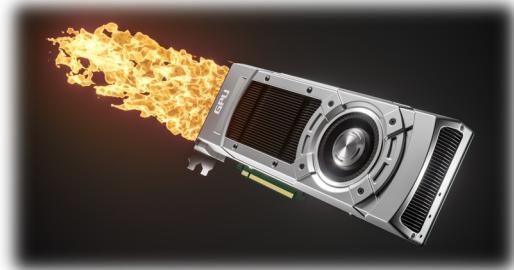


Introduction to GPU Acceleration on Alpine

August 2, 2022
2022 RMACC HPC Symposium

Layla Freeborn
layla.freeborn@colorado.edu



Research Computing at CU Boulder

Computing and Data Beyond the Desktop

- large-scale computing resources
- storage of research data
- high-speed data transfer
- data sharing support
- consultation in computational science and data management

Main Page:
colorado.edu/rc

Ask for Help:
colorado.edu/rc/userservices/contact



RMACC Cyber Infrastructure

<https://ask.cyberinfrastructure.org/c/rmacc/65>

- Provides opportunity for RMACC members to converse amongst themselves and with the larger research computing community
- The “go to” general Q&A platform for the global research computing community - researchers, facilitators, research software engineers, CI engineers, sys admins and others.



The Slido logo, consisting of the word "slido" in a lowercase, sans-serif font.

Join at
slido.com
#AlpineGPU



University of Colorado
Boulder | Colorado Springs | Denver | Anschutz Medical Campus

Overview

- Intro to Alpine's Heterogeneous Architecture
- Criteria for GPU Acceleration
- Factors Affecting GPU Speedup
- GPU Programming Tools
- Requesting GPUs on Alpine with Slurm
- GPU Monitoring on Alpine
- Hands-on Example: matmul with TensorFlow



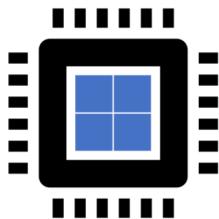
Overview

- **Intro to Alpine's Heterogeneous Architecture**
- Criteria for GPU Acceleration
- Factors Affecting GPU Speedup
- GPU Programming Tools
- Requesting GPUs on Alpine with Slurm
- GPU Monitoring on Alpine
- Hands-on Example: matmul with TensorFlow



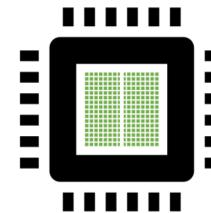
CPUs vs GPUs

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

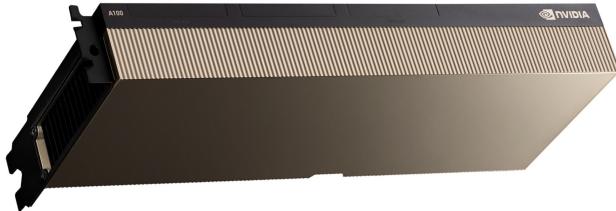


Alpine's Heterogeneous Architecture

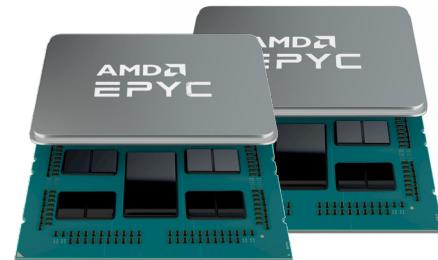
A100
GPU



MI100
GPU



Milan
CPU



Phase 1

8 nodes with 3 GPUs per node

8 nodes with 3 GPUs per node

64 nodes with 64 cores per node



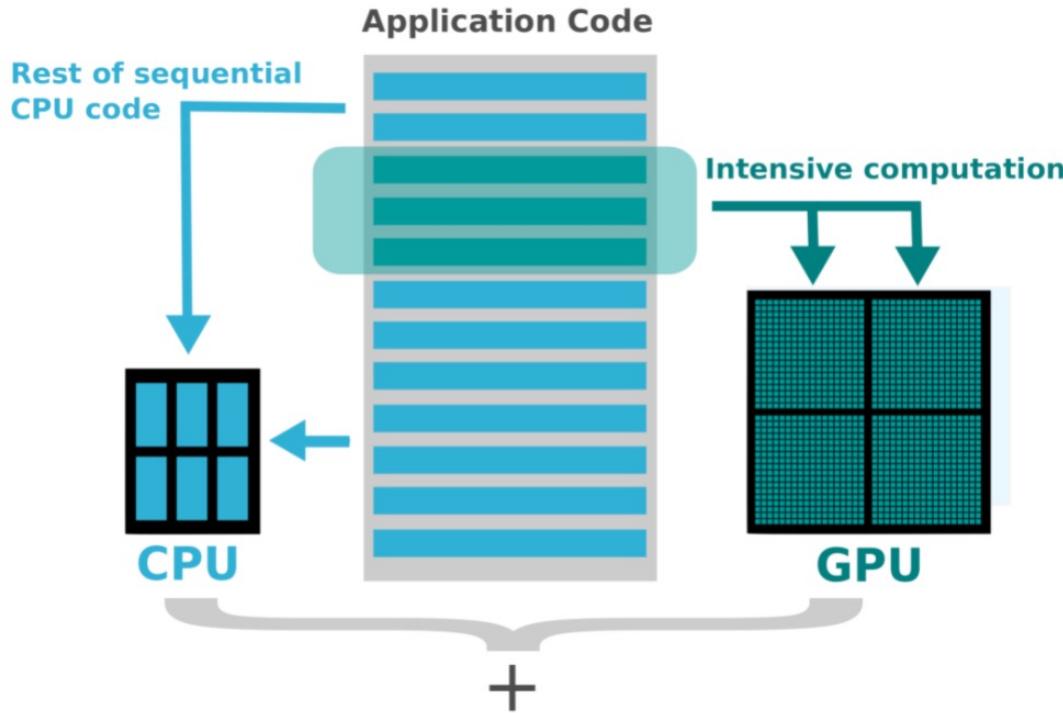
University of Colorado

Boulder | Colorado Springs | Denver | Anschutz Medical Campus

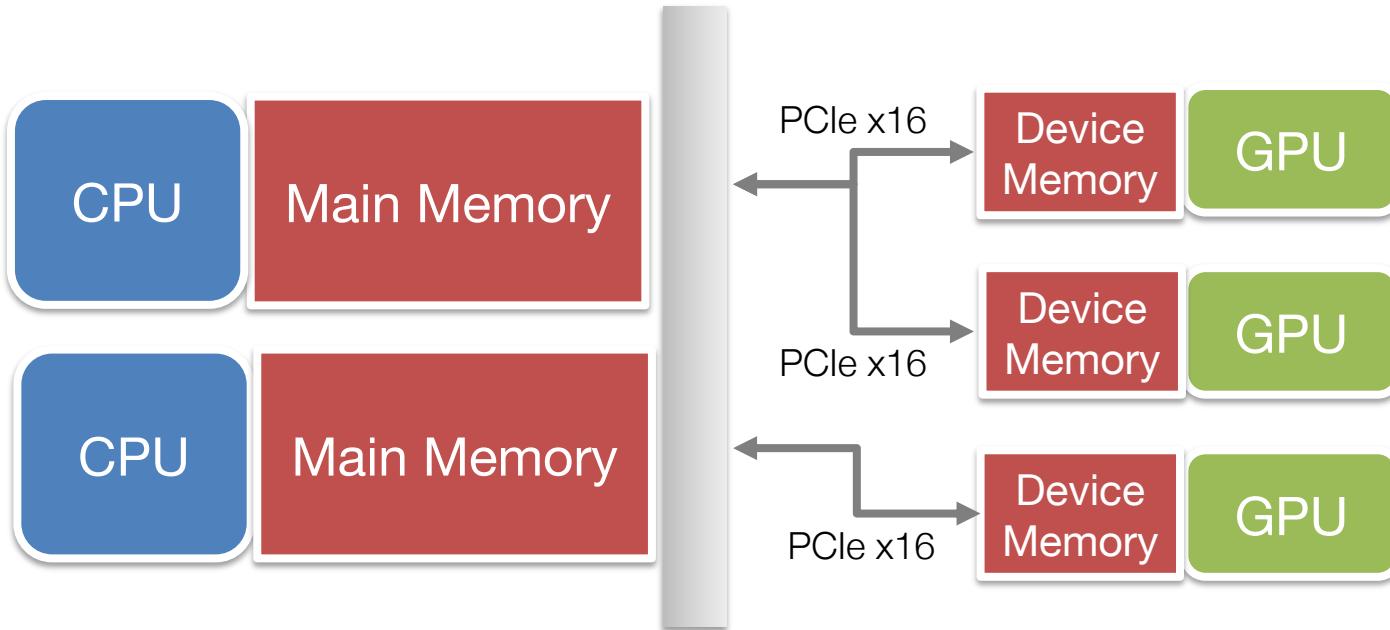
GPU-Accelerated Code

Offloading model

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



GPU-Accelerated Code



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



Alpine's Heterogeneous Architecture

Want to learn more about the design of Alpine
and the roadmap to scaling it up?

Welcome to Alpine

Craig Earley

~tomorrow~

10-10:45, Room 204



University of Colorado
Boulder | Colorado Springs | Denver | Anschutz Medical Campus

Overview

- Intro to Alpine's Heterogeneous Architecture
- **Criteria for GPU Acceleration**
- Factors Affecting GPU Speedup
- GPU Programming Tools
- Requesting GPUs on Alpine with Slurm
- GPU Monitoring on Alpine
- Hands-on Example: matmul with TensorFlow



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

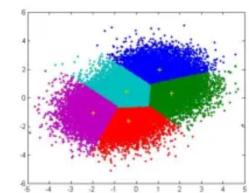


Common GPU-Accelerated Workflows

- Floating-point operations
- Operations that involve running the exact same code on a huge number of different data points at once.

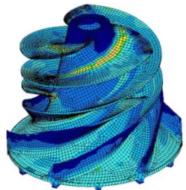


Common GPU-Accelerated Workflows



1. Dense Linear Algebra

vector and matrix operations, e.g.
clustering algorithms, K-means



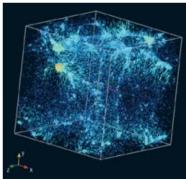
2. Sparse Linear Algebra

matrix multiplication matrices
composed mostly of 0s, e.g.
finite element analysis



3. Spectral Methods

solving differential equations, e.g.
fluid dynamics, quantum mechanics,
weather prediction



4. N-Body Methods

simulation of dynamical system of
particles, e.g. astronomy,
computational chemistry, physics



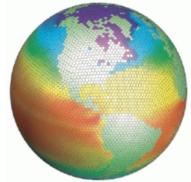
University of Colorado
Boulder | Colorado Springs | Denver | Anschutz Medical Campus

GPU Acceleration- 8/2/22

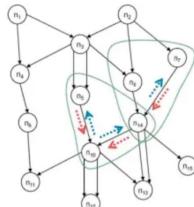
15

Shamelessly borrowed from
StreamHPC.com

Common GPU-Accelerated Workflows

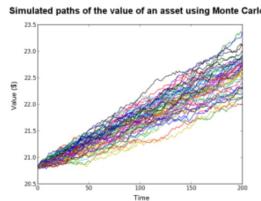


5. Structured Grids



computations that depend on neighbors in an irregular grid, e.g. image processing, physics simulations

6. Unstructured Grids



Grids with different elements that have a different number of neighbors, e.g. computational fluid dynamics

7. Map-Reduce & Monte Carlo

0111 1001 0011 0101 0101 0001 1001 1101 0011 1101 1010 0111 0001 0010 0001 1001
0111 0010 0001 0100 1000 1100 1001 0101 0011 0011 0101 1101 1101 0011 0001 0010 0001 1001
0001 0101 0100 0011 0111 0100 1001 0110 0010 0000 0101 0011 1001 1011 1000 0100 0001 1001
1011 1011 1101 1010 1000 1001 1011 1100 1101 0010 0011 1000 1001 0010 0000 0101 0011 1001
1101 1011 1101 1010 1000 1001 1011 1100 1101 0010 0011 1000 1001 0010 0000 0101 0011 1001
1000 0101 0100 1010 1100 0010 0001 1100 0011 1000 1001 1010 0010 0000 0101 0010 0000 0101 0010
0100 1000 1010 1011 1000 0111 0010 1111 1010 1110 0001 1110 0001 1110 0001 0010 0000 0101 0010
0111 1100 0100 0111 0000 0011 0111 1001 1000 1101 1001 1111 1110 0001 0011 1110 0001 0010 0000

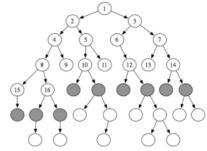
Each process runs completely independent from one another, e.g. Monte-Carlo, Distributed Searching

8. Combinational Logic

Usually involve performing simple operations on large amounts of data, e.g. hashing, encryption, checksums

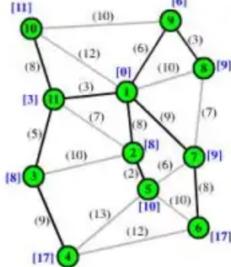


Common GPU-Accelerated Workflows



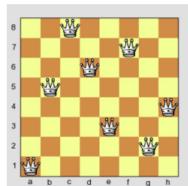
9. Graph Traversal

Problem of visiting all the nodes in a graph, updating and checking values along the way, e.g. search, sorting, serialization



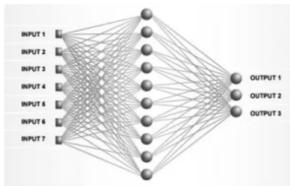
10. Dynamic Programming

Algorithmic technique to compute solutions by solving overlapping subproblems, e.g. graph problems, sequence alignment



11. Backtracking

Build all possible solutions, usually branch-and-bound solution approach, e.g. puzzles, traveling salesman



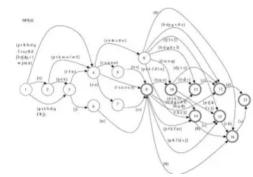
12. Probabilistic Graphical Models

e.g. Bayesian networks, hidden Markov models, neural networks



University of Colorado
Boulder | Colorado Springs | Denver | Anschutz Medical Campus

Common GPU-Accelerated Workflows



13. Finite State Machines

Mathematical model of computation used to design computer programs and sequential logic circuits, e.g. data mining, video compression

Further Reading

- Geng et al. OpenCL and the 13 dwarfs: a work in progress
- Asanovic et al. The landscape of parallel computing research: A view from Berkeley
- Vincent Hindriksen's blog post [The 13 application areas where OpenCL and CUDA can be used](#)





Join at

slido.com
#AlpineGPU



Which statement best describes your experience with GPU Acceleration?

- I've run GPU-accelerated applications before.
- I don't have experience with GPU Acceleration but I hope to incorporate it into my workflows.
- Neither of the above. I'm curious about how GPU Acceleration works on Alpine.



University of Colorado
Boulder | Colorado Springs | Denver | Anschutz Medical Campus

Overview

- Intro to Alpine's Heterogeneous Architecture
- Criteria for GPU Acceleration
- **Factors Affecting GPU Speedup**
- GPU Programming Tools
- Requesting GPUs on Alpine with Slurm
- GPU Monitoring on Alpine
- Hands-on Example: matmul with TensorFlow



Factors Affecting GPU Speedup

1 Data Volume

2 Data Dependency

3 Data Processing Order

4 Data Type

5 Code/Algorithmic Complexity

6 Computational Intensity

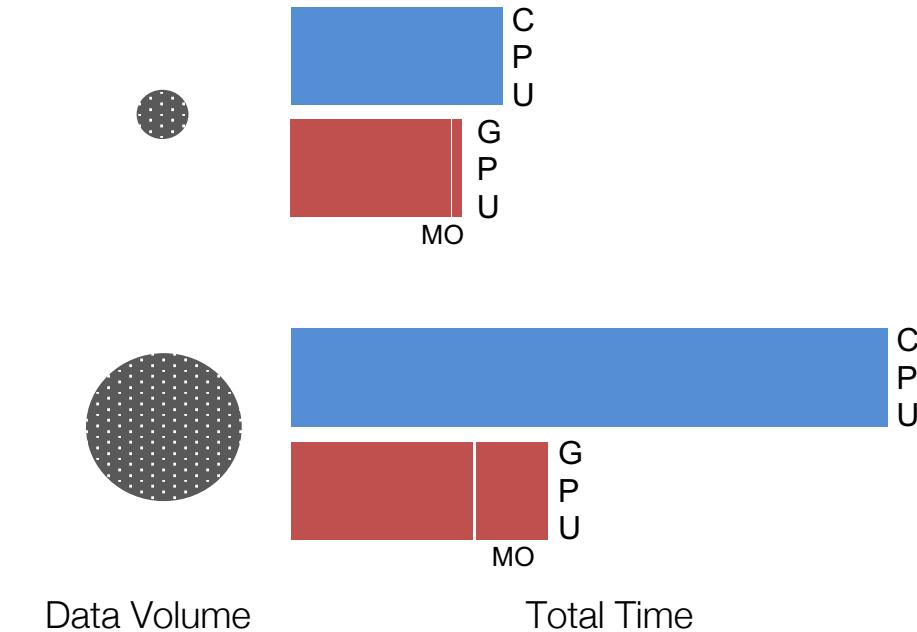


Factors Affecting GPU Speedup

1

Data Volume

- The time spent transferring data to/from CPU to GPU is worthwhile for high data volumes



University of Colorado
Boulder | Colorado Springs | Denver | Anschutz Medical Campus

Factors Affecting GPU Speedup

2

Data Dependency

- Processing a data point depends on previous data points.
 - This should be avoided!

```
for (i=1; i<1000; i++){  
    sum = sum + a[i]; /* S1 */  
}
```

```
for (i=1; i<1000; i++){  
    a[i] = 2 + a[i]; /* S1 */  
}
```



Factors Affecting GPU Speedup

3

Data Processing Order

- Data should be written and read in a continuous order for maximum utilization of a GPU.
 - May require adjustments to algorithm



University of Colorado
Boulder | Colorado Springs | Denver | Anschutz Medical Campus

Factors Affecting GPU Speedup

4

Data Type

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



University of Colorado
Boulder | Colorado Springs | Denver | Anschutz Medical Campus

Factors Affecting GPU Speedup

5

Code/Algorithmic Complexity

- ‘Simple’ code is better ported to GPUs
 - Deeply branched code and while-loops perform poorly on GPUs
 - Recursive functions need rewriting



Factors Affecting GPU Speedup

6

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



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



Get Started on Alpine

Get your username and password

Log in:

```
ssh <username>@tlogin1.rc.colorado.edu
```

Load the Alpine Slurm module:

```
module load slurm/alpine
```



University of Colorado

Boulder | Colorado Springs | Denver | Anschutz Medical Campus

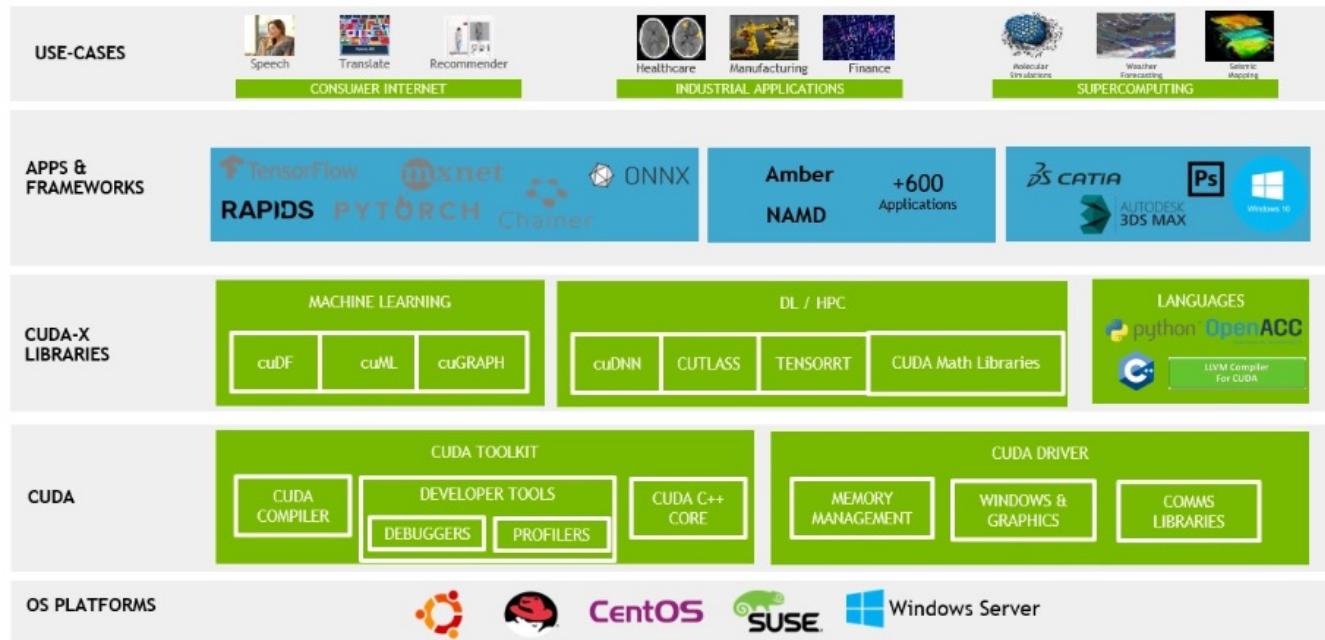
Overview

- Intro to Alpine's Heterogeneous Architecture
- Criteria for GPU Acceleration
- Factors Affecting GPU Speedup
- **GPU Programming Tools**
- Requesting GPUs on Alpine with Slurm
- GPU Monitoring on Alpine
- Hands-on Example: matmul with TensorFlow



GPU Programming Tools

CUDA - a parallel computing platform and programming model



GPU Programming Tools

More
than
CUDA!

GPU Programming Tools

GPU
Libraries

GPU
Directives

GPU
Languages

“Drop-in”
Acceleration

Easily
Accelerate
Applications

Maximum
Flexibility



University of Colorado
Boulder | Colorado Springs | Denver | Anschutz Medical Campus

GPU Acceleration- 8/2/22

32

Adapted from UL HPC
Tutorials

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, cIBLAS
 - NumPy & SciPy → cuPy

GPU Programming Tools

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)
```



University of Colorado
Boulder | Colorado Springs | Denver | Anschutz Medical Campus

GPU Directives

- Easier to learn and involve fewer changes to code than GPU programming languages
- Better portability among devices and platforms
- Compiler directives, e.g. OpenACC, are the most commonly used



GPU Programming Tools

GPU Directives

- Require little understanding of GPU hardware, but more than GPU libraries
 - Host- CPU
 - Device- GPU
 - Kernel- functions launched to the GPU



OpenACC

- 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)
- Let compiler guess data allocation and movement or control it with directive clauses
- Can be used with GPU libraries, OpenMP

Basic Program Structure (C/C++)

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

Compiling (C/C++) for NVIDIA GPU

```
nvcc --acc -Minfo=accel your_program_acc.c  
pgcc -acc -ta=nvidia -c your_program_acc.c
```



GPU Programming Tools

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

Example Kernel Directive (C/C++)

```
//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.

Example Data Directive (C/C++)

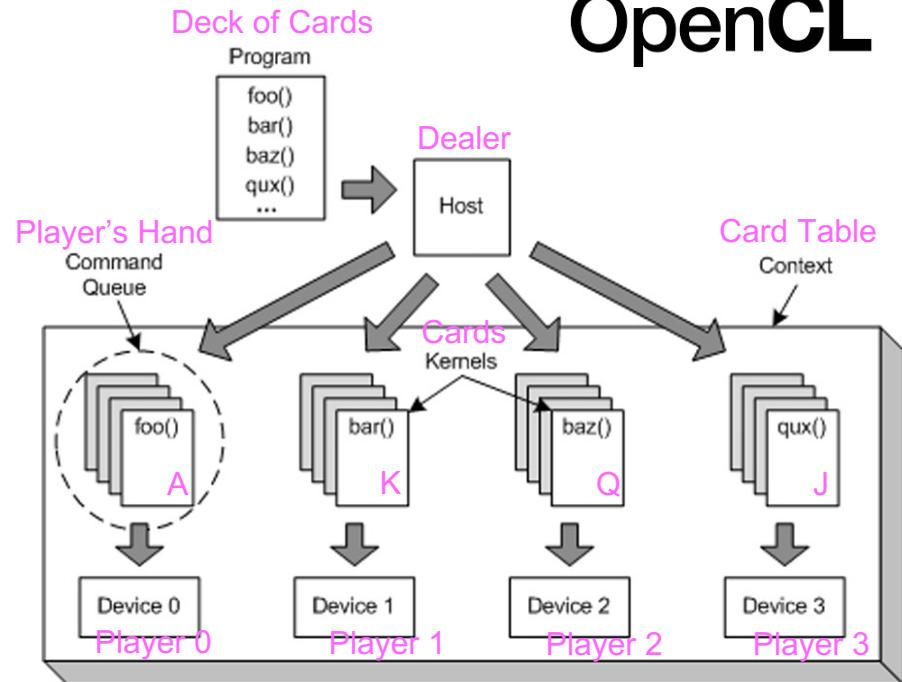
```
#pragma acc data copy(a)
{
    #pragma acc kernels
    {
        for(int i = 0; i < n; i++)
        {
            a[i] = 0.0
        }
    }
}
```



GPU Languages (Language Extensions)

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

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.



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 Frameworks

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



University of Colorado
Boulder | Colorado Springs | Denver | Anschutz Medical Campus

GPU Programming Tools

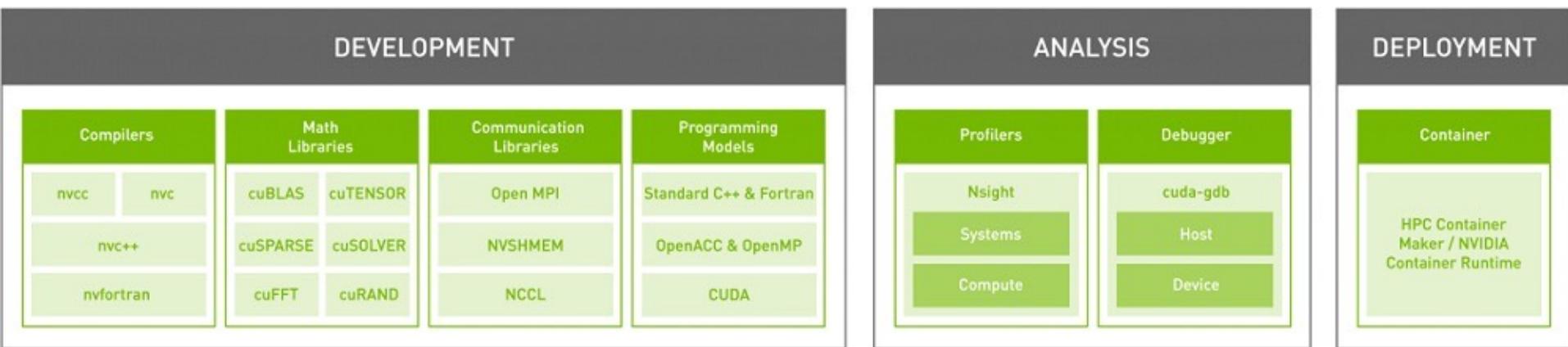


GPU programming tools vary in terms of ease of implementation, portability, and flexibility.



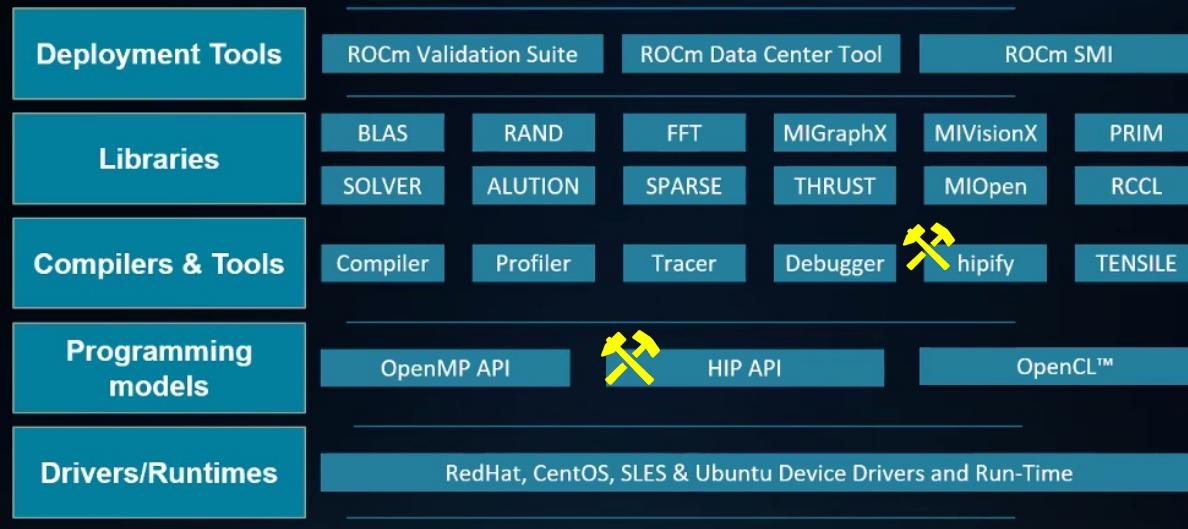
GPU Programming Tools

NVIDIA SDK



GPU Programming Tools

AMD ROCm™ - The Core Components



The Heterogeneous Computing Interface for Portability (HIP) is a vendor-neutral C++ programming model for implementing highly tuned workload for GPUs. HIP (like CUDA) is a dialect of C++ supporting templates, classes, lambdas, and other C++ constructs.

A “hipify” tool is provided to ease conversion of CUDA codes to HIP, enabling code compilation for either AMD or NVIDIA GPU (CUDA) environments. The ROCm™ HIP compiler is based on Clang, the LLVM compiler infrastructure, and the “libc++” C++ standard library.





Join at

slido.com

#AlpineGPU



University of Colorado

Boulder | Colorado Springs | Denver | Anschutz Medical Campus

GPU Programming Tools

GPU
Libraries

"Drop-in"
Acceleration

GPU
Directives

Easily
Accelerate
Applications

GPU
Languages

Maximum
Flexibility

Rank the GPU programming tools by how likely you are to use them, from most (1) to least likely (6).

Overview

- Intro to Alpine's Heterogeneous Architecture
- Criteria for GPU Acceleration
- Factors Affecting GPU Speedup
- GPU Programming Tools
- **Requesting GPUs on Alpine with Slurm**
- GPU Monitoring on Alpine
- Hands-on Example: matmul with TensorFlow



Requesting GPUs on Alpine with Slurm

Access Slurm job scheduler from login node

```
module load slurm/alpine
```

View Alpine GPU resources and configurations

```
sinfo --Format NodeList:30,Partition,Gres |grep gpu |grep -v "mi100|a100"
```

```
c3gpu-c2-u[1,3,5,7,9,11,13,15]aa100
c3gpu-c2-u[1,3,5,7,9,11,13,15]aa100-ucb
c3gpu-c2-u[1,3,5,7,9,11,13,15]aa100-csu
c3gpu-c2-u[1,3,5,7,9,11,13,15]aa100-amc
c3gpu-c2-u[17,19,21,23,25,27,2ami100
c3gpu-c2-u[17,19,21,23,25,27,2ami100-ucb
c3gpu-c2-u[17,19,21,23,25,27,2ami100-csu
c3gpu-c2-u[17,19,21,23,25,27,2ami100-amc
c3gpu-c2-u[17,19,21,23,25,27,2atesting
c3gpu-c2-u[1,3,5,7,9,11,13,15]atesting
```

```
gpu:a100:3
gpu:a100:3
gpu:a100:3
gpu:a100:3
gpu:a100:3
gpu:mi100:3
gpu:mi100:3
gpu:mi100:3
gpu:mi100:3
gpu:mi100:3
gpu:a100:3
```

Nvidia A100s
partition=aa100-<institution>
e.g. partition=aa100-ucb

AMD A100s
partition=ami100-<institution>
e.g. partition=ami100-ucb



University of Colorado

Boulder | Colorado Springs | Denver | Anschutz Medical Campus

Requesting GPUs on Alpine with Slurm

Request a compile node

```
acompile --help
```

```
acompile --ntasks=4 --gpu=amdgpu --time=2:00:00
```

```
acompile --ntasks=4 --gpu=nvidia --time=2:00:00
```

Start an interactive job

```
sinteractive --ntasks=64 --gres=gpu:3 --partition=aa100-ucb --time=1:00:00
```

```
sinteractive --ntasks=20 --gres=gpu:1 --partition=ami100-ucb
```

```
sinteractive --gres=gpu:1 --partition=ami100-ucb --qos=long --time=7-00:00:00
```

```
sinteractive --gres=gpu:1 --nodes=2 --partition=ami100-ucb
```



University of Colorado

Boulder | Colorado Springs | Denver | Anschutz Medical Campus

Overview

- Intro to Alpine's Heterogeneous Architecture
- Criteria for GPU Acceleration
- Factors Affecting GPU Speedup
- GPU Programming Tools
- Requesting GPUs on Alpine with Slurm
- **GPU Monitoring on Alpine**
- Hands-on Example: matmul with TensorFlow



GPU Monitoring & Performance Profiling Tools

nvidia-smi

- 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 & Performance Profiling Tools

NVIDIA-SMI 510.47.03									
		Driver Version: 510.47.03		CUDA Version: 11.6					
GPU Fan	Name	Persistence-M	Bus-Id	Disp.A	Memory-Usage	Volatile	Uncorr.	ECC	
						GPU-Util	Compute M.	MIG M.	
0 N/A	NVIDIA A100-PCI...	Off	00000000:21:00.0	Off	0MiB / 40960MiB	0%	Default	0	
1 N/A	NVIDIA A100-PCI...	Off	00000000:81:00.0	Off	0MiB / 40960MiB	0%	Default	0	
2 N/A	NVIDIA A100-PCI...	Off	00000000:E2:00.0	Off	0MiB / 40960MiB	0%	Default	0	

Processes:									
GPU ID	GI ID	CI ID	PID	Type	Process name	GPU Memory Usage			
No running processes found									



University of Colorado

Boulder | Colorado Springs | Denver | Anschutz Medical Campus

GPU Monitoring & Performance Profiling Tools

rocm-smi

- Command-line utility tool for monitoring AMD GPUs
- Returns extensive information
- Available on Alpine AMD nodes without loading any modules



GPU Monitoring & Performance Profiling Tools

```
===== 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 =====
```

More Information:

```
rocm-smi --help
```

Documentation available at

https://rocmdocs.amd.com/en/latest/ROCM_System_Management/ROCM-SMI-CLI.html



University of Colorado

Boulder | Colorado Springs | Denver | Anschutz Medical Campus



Join at
slido.com
#AlpineGPU

Join the quiz!

Overview

- Intro to Alpine's Heterogeneous Architecture
- Criteria for GPU Acceleration
- Factors Affecting GPU Speedup
- GPU Programming Tools
- Requesting GPUs on Alpine with Slurm
- GPU Monitoring on Alpine
- **Hands-on Example: matmul with TensorFlow**



GPU Speedup Example: Matrix Multiplication with Tensorflow

- A **matrix** is a rectangular array of numbers arranged in rows and columns.
- The **order** of a matrix is the number of rows and columns.

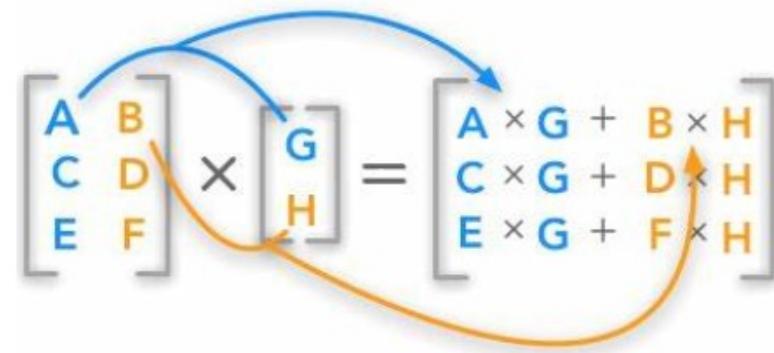
$$\begin{pmatrix} 6 & 2 & 4 \\ 2 & 1 & 6 \end{pmatrix}$$

order = 2x3



GPU Speedup Example: Matrix Multiplication with Tensorflow

Matrix multiplication is used
in many ML algorithms.



We will use Tensorflow's
`tf.matmul` function.



GPU Speedup Example: Matrix Multiplication with Tensorflow

tf.matmul-cpu.py

```
import os
import numpy as np
import tensorflow as tf
import time

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
os.environ['CUDA_VISIBLE_DEVICES'] = '-1'
if tf.test.gpu_device_name():
    print("GPU found")
else:
    print("Tensorflow is using only CPUs.")

numpy_start = time.time()
a = np.random.rand(10000,70000)
print("Numpy generated a random 10000x70000 matrix, matrix a")
b = np.random.rand(70000,10000)
print("Numpy generated a random 70000x10000 matrix, matrix b")
numpy_end = time.time()
print("It took numpy", numpy_end-numpy_start, "seconds to generate matrices a and b.")
print("Tensorflow is beginning matrix multiplication.")
tf_start = time.time()
tf.matmul(a,b)
tf_end=time.time()
print("Tensorflow multiplied matrices a and b in", tf_end-tf_start,"seconds.")
```

GPU Speedup Example: Matrix Multiplication with Tensorflow

tf.matmul-gpu.py

```
import os
import numpy as np
import tensorflow as tf
import time

os.environ['CUDA_VISIBLE_DEVICES'] = '0'
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

numpy_start = time.time()
a = np.random.rand(10000,70000)
print("Numpy generated a random 10000x70000 matrix, matrix a")
b = np.random.rand(70000,10000)
print("Numpy generated a random 70000x10000 matrix, matrix b")
numpy_end = time.time()
print("It took numpy", numpy_end-numpy_start, "seconds to generate matrices a and b.")
print("Tensorflow is beginning matrix multiplication.")
tf_start = time.time()
tf.matmul(a,b)
tf_end=time.time()
print("Tensorflow multiplied matrices a and b in", tf_end-tf_start,"seconds.")
```

GPU Speedup Example: Matrix Multiplication with Tensorflow

<https://bit.ly/AlpineGPU2>



GPU Speedup Example: Matrix Multiplication with Tensorflow

Run CPU-only script:

```
python tf.matmul-cpu.py
```

```
Tensorflow is using only CPUs.  
Numpy generated a random 10000x70000 matrix, matrix a.  
Numpy generated a random 70000x10000 matrix, matrix b.  
It took numpy 6.565369129180908 seconds to generate matrices a and b.  
Tensorflow is beginning matrix multiplication.  
Tensorflow multiplied matrices a and b in 45.82650423049927 seconds.
```

Run GPU-accelerated script:

```
python tf.matmul-gpu.py
```

```
Numpy generated a random 10000x70000 matrix, matrix a.  
Numpy generated a second random 70000x10000 matrix, matrix b.  
It took numpy 6.620205402374268 seconds to generate the matrices.  
Tensorflow is beginning matrix multiplication.  
Tensorflow multiplied matrices a and b in 3.0970964431762695 seconds.
```



University of Colorado

Boulder | Colorado Springs | Denver | Anschutz Medical Campus

GPU Speedup Example: Matrix Multiplication with Tensorflow

```
import os
import numpy as np
import tensorflow as tf
import time

os.environ["CUDA_VISIBLE_DEVICES"]="0"
os.environ["TF_CPP_MIN_LOG_LEVEL"]="3"

numpy_start=time.time()
a = np.random.rand(50000, 20000)
print("NumPy generated a random 50000x20000 matrix, matrix a.")
b = np.random.rand(20000, 50000)
numpy_end=time.time()

print("NumPy generated a second random 20000x50000 matrix, matrix b.")
print("It took numpy", numpy_end-numpy_start, "seconds to generate the matrices.")
print("Tensorflow is beginning matrix multiplication.")

tf_start = time.time()
c = tf.matmul(a,b)
tf_end = time.time()
print("Tensorflow multiplied matrices a and b in", tf_end-tf_start, "seconds.")
quit()
```

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%	Default	0
	N/A	36C	P0	40W / 250W	0MiB / 40960MiB			Disabled	
1	NVIDIA A100-PCI...	Off	00000000:81:00.0	Off			0%	Default	0
	N/A	36C	P0	40W / 250W	0MiB / 40960MiB			Disabled	
2	NVIDIA A100-PCI...	Off	00000000:E2:00.0	Off			0%	Default	0
	N/A	37C	P0	40W / 250W	0MiB / 40960MiB			Disabled	
Processes:									
GPU	GI	CI	PID	Type	Process name		GPU Memory Usage		
ID	ID								
No running processes found									



University of Colorado

Boulder | Colorado Springs | Denver | Anschutz Medical Campus

GPU Speedup Example: Matrix Multiplication with Tensorflow

```
import os
import numpy as np
import tensorflow as tf
import time

os.environ["CUDA_VISIBLE_DEVICES"]="0"
os.environ["TF_CPP_MIN_LOG_LEVEL"]="3"

numpy_start=time.time()
a = np.random.rand(50000, 20000)
print("Numpy generated a random 50000x20000 matrix, matrix a.")
b = np.random.rand(20000, 50000)
numpy_end=time.time()

print("Numpy generated a second random 20000x50000 matrix, matrix b.")
print("It took numpy", numpy_end-numpy_start, "seconds to generate the matrices.")
print("Tensorflow is beginning matrix multiplication.")
tf_start = time.time()
c = tf.matmul(a,b)
tf_end = time.time()
print("Tensorflow multiplied matrices a and b in", tf_end-tf_start, "seconds.")
quit()
```

NVIDIA-SMI 510.47.03 Driver Version: 510.47.03 CUDA Version: 11.6						
Fan	GPU Name	Persistence-M	Bus-Id	Disp.A	Volatile Uncorr.	ECC MIG M.
	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util	
0	NVIDIA A100-PCI...	Off	00000000:21:00.0	Off	0%	Default
N/A	36C	P0	40W / 250W	38819MiB / 40960MiB		Disabled
1	NVIDIA A100-PCI...	Off	00000000:81:00.0	Off	0%	Default
N/A	35C	P0	40W / 250W	0MiB / 40960MiB		Disabled
2	NVIDIA A100-PCI...	Off	00000000:E2:00.0	Off	0%	Default
N/A	37C	P0	40W / 250W	0MiB / 40960MiB		Disabled
Processes:						
GPU	GI	CI	PID	Type	Process name	GPU Memory Usage
ID	ID					
0	N/A	N/A	3447060	C	python	38817MiB



University of Colorado

Boulder | Colorado Springs | Denver | Anschutz Medical Campus

GPU Speedup Example: Matrix Multiplication with Tensorflow

- How could we speed this up even further?
- How would matrix size affect the observed speed up?

Thank you!

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

Alpine Documentation:

<https://curc.readthedocs.io/en/latest/clusters/alpine/index.html>

Ask for Help: colorado.edu/rc/userservices/contact

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.

