Introduction to GPU Acceleration

October 24, 2025
Andrew Monaghan
rc-help@colorado.edu

View the Slides

https://github.com/ResearchComputing/Intro_GPU_Acceleration

Meet the User Support Team

Layla
Freeborn

Brandon
Reyes

Andy
Monaghan

Michael
Schneider

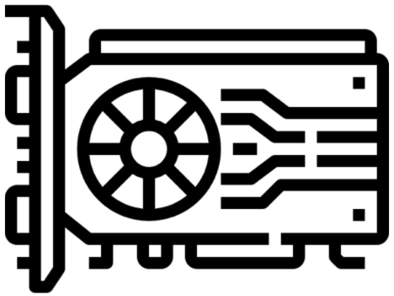
John
Reiland

Dylan
Gottlieb

Mohal
Khandelwal

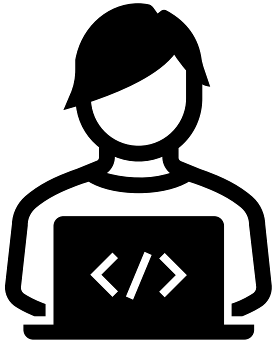
Ragan
Lee

Session Overview



Basics Of GPUs

1



Code Optimization

2

Monitoring GPU Usage

3

GPU Icon

CPUs vs GPUs

Processing Unit

CPUs vs GPUs

Central

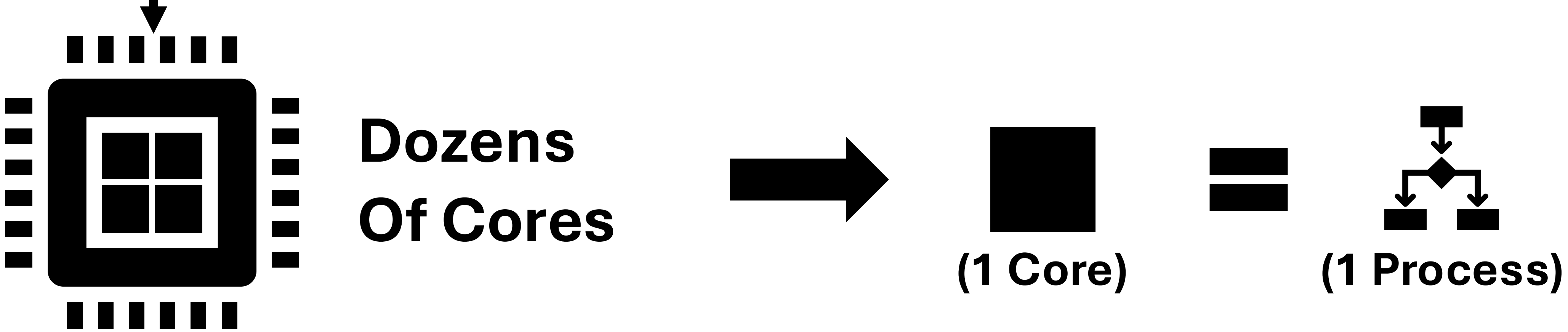
Processing Unit

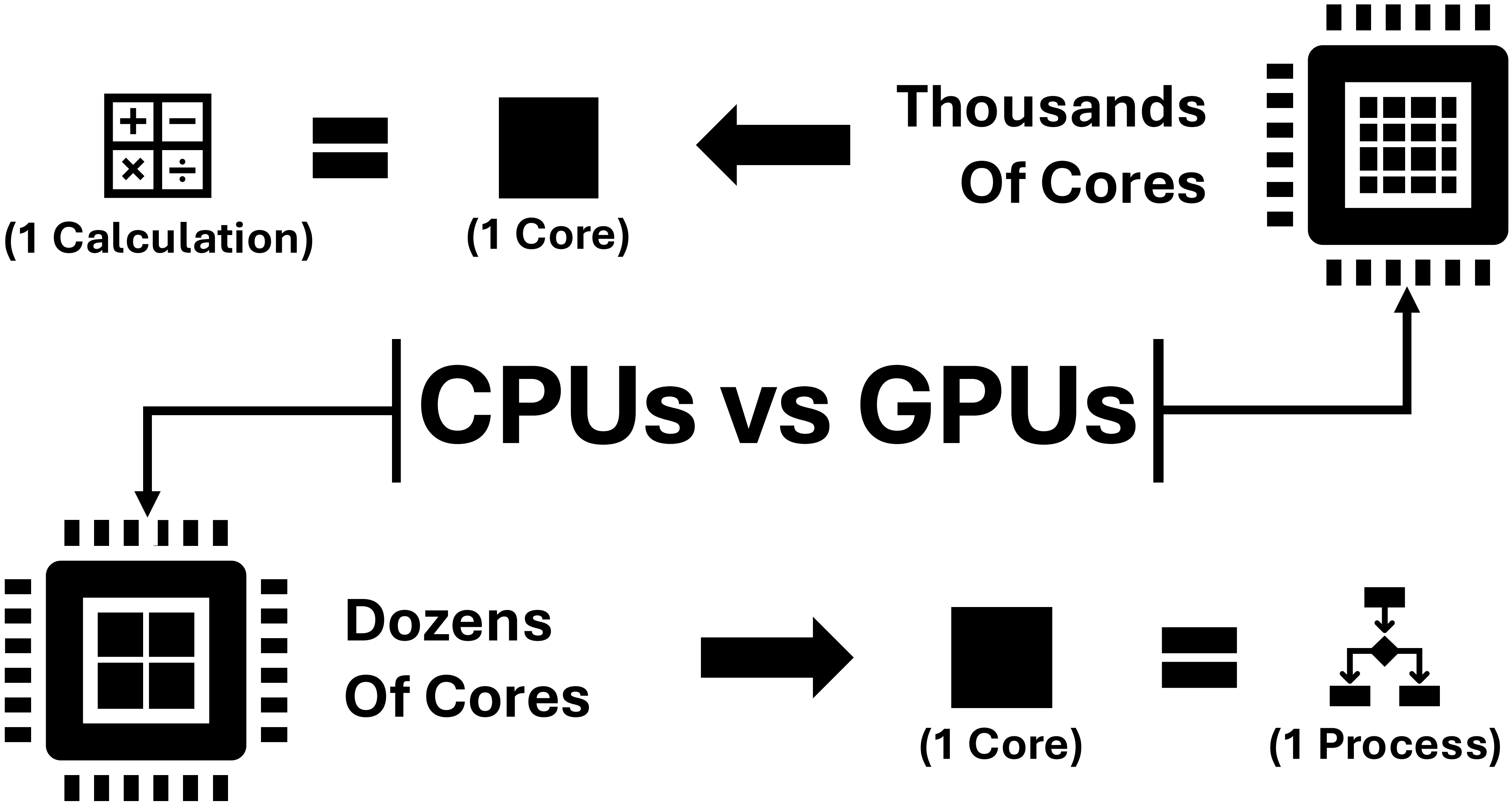
CPUs vs GPUs

Central

Graphics

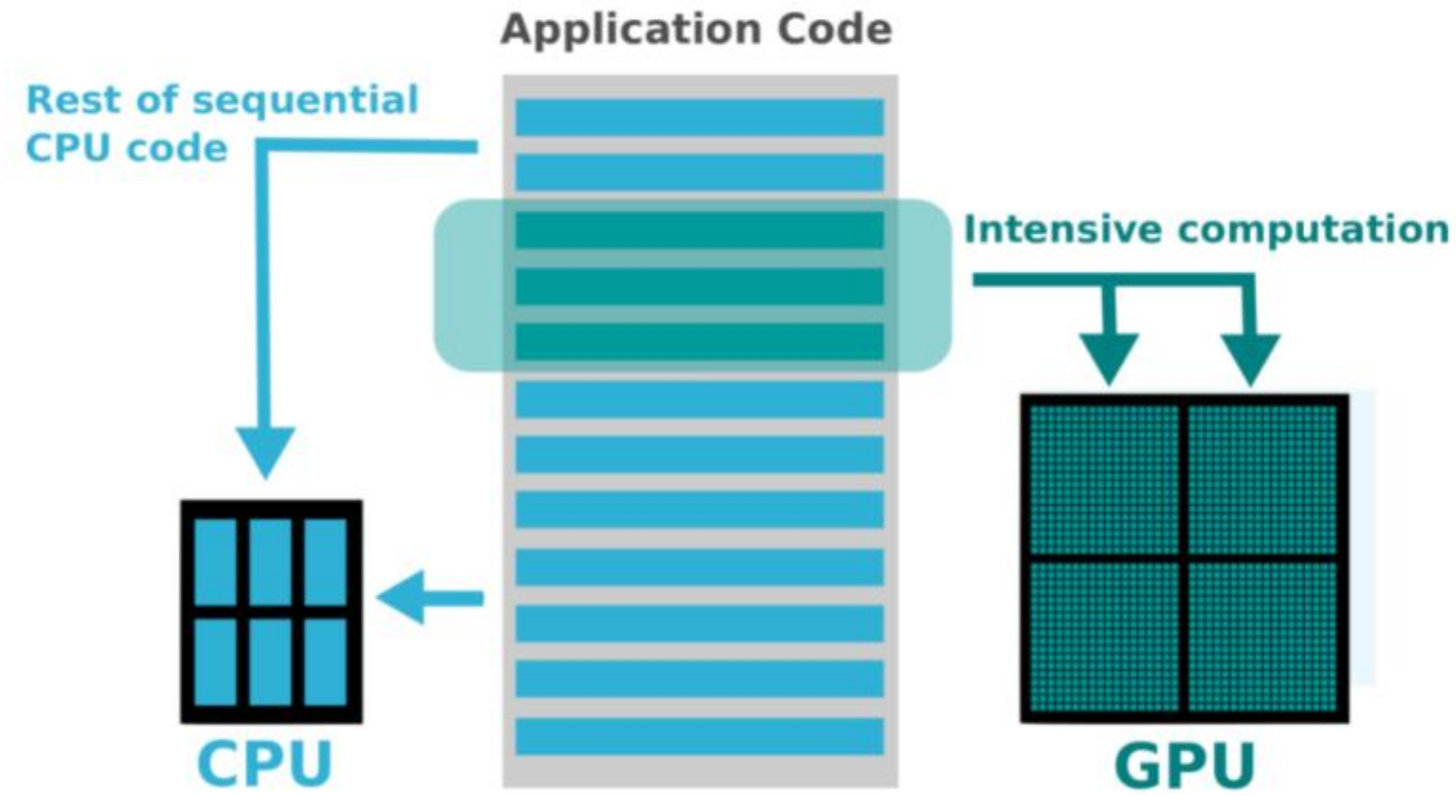
CPUs vs GPUs





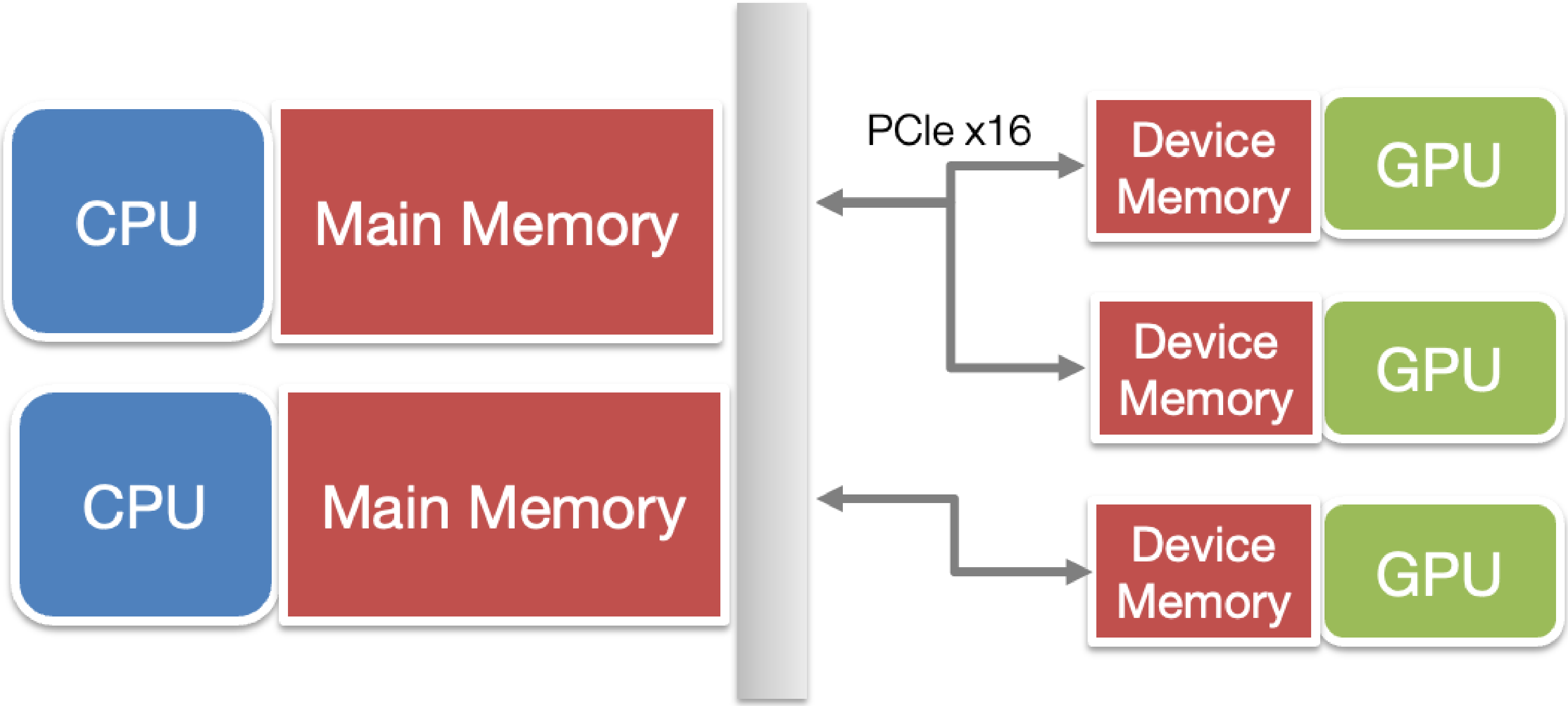
(1 Calculation)
(1 Core)
Thousands
Of Cores
CPUs vs GPUs
Dozens
Of Cores
(1 Core)
(1 Process)

Computational Offloading



Graphic Source

Data Offloading CPU -> GPU



Data needs to be copied from CPU to GPU, computation is on the GPU, then output is transferred back to CPU.

Criteria for GPU Acceleration

1. The time spent on computationally intensive parts of the workflow exceeds the time spent transferring data to and from GPU memory
2. Computations are massively parallel- the computations can be broken down into hundreds or thousands of independent units of work

Factors Affecting GPU Speedup

1. Computational Intensity
2. Data Dependency
3. Data Type
4. Code/Algorithmic Complexity

Factors Affecting GPU Speedup

1 Computational Intensity

GPUs perform best when there is a lot of processing compared to loading and storing data (FLOP per Byte ratio).

Factors Affecting GPU Speedup

2 Data Dependency

Data Dependency- A situation in which an instruction is dependent on a result from a sequentially previous instruction before it can complete its execution. (avoid!)

Factors Affecting GPU Speedup

3 Data Type

- Operations on strings are slow unless they can be treated as numbers.
- Performance per GPU can vary if workflows include 16-bit and 64-bit floats.

Factors Affecting GPU Speedup

4 Code/Algorithmic Complexity

Simple code is better ported to GPUs.

- Deeply-branched code and while-loops may perform poorly on GPUs
- Recursive functions need to be re-written

Alpine GPU Partitions

	NVIDIA			AMD
Partition	aa100 (atesting_a100)	al40	gh200*	ami100 (attesting_mi100)
# Nodes	12	3	2	8
GPU Type	A100	L40	GH200	MI100
GPUs/Node	3	3	1	3
Cores/GPU	7k	15K	17k	7.7k
VRAM/GPU	40 / 80	48	96	32
Purpose	General	AI Inference	AI Training, High Data I/O	General

Exploring Alpine GPU Partitions

Try these commands.

```
$ ssh <username>@login.rc.colorado.edu

$ sinfo --Format Partition

$ sinfo --partition aa100,ami100,atesting_a100,atesting_mi100 --Format=Partition,Nodes,Time

$ scontrol show partition atesting_a100

$ scontrol show node c3gpu-c2-u13
```

Requesting Alpine GPUs with Slurm batch script

Slurm flags needed to request 1 NVIDIA GPU node with 2 GPUs and 20 CPU cores

```
--partition=aa100
--gres=gpu:2
--ntasks=20
```

in a job script submitted with ***sbatch*** command

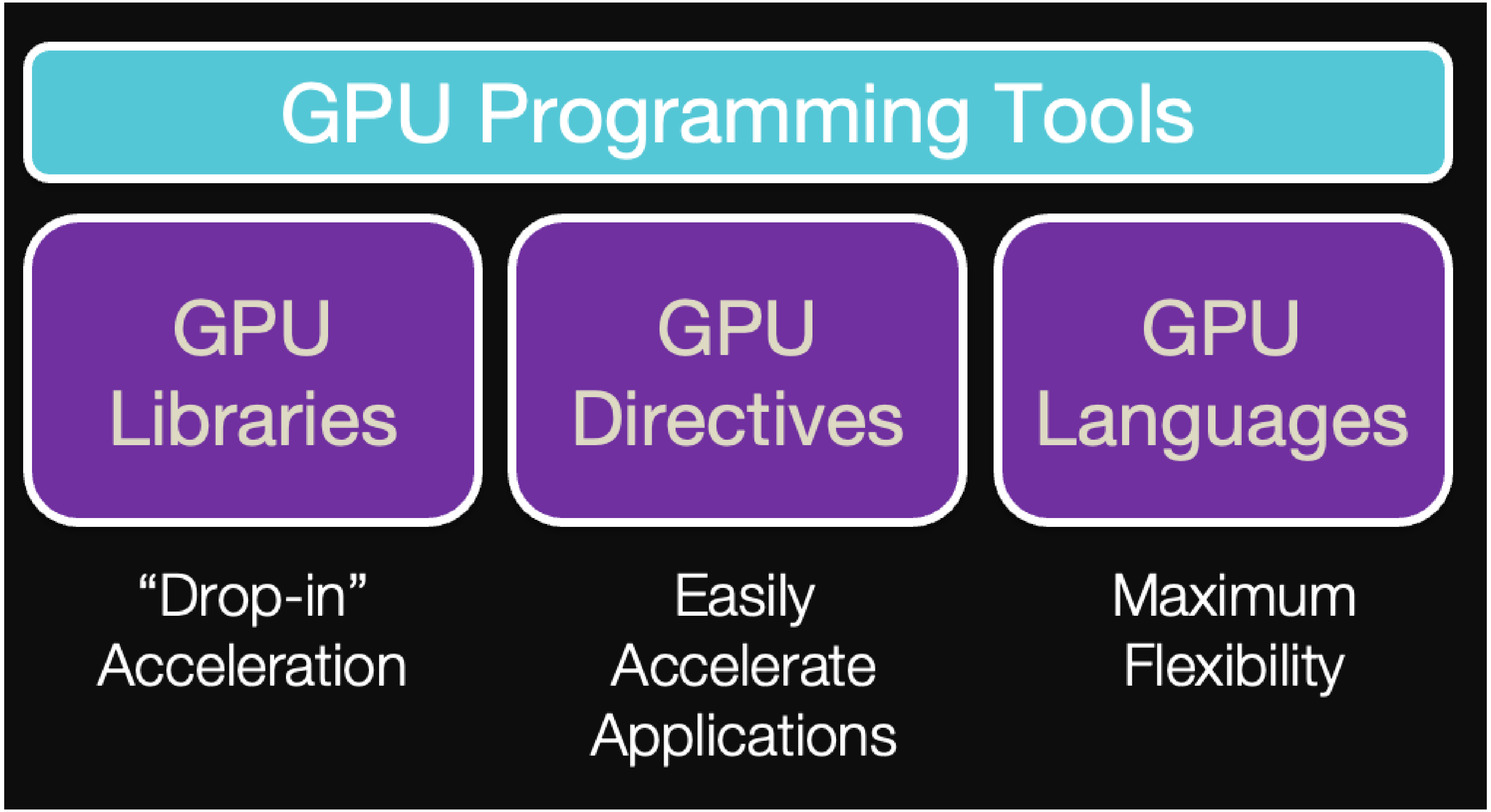
```
#SBATCH --partition=aa100
#SBATCH --qos=normal
#SBATCH --gres=gpu:2
#SBATCH --nodes=1
#SBATCH --ntasks=20
#SBATCH --time=12:00:00
#SBATCH --job-name=gpu_test
#SBATCH --output=gpu_test_%j.out
#SBATCH --error=gpu_test_%j.err
```

Requesting Alpine GPUs with Slurm interactive

```
#request one NVIDIA GPU  on the A100 testing partition:

sinteractive --partition=atesting_a100 --qos=testing --gres=gpu:1 --nodes=1 --ntasks=10 --
time=1:00:00
```

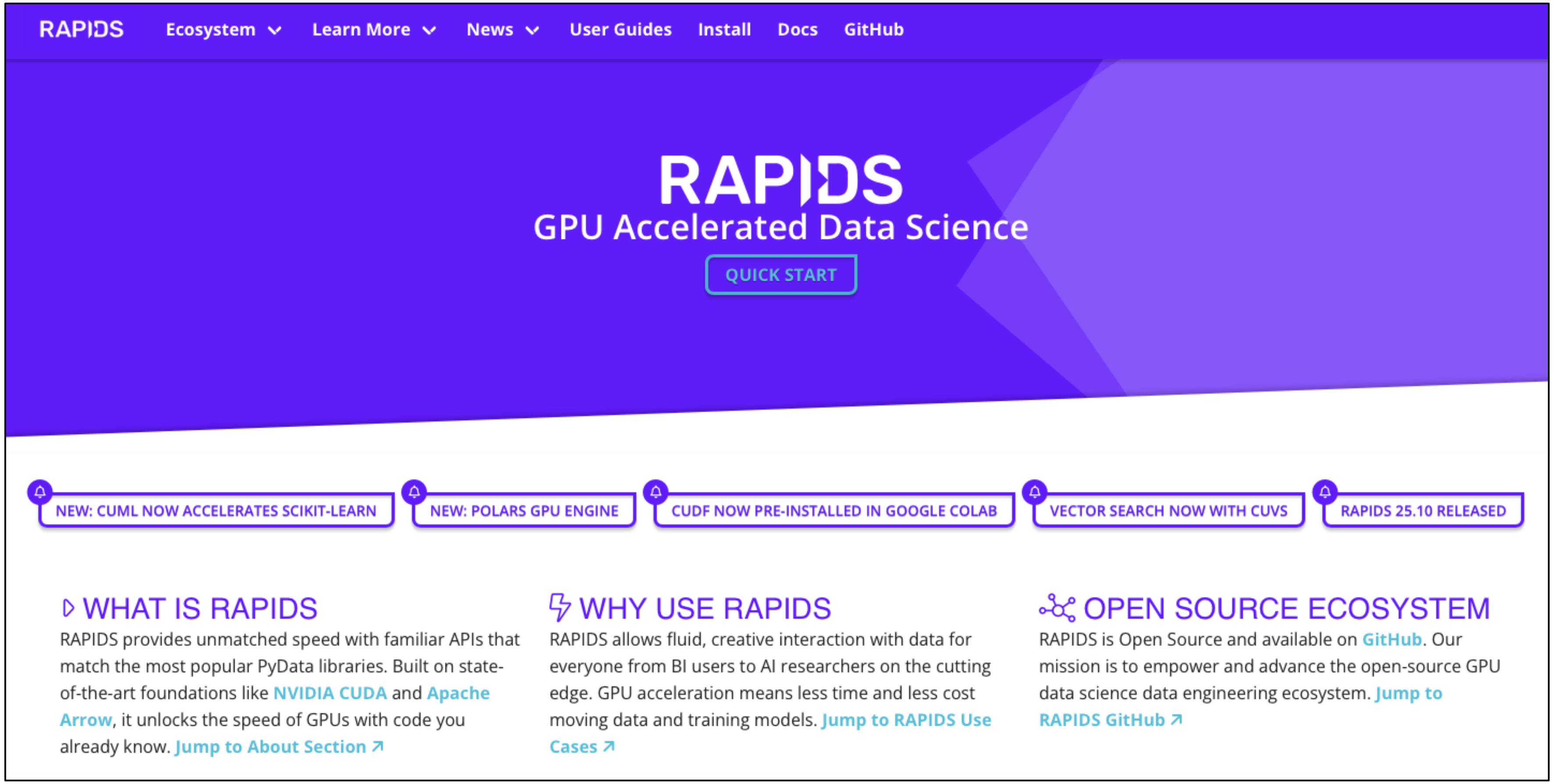
Note: Queue waits for non-testing GPU partitions are typically long (12-24 hours). Therefore, interactive jobs -- which require you to wait for the job to start -- will usually be run on the testing partitions during the onboarding and debugging stages of your workflow.

Code Optimization



GPU Icon

GPU Libraries – Drop in Replacement

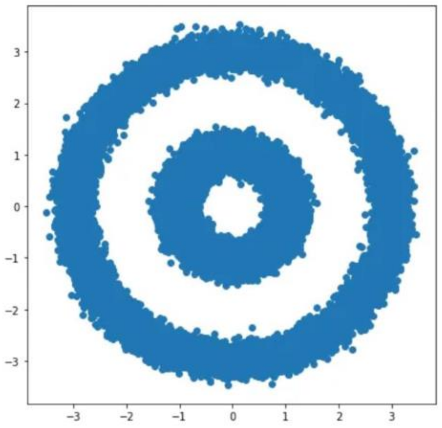


https://rapids.ai

GPU Libraries – Drop in Replacement

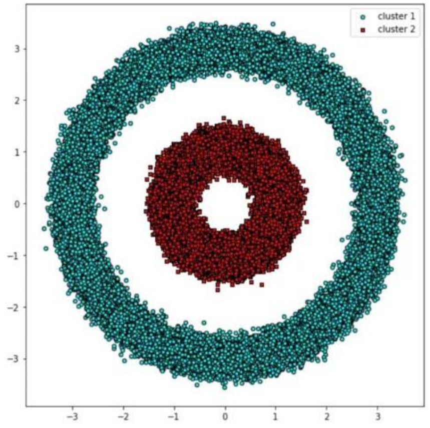
```
#create dataset with 100,000 points
from sklearn.datasets import make_circles
X, y = make_circles(n_samples=int(1e5), factor=.35, noice=.05)

#run DBSCAN clustering algorithm
from sklearn.cluster import DBSCAN
db = DBSCAN(eps=0.6, min_samples=2)
y_db = db.fit_predict(X)
```

Dataset



Input

Clusters



Result

GPU Libraries – Drop in Replacement

```
#create dataset with 100,000 points
from sklearn.datasets import make_circles
X, y = make_circles(n_samples=int(1e5), factor=.35, noice=.05)
```

```
#run DBSCAN clustering algorithm
from sklearn.cluster import DBSCAN
db = DBSCAN(eps=0.6, min_samples=2)
y_db = db.fit_predict(X)
```

GPU Libraries – Drop in Replacement

```
#create dataset with 100,000 points
from sklearn.datasets import make_circles
X, y = make_circles(n_samples=int(1e5), factor=.35, noice=.05)

#convert dataset to Pandas DataFrame




#run DBSCAN clustering algorithm
from sklearn.cluster import DBSCAN
db = DBSCAN(eps=0.6, min_samples=2)
y_db = db.fit_predict(X)
```

GPU Libraries – Drop in Replacement

```
#create dataset with 100,000 points
from sklearn.datasets import make_circles
X, y = make_circles(n_samples=int(1e5), factor=.35, noice=.05)

#convert dataset to Pandas DataFrame
import pandas as pd
import cudf
X_df = pd.DataFrame({‘fea%d’%i: X[:,i] for i in range(X.shape[1])})
X_gpu = cudf.DataFrame.from_pandas(X_df)

#run DBSCAN clustering algorithm
from sklearn.cluster import DBSCAN
db = DBSCAN(eps=0.6, min_samples=2)
y_db = db.fit_predict(X)
```

GPU Libraries – Drop in Replacement

```
#create dataset with 100,000 points
from sklearn.datasets import make_circles
X, y = make_circles(n_samples=int(1e5), factor=.35, noice=.05)

#convert dataset to Pandas DataFrame
import pandas as pd
import cudf
X_df = pd.DataFrame({‘fea%d’%i: X[:,i] for i in range(X.shape[1])})
X_gpu = cudf.DataFrame.from_pandas(X_df)

#run DBSCAN clustering algorithm
from sklearn.cluster import DBSCAN
db = DBSCAN(eps=0.6, min_samples=2)
y_db = db.fit_predict(X)
```

❯

```
#run GPU-accelerated DBSCAN
from cuml import DBSCAN
```

GPU Libraries – Drop in Replacement

```
#create dataset with 100,000 points
from sklearn.datasets import make_circles
X, y = make_circles(n_samples=int(1e5), factor=.35, noice=.05)

#convert dataset to Pandas DataFrame
import pandas as pd
import cudf
X_df = pd.DataFrame({'fea%d'%i: X[:,i] for i in range(X.shape[1])})
X_gpu = cudf.DataFrame.from_pandas(X_df)
```

~~#run DBSCAN clustering algorithm~~
~~from sklearn.cluster import DBSCAN~~

```
db = DBSCAN(eps=0.6, min_samples=2)
y_db = db.fit_predict(X)
```

```
#run GPU-accelerated DBSCAN
from cuml import DBSCAN
```

GPU Libraries – Drop in Replacement

```
#create dataset with 100,000 points
from sklearn.datasets import make_circles
X, y = make_circles(n_samples=int(1e5), factor=.35, noice=.05)

#convert dataset to Pandas DataFrame
import pandas as pd
import cudf
X_df = pd.DataFrame({‘fea%d’%i: X[:,i] for i in range(X.shape[1])})
X_gpu = cudf.DataFrame.from_pandas(X_df)

#run GPU-accelerated DBSCAN
from cuml import DBSCAN
db = DBSCAN(eps=0.6, min_samples=2)
y_db = db.fit_predict(X)
```

1

2

GPU-Enabled Frameworks (deep learning)

TensorFlow

Known for: comprehensiveness, scalability

PyTorch

Known for: Flexibility, pythonic, intuitive

K Keras

Known for: Ease of use

mxnet

Known for: Distributed training

GPU Compiler Directives

OpenACC

Key Terms

- Host == CPU
- Device == GPU
- Kernel == Functions launched on GPU

GPU Compiler Directives

Kernel directives
Generate parallel accelerator kernels for the loop following the directive.

GPU Compiler Directives

OpenACC

Kernel directives

Generate parallel accelerator kernels for the loop following the directive.

```
//Hello_World_OpenACC.c
void Print_Hello_World()
{
  #pragma acc kernels
  for(int i=0; i<5; i++)
  {
    printf("Hello World!\n")
  }
}
```

GPU Compiler Directives

Kernel directives
Generate parallel accelerator kernels for the loop following the directive.

```
//Hello_World_OpenACC.c
void Print_Hello_World()
{
  #pragma acc kernels
  for(int i=0; i<5; i++)
  {
    printf(“Hello World!\n”)
  }
}
```

Data directives
Generate code to manage specific data operations to support parallelism

GPU Compiler Directives

OpenACC

Kernel directives
Generate parallel accelerator kernels for the loop following the directive.

```
//Hello_World_OpenACC.c
void Print_Hello_World()
{
  #pragma acc kernels
  for(int i=0; i<5; i++)
  {
    printf("Hello World!\n")
  }
}
```

Data directives
Generate code to manage specific data operations to support parallelism

```
//Hello_World_OpenACC.c
#pragma acc data copy(a)
{
  #pragma acc kernels
  for(int i=0; i<5; i++)
  {
    printf("Hello World!\n")
  }
}
```

GPU Languages

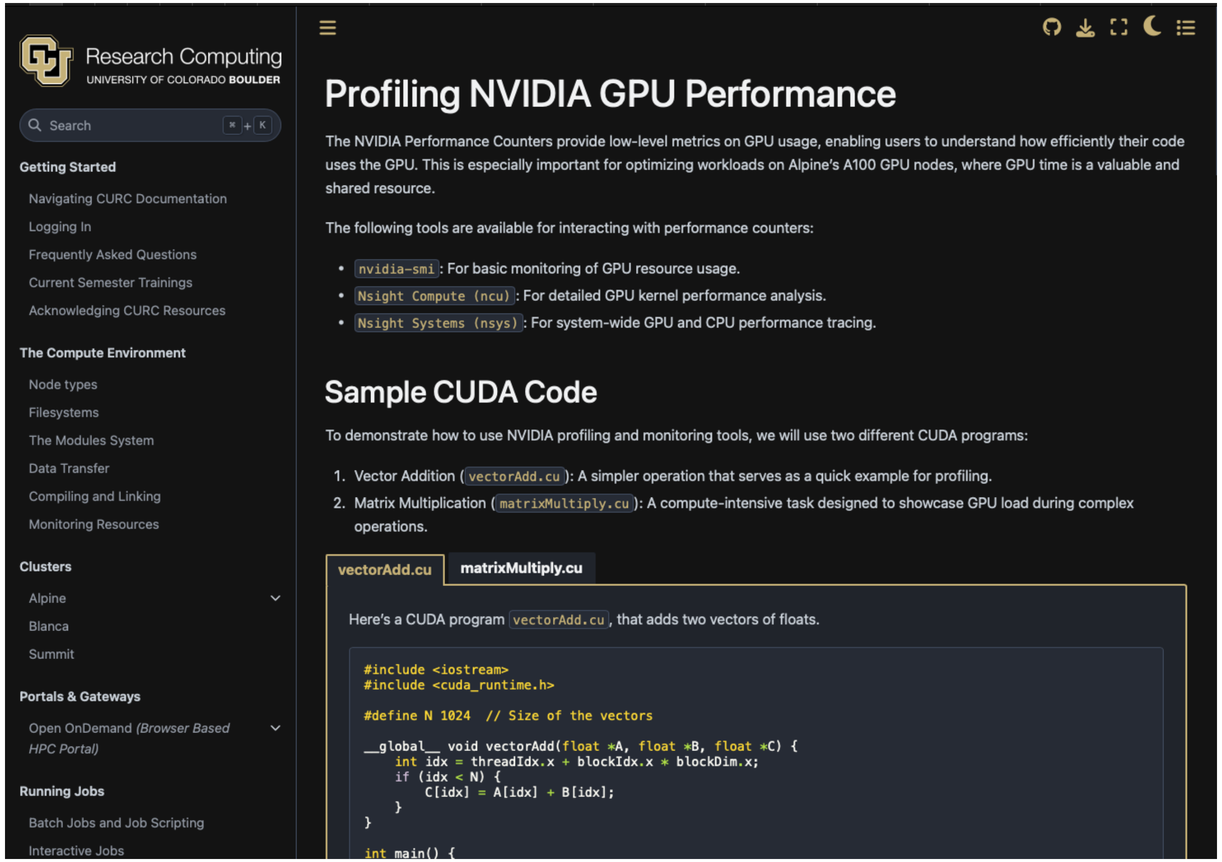
- OpenCL (NVIDIA, AMD, & CPUs)
  - Flexible / portable option
- HIP (AMD -> NVIDIA)
  - AMD developed
  - Can convert CUDA code via `hippify`
- CUDA (NVIDIA only)
  - Most robust and largest developer community

Monitoring GPU Usage

- Nvidia-smi
- rocm-smi

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 510.47.03    Driver Version: 510.47.03    CUDA Version: 11.6     |
|-------------------------------+----------------------+----------------------+
| 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-PCI...  Off  | 00000000:21:00.0 Off |                    0 |
| N/A   36C    P0    40W / 250W |      0MiB / 40960MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   1  NVIDIA A100-PCI...  Off  | 00000000:81:00.0 Off |                    0 |
| N/A   36C    P0    40W / 250W |      0MiB / 40960MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   2  NVIDIA A100-PCI...  Off  | 00000000:E2:00.0 Off |                    0 |
| N/A   37C    P0    40W / 250W |      0MiB / 40960MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Monitoring GPU Usage on NVIDIA GPUs



Research Computing
UNIVERSITY OF COLORADO BOULDER

Search

Getting Started

- Navigating CURC Documentation
- Logging In
- Frequently Asked Questions
- Current Semester Trainings
- Acknowledging CURC Resources

The Compute Environment

- Node types
- Filesystems
- The Modules System
- Data Transfer
- Compiling and Linking
- Monitoring Resources

Clusters

- Alpine
- Blanca
- Summit

Portals & Gateways

- Open OnDemand *(Browser Based HPC Portal)*

Running Jobs

- Batch Jobs and Job Scripting
- Interactive Jobs

Profiling NVIDIA GPU Performance

The NVIDIA Performance Counters provide low-level metrics on GPU usage, enabling users to understand how efficiently their code uses the GPU. This is especially important for optimizing workloads on Alpine's A100 GPU nodes, where GPU time is a valuable and shared resource.

The following tools are available for interacting with performance counters:

- `nvidia-smi`: For basic monitoring of GPU resource usage.
- `Nsight Compute (ncu)`: For detailed GPU kernel performance analysis.
- `Nsight Systems (nsys)`: For system-wide GPU and CPU performance tracing.

Sample CUDA Code

To demonstrate how to use NVIDIA profiling and monitoring tools, we will use two different CUDA programs:

1. Vector Addition (`vectorAdd.cu`): A simpler operation that serves as a quick example for profiling.
2. Matrix Multiplication (`matrixMultiply.cu`): A compute-intensive task designed to showcase GPU load during complex operations.

vectorAdd.cu | matrixMultiply.cu

Here's a CUDA program `vectorAdd.cu`, that adds two vectors of floats.

```
#include <iostream>
#include <cuda_runtime.h>

#define N 1024  // Size of the vectors

__global__ void vectorAdd(float *A, float *B, float *C) {
    int idx = threadIdx.x + blockIdx.x * blockDim.x;
    if (idx < N) {
        C[idx] = A[idx] + B[idx];
    }
}

int main() {
```

https://curc.readthedocs.io/en/latest/programming/profiling-nvidia-gpu-performance.html

Troubleshooting GPU Workflows

- Is your application and/or code GPU accelerated?

  Confirm that you installed the GPU accelerated version!

- Does your application or code support **multi-**GPU acceleration?

- Is your application ROCM- or CUDA-aware?

  You can't run CUDA code on AMD GPUs. Not all applications are available for AMD GPUs.

- Can your application "see" the GPU?

- Did you request enough CPUs and RAM?

Documentation

https://curc.readthedocs.io/en/latest/

Survey and feedback

Survey: http://tinyurl.com/curc-survey18