = np.arange(N, dtype = np.float32)
array_B = np.arange(N, dtype = np.float32)
result = np.zeros(N, dtype = np.float32)

blockpergrid = N + (threadsperblock - 1) // threadsperblock


cu_substraction[blockpergrid, threadsperblock](array_A, array_B, result)
```

Thus, the formula to calculate the global thread ID is the product of the number of blocks by threads per block + the local threadID

Cuda kernel

```python
import numba.cuda as cuda
import numpy as np

# Source:  Openhackaton Python numba Princeton hackaton 2023

N = 500000
threadsperblock = 128

@cuda.jit('void(float32[:], float32[:], float32[:])')
def cu_substraction(array_A,array_B,result):

    """ Here, we define the kernel """

    bx = cuda.blockIdx.x
    bw = cuda.blockDim.x
    tx = cuda.threadIdx.x

    tid = tx + bx * bw

    if tid < N:
        result[tid] = array_A[tid] - array_B[tid]


# Defining the array.
array_A = np.arange(N, dtype = np.float32)
array_B = np.arange(N, dtype = np.float32)
result = np.zeros(N, dtype = np.float32)

blockpergrid = N + (threadsperblock - 1) // threadsperblock


cu_substraction[blockpergrid, threadsperblock](array_A, array_B, result)
```



It is possible that block size and grid size are not divisor of the array size, hence why by precaution we check the boundary

Cuda kernel

```
threadsperblock = 128

# cuda jit kernel definition. We define the data types
@cuda.jit('void(float32[:], float32[:], float32[:])')
def cu_substraction(array_A,array_B,result):

    """ Here, we define the kernel """

    bx = cuda.blockIdx.x
    bw = cuda.blockDim.x
    tx = cuda.threadIdx.x

    # We calculate the unique thread ID
    tid = tx + bx * bw

    if tid < N:
        result[tid] = array_A[tid] - array_B[tid]


# Defining the array.
array_A = np.arange(N, dtype = np.float32)
array_B = np.arange(N, dtype = np.float32)
result = np.zeros(N, dtype = np.float32)

# We copy them to the device
d_array_A = cuda.to_device(array_A)
d_array_B = cuda.to_device(array_B)
d_result = cuda.to_device(result)

blockpergrid = N + (threadsperblock - 1) // threadsperblock

# We call the kernel
#cu_substraction[blockpergrid, threadsperblock](array_A, array_B, result)
cu_substraction[blockpergrid, threadsperblock](d_array_A, d_array_B, d_result)
```

We define the array and thanks to cuda we may send it directly to the GPU to improve the performance.

Cuda kernel

```
threadsperblock = 128

# cuda jit kernel definition. We define the data types
@cuda.jit('void(float32[:], float32[:], float32[:])')
def cu_substraction(array_A,array_B,result):

    """ Here, we define the kernel """

    bx = cuda.blockIdx.x
    bw = cuda.blockDim.x
    tx = cuda.threadIdx.x

    # We calculate the unique thread ID
    tid = tx + bx * bw

    if tid < N:
        result[tid] = array_A[tid] - array_B[tid]


# Defining the array.
array_A = np.arange(N, dtype = np.float32)
array_B = np.arange(N, dtype = np.float32)
result = np.zeros(N, dtype = np.float32)

# We copy them to the device
d_array_A = cuda.to_device(array_A)
d_array_B = cuda.to_device(array_B)
d_result = cuda.to_device(result)

blockpergrid = N + (threadsperblock - 1) // threadsperblock

# We call the kernel
#cu_substraction[blockpergrid, threadsperblock](array_A, array_B, result)
cu_substraction[blockpergrid, threadsperblock](d_array_A, d_array_B, d_result)
```

we define the number of Blocks per grid.

Thread per block is also warp size!!

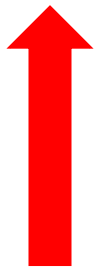
Cuda kernel

```
threadsperblock = 128

# cuda jit kernel definition. We define the data types
@cuda.jit('void(float32[:], float32[:], float32[:])')
def cu_substraction(array_A,array_B,result):

    """ Here, we define the kernel """

    bx = cuda.blockIdx.x
    bw = cuda.blockDim.x
    tx = cuda.threadIdx.x

    # We calculate the unique thread ID
    tid = tx + bx * bw

    if tid < N:
        result[tid] = array_A[tid] - array_B[tid]


# Defining the array.
array_A = np.arange(N, dtype = np.float32)
array_B = np.arange(N, dtype = np.float32)
result = np.zeros(N, dtype = np.float32)

# We copy them to the device
d_array_A = cuda.to_device(array_A)
d_array_B = cuda.to_device(array_B)
d_result = cuda.to_device(result)

blockpergrid = N + (threadsperblock - 1) // threadsperblock

# We call the kernel
#cu_substraction[blockpergrid, threadsperblock](array_A, array_B, result)
cu_substraction[blockpergrid, threadsperblock](d_array_A, d_array_B, d_result)
```

we call the kernel

Cuda kernel

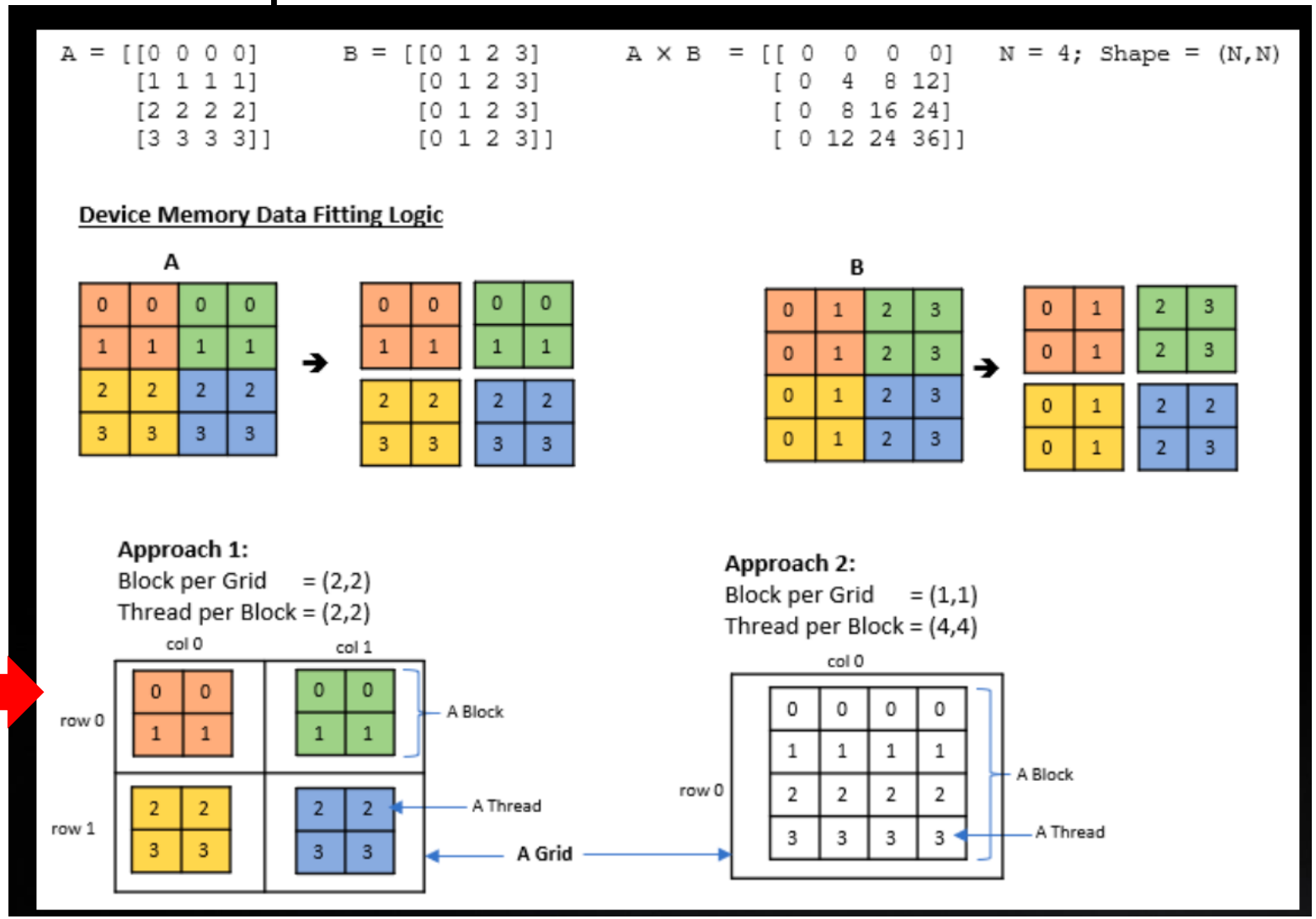
```
# cuda jit kernel definition. We define the data types
@cuda.jit('void(float32[:], float32[:], float32[:])')
def cu_substraction(array_A,array_B,result):

    tid = cuda.grid(1)

    if tid < N:
        result[tid] = array_A[tid] - array_B[tid]
```



Here, we have a slightly cleaner code by calling cuda grid

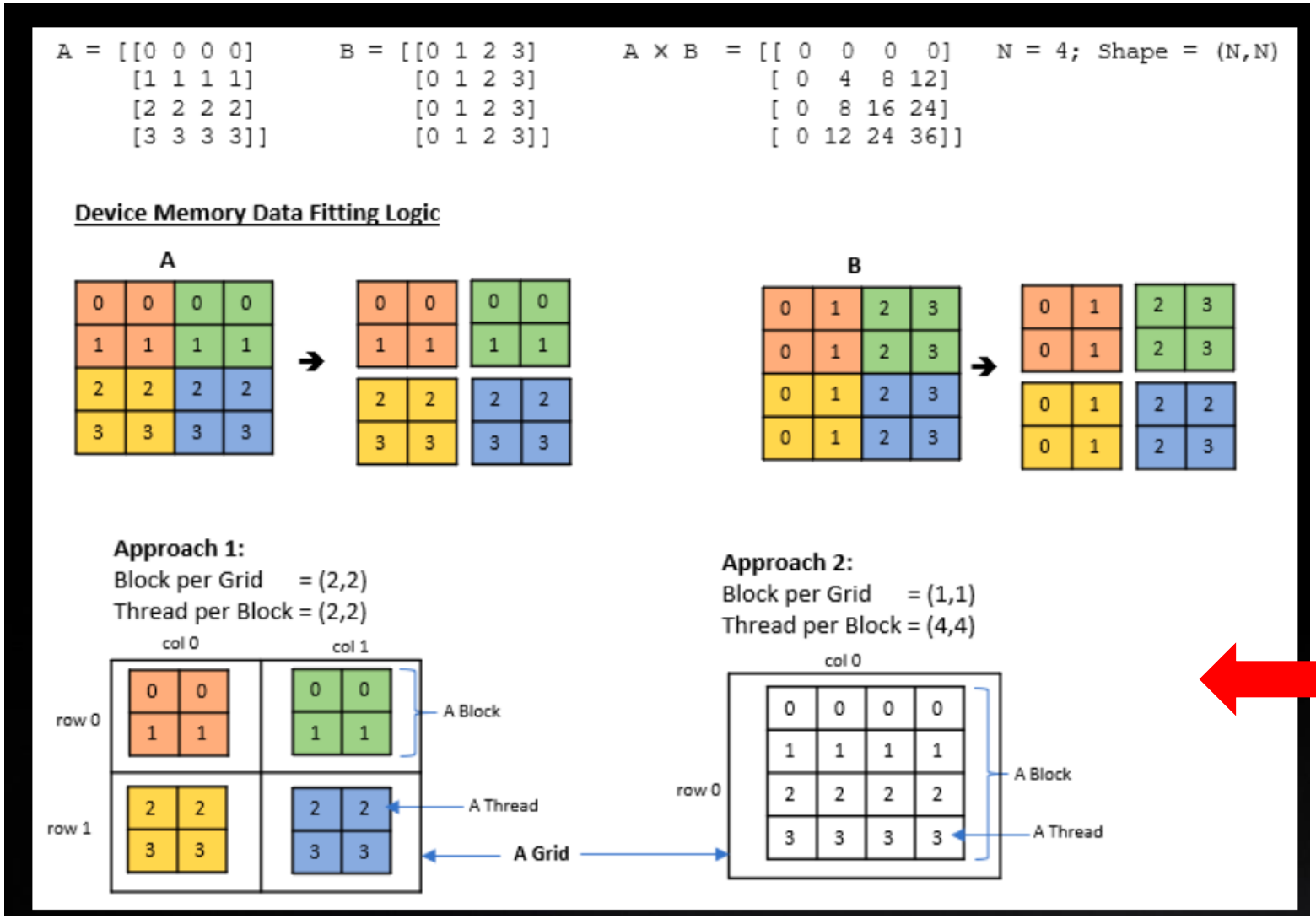
Matrix concept

Option 1: Spread array across blocks



Source: Openhackaton Python numba Princeton hackaton

Matrix concept



Option 2: If the number of threads per block is exceeded it will not work

Source: Openhackaton Python numba Princeton hackaton


III – Little introduction
with cupy

What is cupy?

- Open source library providing GPU accelerated computing with Python.

- Compatible support for Numpy multidimensional arrays.

Source: Openhackaton Python numba Princeton hackaton

How to install it?

- Use apptainer to download: https://hub.docker.com/r/cupy/cupy/

```
apptainer pull docker://cupy/cupy
```

- Use anaconda”

  Install it with mamba following this: https://github.com/kf-cuanschutz/CU-Anschutz-HPC-documentation/blob/main/mamba_tutorial.md

mamba install –c conda-forge cupy cutensor cudatoolkit

Source: Openhackaton Python cupy Princeton hackaton

Cupy is compatible with a lot of CUDA libraries

REQUIREMENTS & ARCHITECTURE

4. Full RAPIDS Package

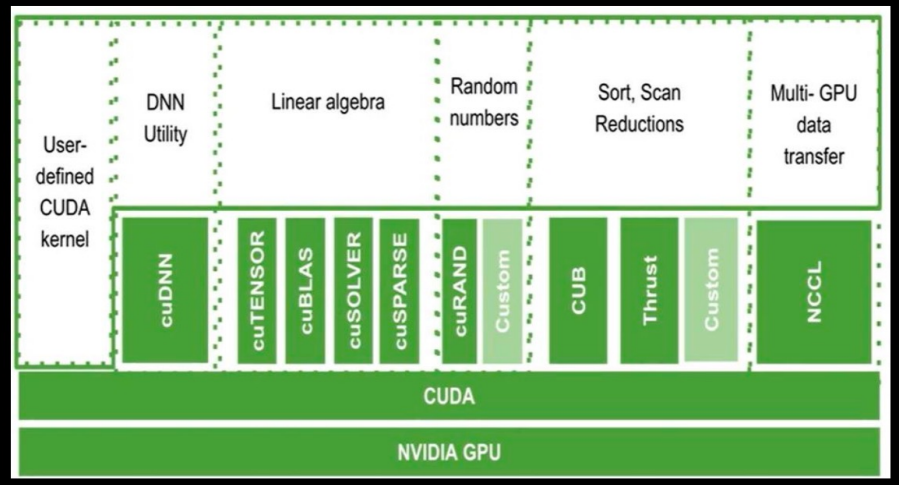
	Preferred		Advanced	
METHOD	Conda	Docker + Examples	Docker + Dev Env	Source
RELEASE	Stable (0.18)		Nightly (0.19a)	
TYPE	RAPIDS and BlazingSQL		RAPIDS Core (w/o BlazingSQL)	
PACKAGES	All Packages	cuDF	cuML	cuGraph	cuSignal	cuSpatial	cuxfilter
LINUX	Ubuntu 16.04	Ubuntu 18.04	Ubuntu 20.04	CentOS 7	CentOS 8	RHEL 7&8
PYTHON	Python 3.7		Python 3.8			
CUDA	CUDA 10.1.2	CUDA 10.2	CUDA 11.0			

NOTE: Ubuntu 16.04/18.04/20.04 & CentOS 7/8 use the same `conda create` commands.

COMMAND

```
conda create -n rapids-0.18 -c rapidsai -c nvidia -c conda-forge \
    -c defaults rapids-blazing=0.18 python=3.7 cudatoolkit=10.1
```

CuPy Architecture



Source: Openhackaton Python cupy Princeton hackaton

Cupy calculations

```
import cupy as cp
import numpy as np
import time

# CPU Array

start = time.time() #Start
X = np.array([1,2,3,4,5])
end = time.time() # End

# Print duration
print("cpu calculation took {0} seconds)".format(end-start))


start = time.time() #Start
X_gpu = cp.array([1,2,3,4,5])
end = time.time() # End

# Print duration
print("gpu calculation took {0} seconds)".format(end-start))
```

Source: Openhackaton Python cupy Princeton hackaton

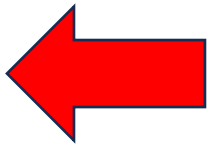
Cupy calculations

```
(cupy_env_2) [kfotso@xsede.org@c3gpu-a9-u31-1 other_example]$ python cupy_code1.py
cpu calculation took 9.536743164062e-06 seconds)
gpu calculation took 0.2535276412963867 seconds)
```

The array was very small and thus the calculation was not complex enough to justify GPU usage

Numpy to cupy conversion

```
# We want to demonstrate conversion from numpy to cupy
start = time.time() #Start
rand_arr = np.random.rand(1000, 1000)
end = time.time() # End

# Print duration
print("Random array  took {0} seconds)".format(end-start))

start = time.time() #Start
rand_arr_gpu = cp.asarray(rand_arr)
end = time.time() # End

# Print duration
print("Conversion from CPU to GPU  took {0} seconds)".format(end-start))
```



Now, we want to generate a bigger array

Numpy to cupy conversion

```
Random array  took 0.006761789321899414 seconds)
Conversion from CPU to GPU  took 0.0030117034912109375 seconds)
(cupy_env_2) [kfotso@xsede.org@c3gpu-a9-u31-1 other_example]$
```

GPU completed faster this time.

Element wise kernel

ELEMENTWISE KERNEL

- **Example : z = x*w + b**

```
import cupy as cp
input_list = 'float32 x , float32 w, float32 b'
output_list = 'float32 z'
code_body  = 'z =  (x * w) + b'
# elementwisekernel class defined
dnnLayerNode = cp.ElementwiseKernel(input_list, output_list, code_body,'dnnLayerNode')
x = cp.arange(9, dtype=cp.float32).reshape(3,3)
w = cp.arange(9, dtype=cp.float32).reshape(3,3)
b = cp.array([-0.5], dtype=cp.float32)
z = cp.empty((3,3), dtype=cp.float32)
# kernel call with argument passing
dnnLayerNode(x,w,b,z)
print(z)
#output
[[-0.5  0.5  3.5]
 [ 8.5 15.5 24.5]
 [35.5 48.5 63.5]]
```

Data type

Input argument list

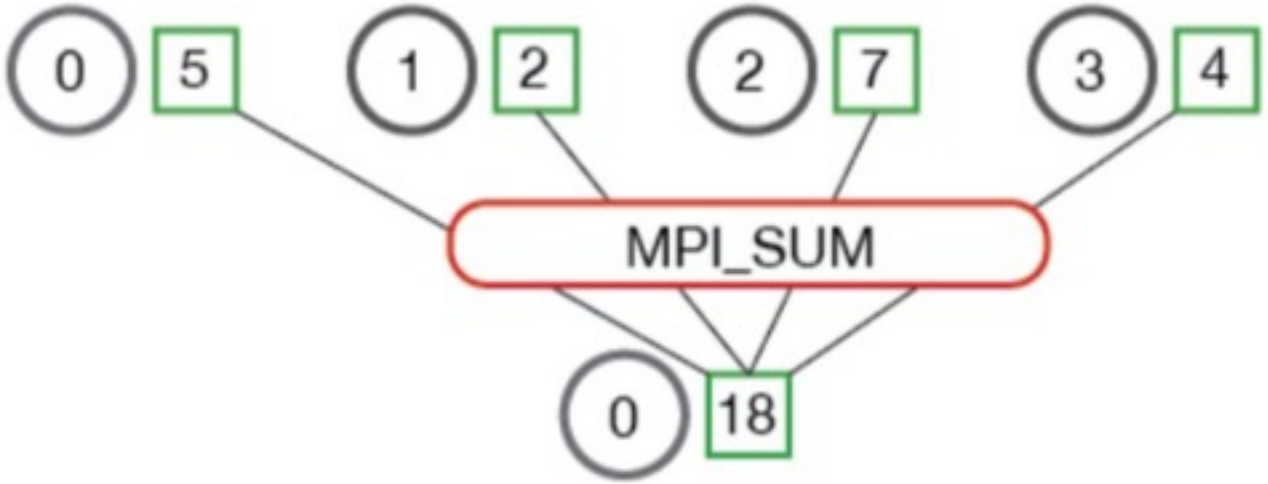
Output argument list

A loop body code

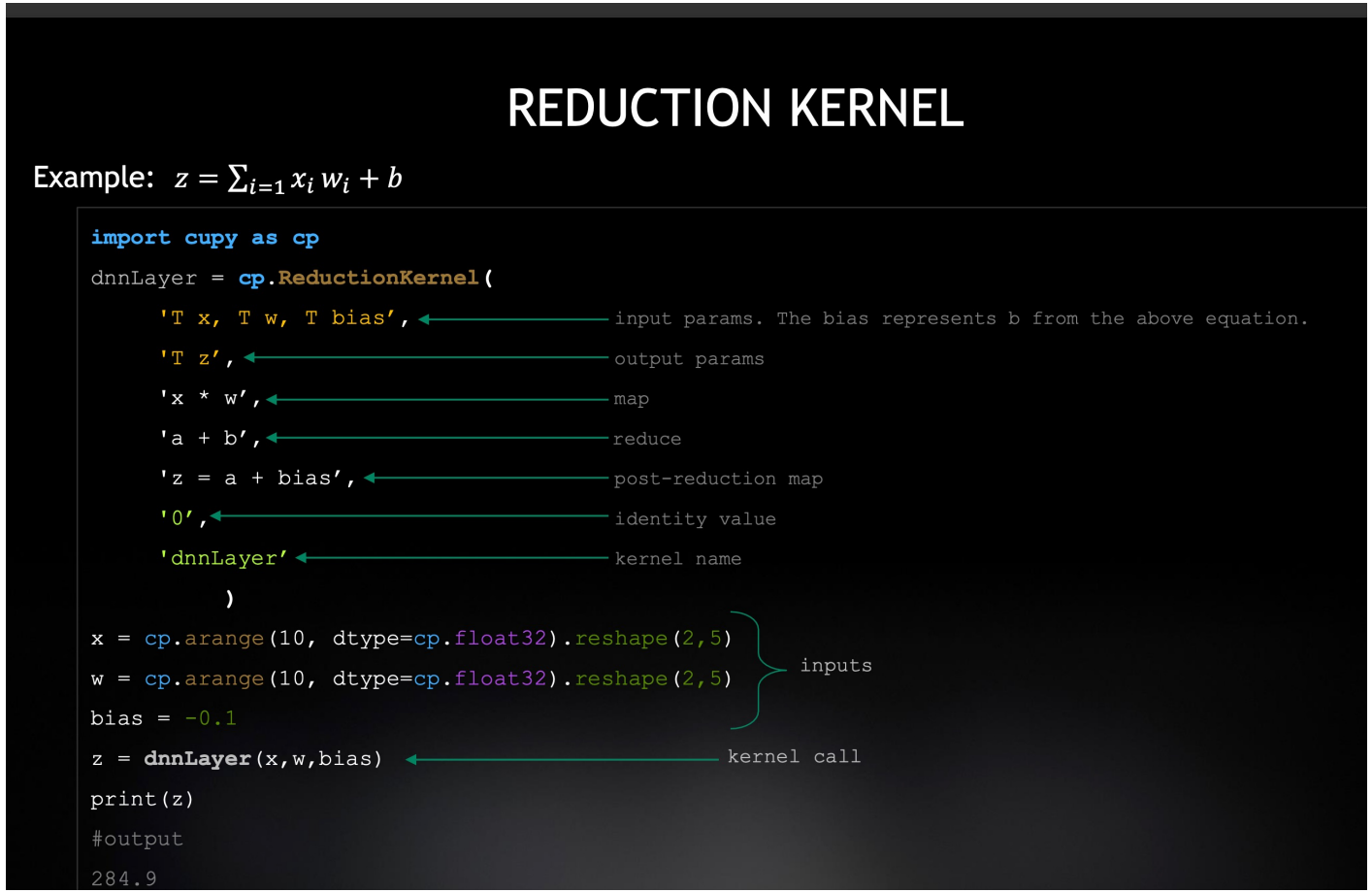
Source: Openhackaton Python cupy Princeton hackaton

Collective communication

- Reduce: Takes values from an array on each processes and reduces them to a single result root process.



Overview of some parallel



Source: Openhackaton Python cupy Princeton hackaton

Next time

- We will go a little more in depth with numba and cupy
- We will learn GPU profiling