GPU Acceleration Part I: An Intro to GPU Acceleration on CURC Alpine

28 Nov 2023

Layla Freeborn, Ph.D.
layla.freeborn@colorado.edu

Slides, Hands-on Tutorial, Scripts

visit
https://github.com/ResearchComputing/Intro_GPU_Acceleration.git or do

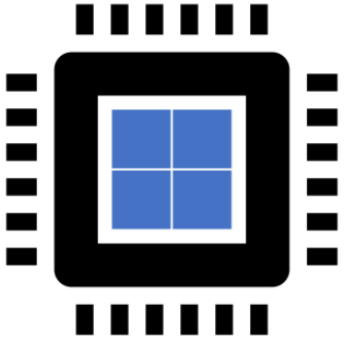
```
git clone
```
https://github.com/ResearchComputing/Intro_GPU_Acceleration.git

Session Overview

1. CPUs vs GPUs
2. Heterogeneous Systems
3. Criteria for GPU Acceleration and Factors Affecting GPU Speedup
4. Alpine GPU Partitions and Requesting GPUs with Slurm
5. GPU Programming Tools
6. GPU Monitoring Tools
7. Basic GPU Application Troubleshooting
8. Self-guided hands-on tutorial*

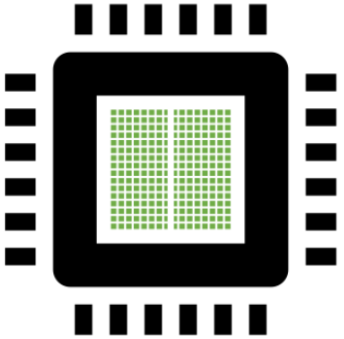
CPUs vs GPUs

Central Processing Unit



- General workhorse
- Contains dozens of cores
- Good for serial processing and handling multiple

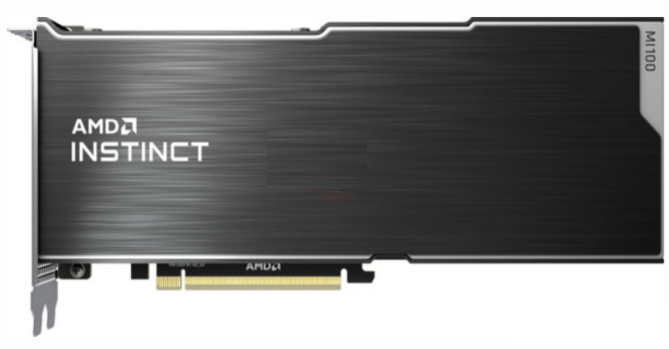
Graphics Processing Unit



- Specialized
- Contains thousands of cores
- Good for parallel processing and handling specific tasks quickly

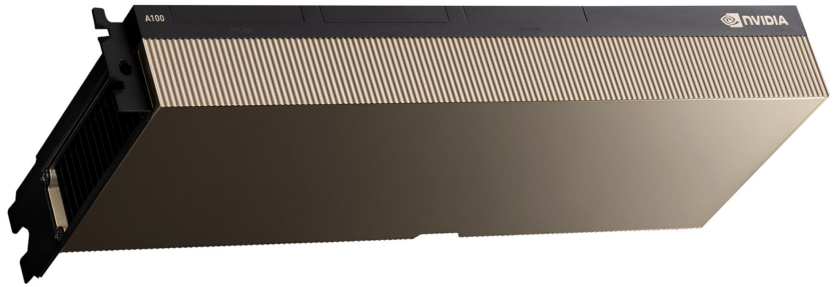
Heterogeneous Systems

MI100 GPU



8 nodes with 3 GPUs per node

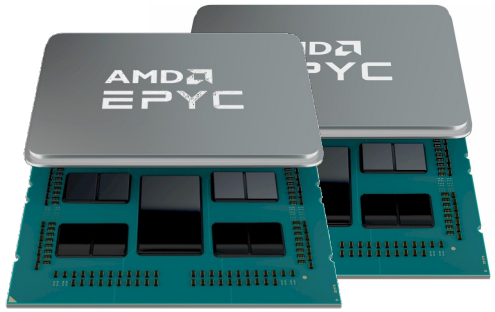
A100 GPU



12 nodes with 3 GPUs per node

- 40 GB and 80 GB VRAM

Milan CPU



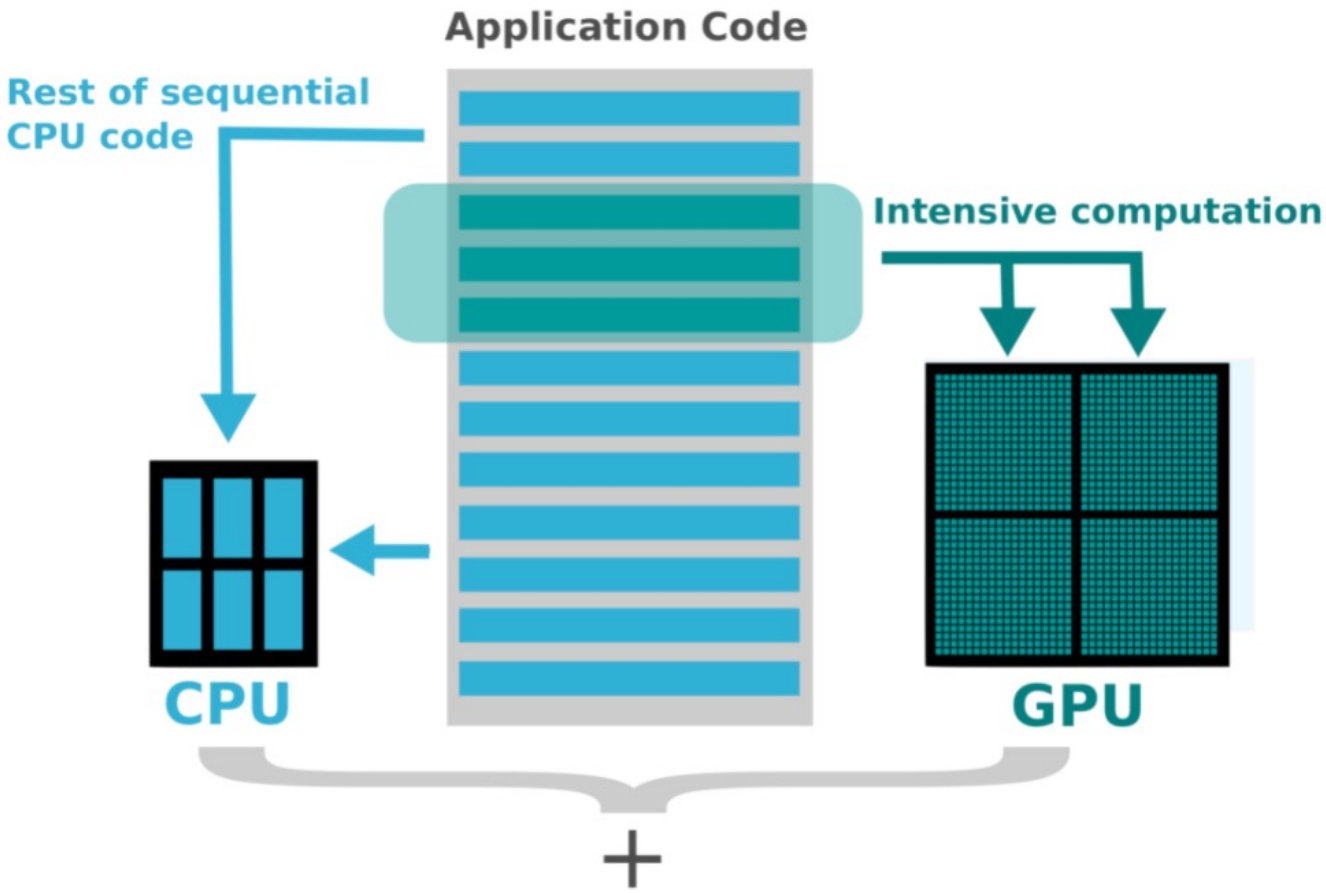
369 nodes

- 347 nodes with 256 GB RAM
- 22 nodes with 1 TB RAM

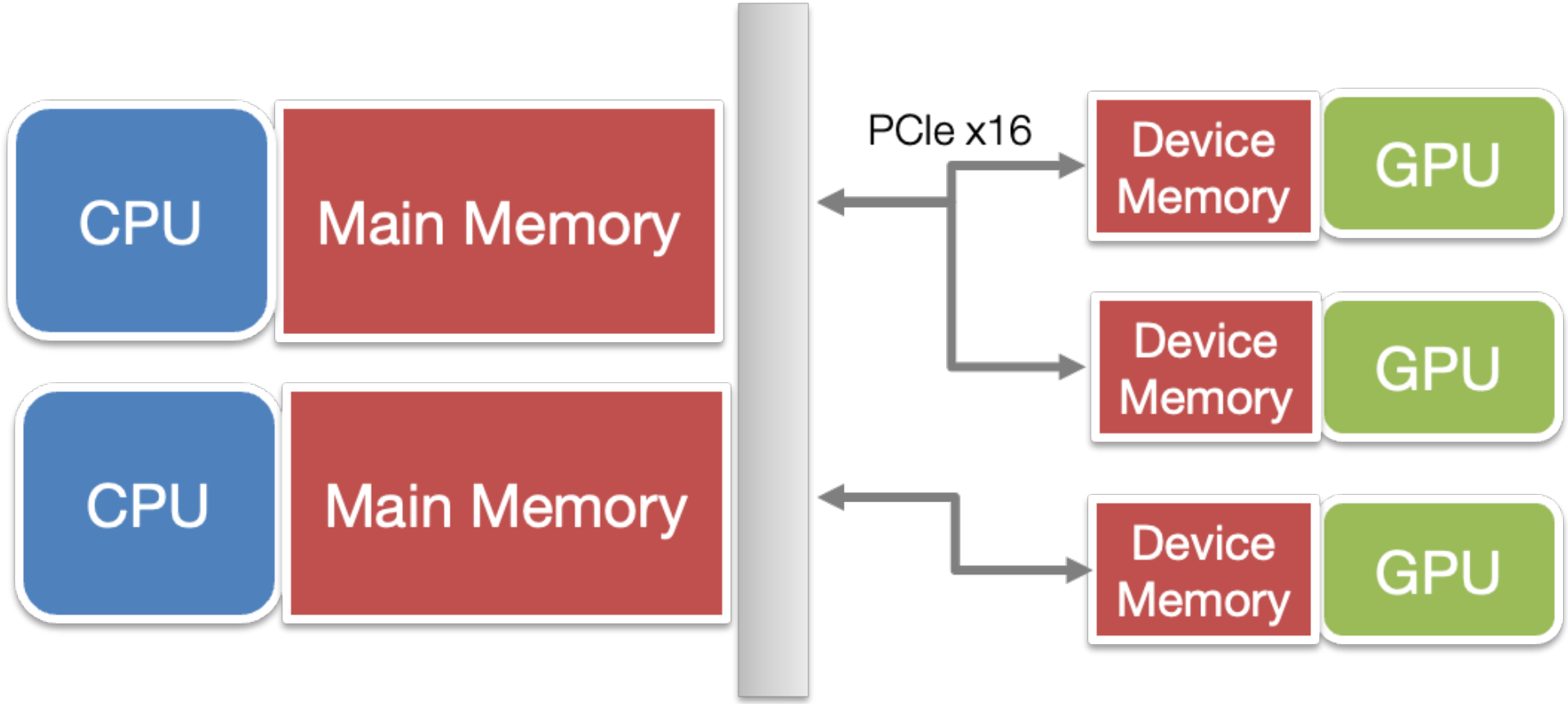
Heterogeneous Systems

GPU acceleration often follows an **offloading model.**

- Increased speed can be achieved by porting computationally intensive parts of code to GPU(s)



Heterogeneous Systems



Data needs to be copied from CPU to GPU, computation is performed 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.

This should be avoided!

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

Factors Affecting GPU Speedup

The performance increases from GPUs depend on several factors related to the data and algorithm.

Alpine GPU Partitions

Try these commands.

```
1 ssh <username>@login.rc.colorado.edu
2 sinfo --Format Partition
3 sinfo --partition aa100,ami100,atesting_a100,atesting_mi100
      --Format Partition,Nodes,Time,
4 scontrol show partition aa100
5 scontrol show partition atesting_a100
6 scontrol show partition ami100
7 scontrol show partition atesting_mi100
8 scontrol show node c3gpu-c2-u17
9 scontrol show node c3gpu-a9-u29-1
```

Requesting Alpine GPUs with Slurm

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

```
--partition=ami100
--gres=gpu:2
--ntasks=20
```

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

```
#SBATCH --partition=ami100
#SBATCH --gres=gpu:2
#SBATCH --ntasks=20
#SBATCH --job-name=gpu_test
#SBATCH --output=gpu_test_%j.out
#SBATCH --error=gpu_test_%j.err
#SBATCH --mail-type=ALL
#SBATCH --mail-user=<email>
```

in an interactive job

```
sinteractive --partition=ami100 --gres=gpu:2 --ntasks=20
```

Requesting Alpine GPUs with Slurm

```
#request one AMD GPU with default time and default CPU cores
sinteractive --partition=ami100 --gres=gpu:1

#request two AMD GPUs with 64 CPU cores for 24 hours
sinteractive --partition=ami100 --gres=gpu:2 --ntasks=64 --time=24:00:00

#request one AMD GPU for 7 days (requires long QoS)
sinteractive --partition=ami100 --gres=gpu:1 --time=7-00:00:00 --qos=long

#request three 80GB NVIDIA A100 GPUs with 20 CPU cores for default time
sinteractive --partition=aa100 --ntasks=20 --gres=gpu:3 --constraint=gpu80
```

Pay attention to defaults!

Alpine GPU Partitions- Defaults and Limits

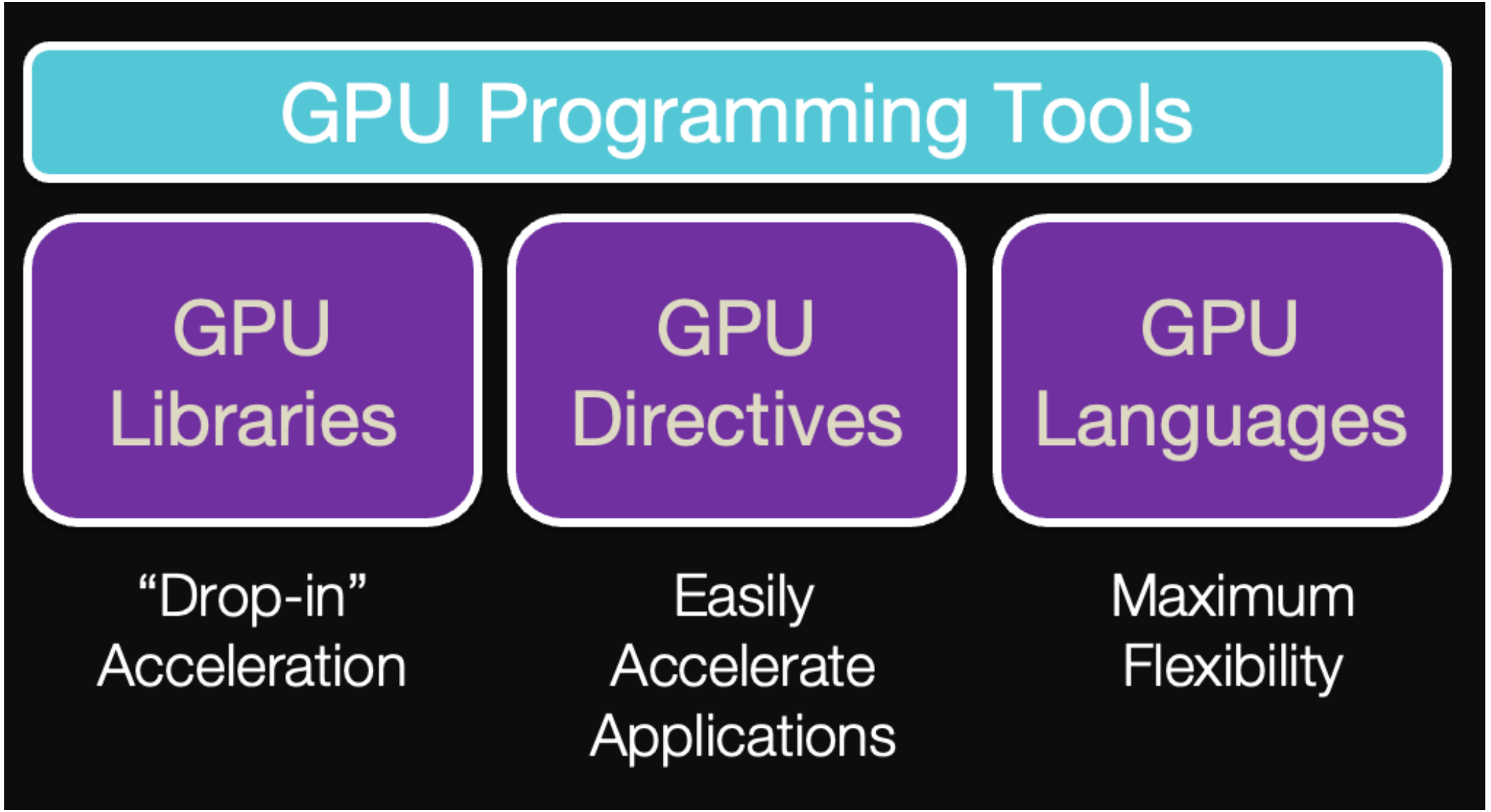
```
sinteractive --partition=<ami100 or aa100> --gres=gpu
```

	Default	Minimum	Maximum
ntasks	1	1	64 per node
GPU cards	1 GPU (- -gres=gpu)	1 GPU per node	Users are limited to their jobs collectively using up to 2/3 of the total GPUs in the partition. Once all running jobs exceed 2/3 of total GPUs, jobs will be queued with the reason QOSMaxGRESPerUser and will eventually run.
Nodes	1	1	12 for --partition=aa100, but see above 8 for --partition=ami100, but see above
Time	12:00:00	time > 0:00	24:00:00 with --qos=normal 7-00:00:00 with --qos=long

Alpine GPU Testing Partitions- Defaults and Limits

```
sinteractive --partition=<atesting_a100 or atesting_mi100> --gres=gpu
```

	Default	Minimum	Maximum
ntasks	1	1	16
GPU cards	0 without --gres=gpu	1	3
Nodes	1	1	1
Time	1:00:00	time > 0:00	1:00:00



Adapted from UL HPC Tutorials

GPU Programming Tools

Definitions

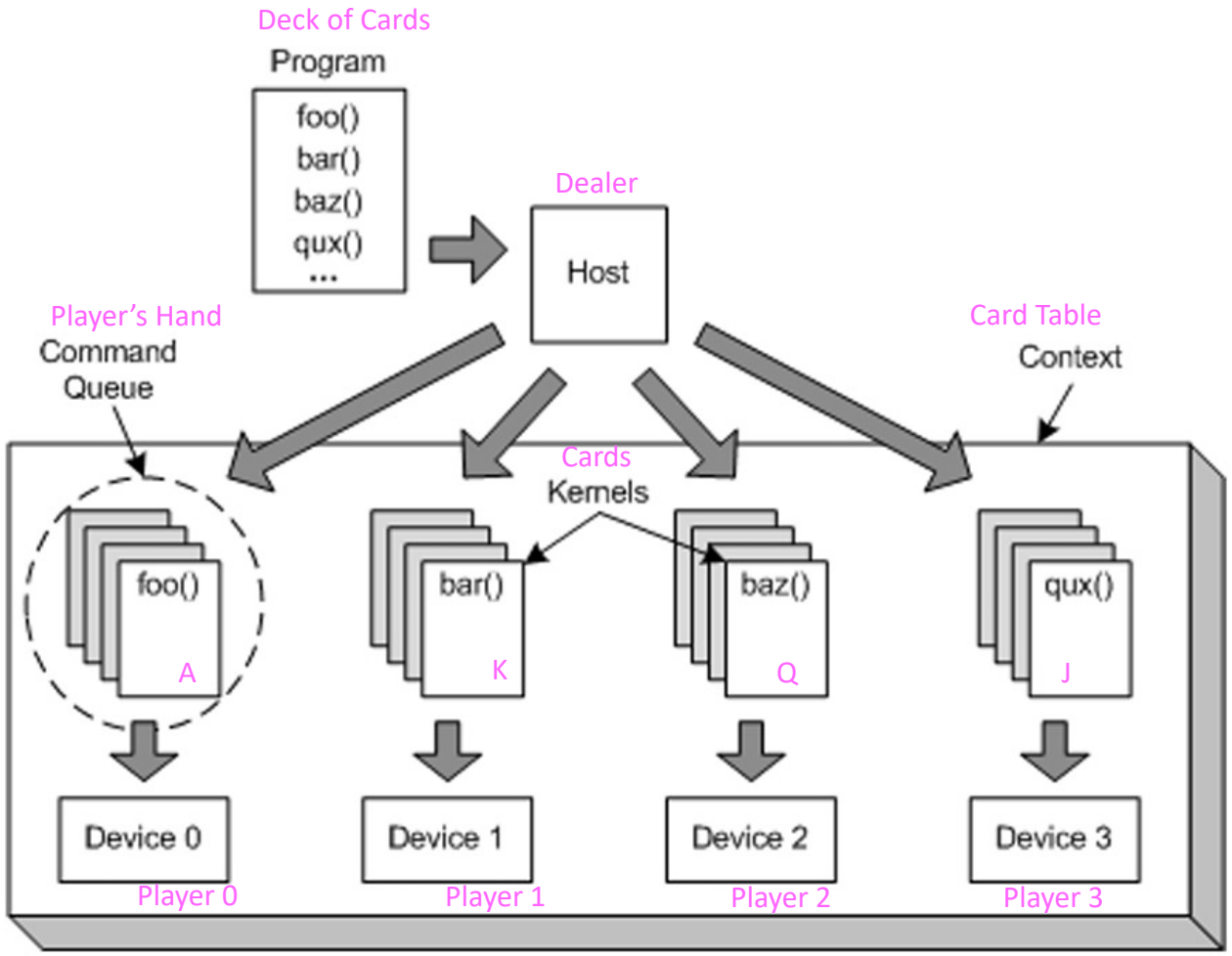
- **Host**- CPU
- **Device**- GPU
- **Kernel**- functions launched to the GPU

GPU Programming Tools

GPU Languages (Language Extensions)

- OpenCL (NVIDIA & AMD GPUs), HIP (NVIDIA & AMD), & CUDA (NVIDIA only)
- Require in-depth knowledge of hardware
- May involve substantial changes to code
- Code must be maintained to keep up with hardware changes & performance guidelines

GPU Programming Tools



Dealer distributes cards to players.
Host distributes kernels to devices.

Player receives cards from dealer.
Device receives kernels from host.

Dealer selects cards from a deck.
Host selects kernels from a program

Each player receives cards as part of a hand.
Each device receives kernels through the command queue.

Card table makes it possible for players to transfer cards to each other.
OpenCL Context allows devices to receive kernels and transfer data.

OpenCL Introduction, S Grauer-Gray

	Description	Code Example — OpenCL Introduction, S Grauer-Gray
1	Obtain OpenCL platform	`clGetPlatformIDs(1, &platform, NULL)`
2	Obtain device id for at least one device (accelerator)	`clGetDeviceIds(platform, CL_DEVICE_TYPE_GPU, 1, &device, NULL)`
3	Create context for device	`context = clCreateContext(NULL, 1, &device, NULL, NULL, &err)`
4	Create accelerator program from source code	`program = clCreateProgramWithSource (context, 1, (const char**) &program_buffer, &program_size, &err)`
5	Build the program	`clBuildProgram(program, 0,..)`
6	Create kernel(s) from program functions	`kernel = clCreateKernel(program, "kernel_name", &err)`
7	Create command queue for target device	`queue = clCreateCommandQueue(context, device, 0, &err)`
8	Allocate device memory / move input data to device memory	`memObject = clCreateBuffer (context, NULL, SIZE_N, NULL, &err)` `clEnqueueWriteBuffer(command_queue, memObject, ..., TOTAL_SIZE, hostPointer, ...)`
9	Associate arguments to kernel with kernel object	`cl_int clSetKernelArg (kernel, arg_index, arg_size, *arg_value)`

10	Deploy kernel for device execution	`global_size = TOTAL_NUM_THREADS;`
		`local_size = WORKGROUP_SIZE;`
		`clEnqueueNDRangeKernel(command_queue, kernel, 1, NULL, &global_size, &local_size, 0, NULL, NULL);`

11	Move output data to host memory	`clEnqueueReadBuffer(command_queue, memObject, blocking_read, offset, TOTAL_SIZE, hostPointer, 0, NULL, NULL)`
12	Release context/program/kernels/memory	`clReleaseMemObject(memObject) / clReleaseKernel(kernel) / clReleaseProgram(program) / clReleaseContext(context)`

GPU Programming Tools

GPU Compiler Directives

- Easier to learn and involve fewer changes to code than GPU programming languages
- Better portability among devices and platforms
- Require little understanding of GPU hardware, but more than GPU libraries
- OpenACC is the most commonly used

GPU Programming Tools

GPU Compiler Directives

- A collection of compiler directives that specify loops and regions of code in standard C, C++, and Fortran to be offloaded from a host CPU to an attached parallel accelerator (developer.nvidia.com)
- Lets compiler guess data allocation and movement or control it with directive clauses
- Can be used with GPU libraries, OpenMP

GPU Programming Tools

OpenACC

Basic program structure

```
#include "openacc.h"
#pragma acc <directive> [clauses [[,] clause]…] new-line
<code>
```

Kernel directives tell the compiler to generate parallel accelerator kernels for the loop nests following the directive.

Data directives tell the compiler to create code that performs specific data movements and provides hints about data usage.

Compile C code for NVIDIA GPU

```
pgcc –acc –ta=nvidia –c your_program_acc.c
```

Compile C++ code for NVIDIA GPU

```
nvcc --acc -Minfo=accel your_program_acc.c
```

GPU Programming Tools

OpenACC

Kernel directives tell the compiler to generate parallel accelerator kernels for the loop nests 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 tell the compiler to create code that performs specific data movements and provides hints about data usage.

```
#pragma acc data copy(a)
{
        #pragma acc kernels
         {
             for(int i = 0;
i <  n; i++)
             {
                  a[i] = 0.0
             }
         }
}
```

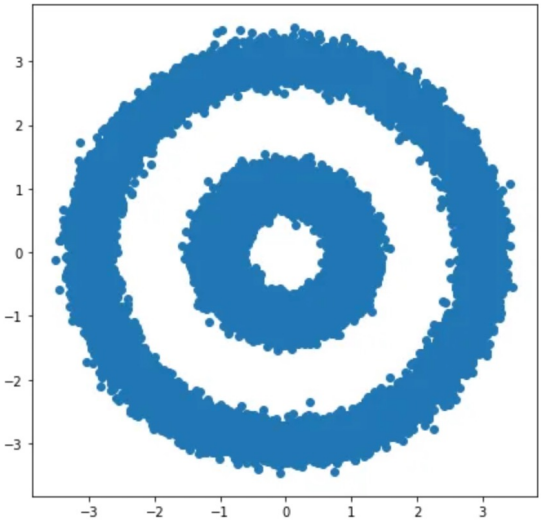
GPU Programming Tools

GPU Libraries

- Require little understanding of GPU hardware
- Usually involve minor changes to code
- Accelerated versions of many scientific libraries are available, e.g.
  - LAPACK → MAGMA, libFLAME
  - BLAS → cuBLAS, clBLAS
  - NumPy & SciPy → cuPy

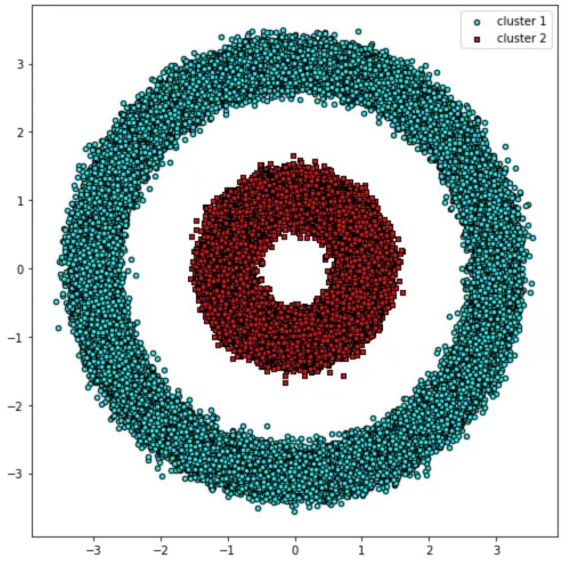
DBSCAN on CPU

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

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



DBSCAN with RAPIDS on GPU

```
#convert data to pandas.DataFrame then create cudf.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)

#use GPU-accelerated version of DBSCAN from cuML
from cuml import DBSCAN as cuml
DBSCANdb_gpu = cumlDBSCAN(eps=0.6, min_samples=2)
y_db_gpu = db_gpu.fit_predict(X_gpu)
```



Result of running DBSCAN on the CPU using Scikit-Learn

GPU Programming Tools

GPU Frameworks

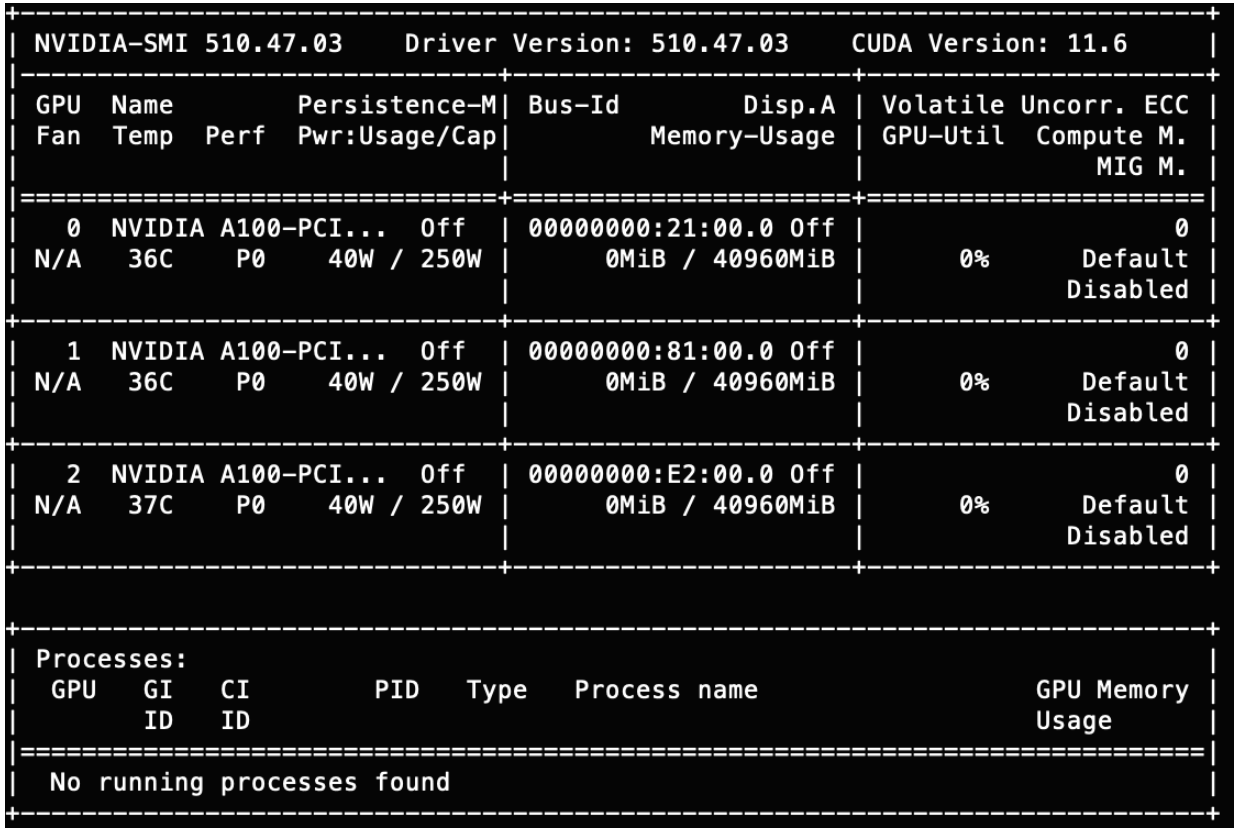
- Offer building blocks for designing, training, and validating workflows (like deep learning)
  - e.g. PyTorch, TensorFlow, Keras
- Rely on GPU-accelerated libraries

GPU Programming Tools

GPU programming tools vary in terms of ease of implementation, portability, and flexibility.

GPU Monitoring Tools

- `nvidia-smi` for NVIDIA GPUs
  - Command-line utility tool for monitoring NVIDIA GPUs
  - Returns device- and process-level information
  - Available on Alpine Nvidia nodes without loading any modules



GPU Monitoring Tools

- `rocm-smi` for AMD GPUs
  - Command-line utility tool for monitoring AMD GPUs
  - Returns extensive information (use `rocm-smi --help`)
  - Available on Alpine AMD nodes without loading any modules

```
======================== ROCm System Management Interface ========================
================================== Concise Info ==================================
GPU  Temp   AvgPwr  SCLK    MCLK     Fan  Perf  PwrCap  VRAM%  GPU%
0    33.0c  39.0W   300Mhz  1200Mhz  0%   auto  290.0W    0%   0%
1    35.0c  41.0W   300Mhz  1200Mhz  0%   auto  290.0W    0%   0%
2    34.0c  35.0W   300Mhz  1200Mhz  0%   auto  290.0W    0%   0%
==================================================================================
WARNING:                    One or more commands failed
============================== End of ROCm SMI Log ===============================
```

Basic GPU Application Troubleshooting

- 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? *Default is 1 CPU and ~3.7 G!*
- Did you load the correct version of ROCm or CUDA?
- Is your application utilizing the GPU during the accelerated part of the code?

Questions?

Please provide feedback on this training!

http://tinyurl.com/curc-survey18

The Alpine supercomputer is funded by contributions from the University of Colorado Boulder, the University of Colorado Anschutz, Colorado State University, and the National Science Foundation.