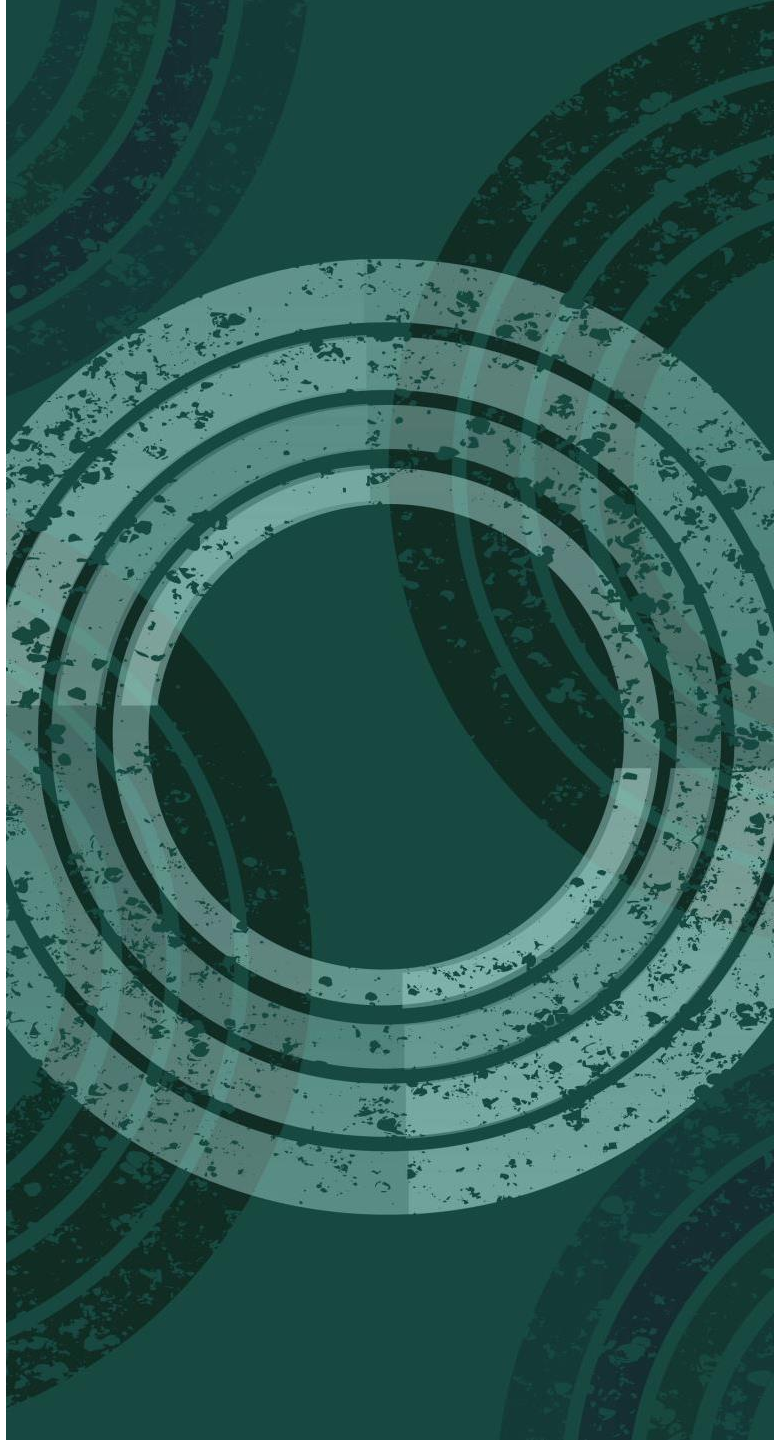
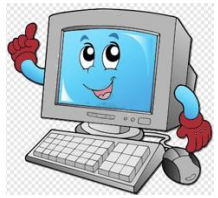
NVIDIA & AMD GPU CONTEXT ON ALPINE

By Kevin Fotso



Anatomy of supercomputing



Laptop or PC

Login node

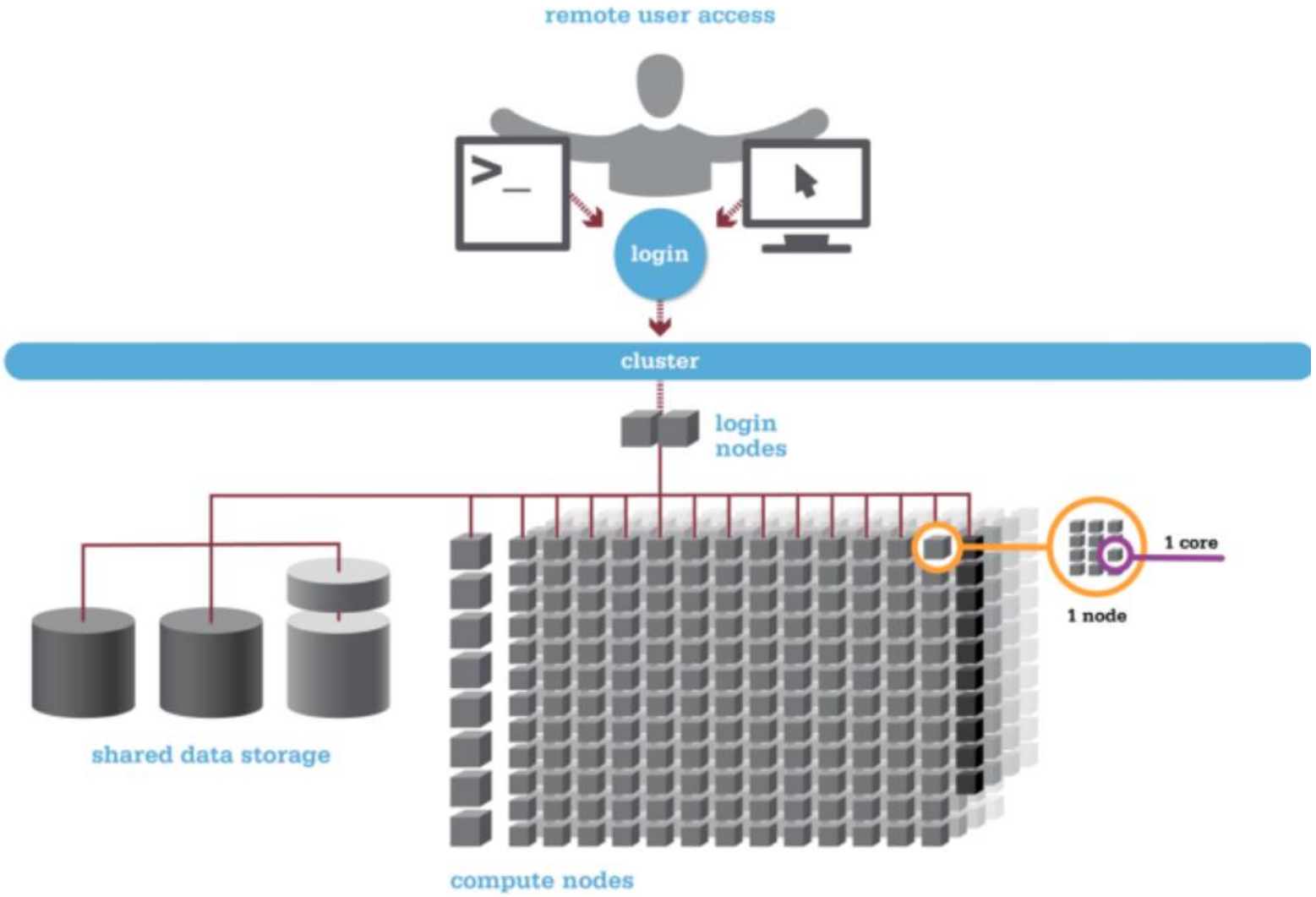
Slurm: scheduler

Compute node

Fast interconnect

Storage

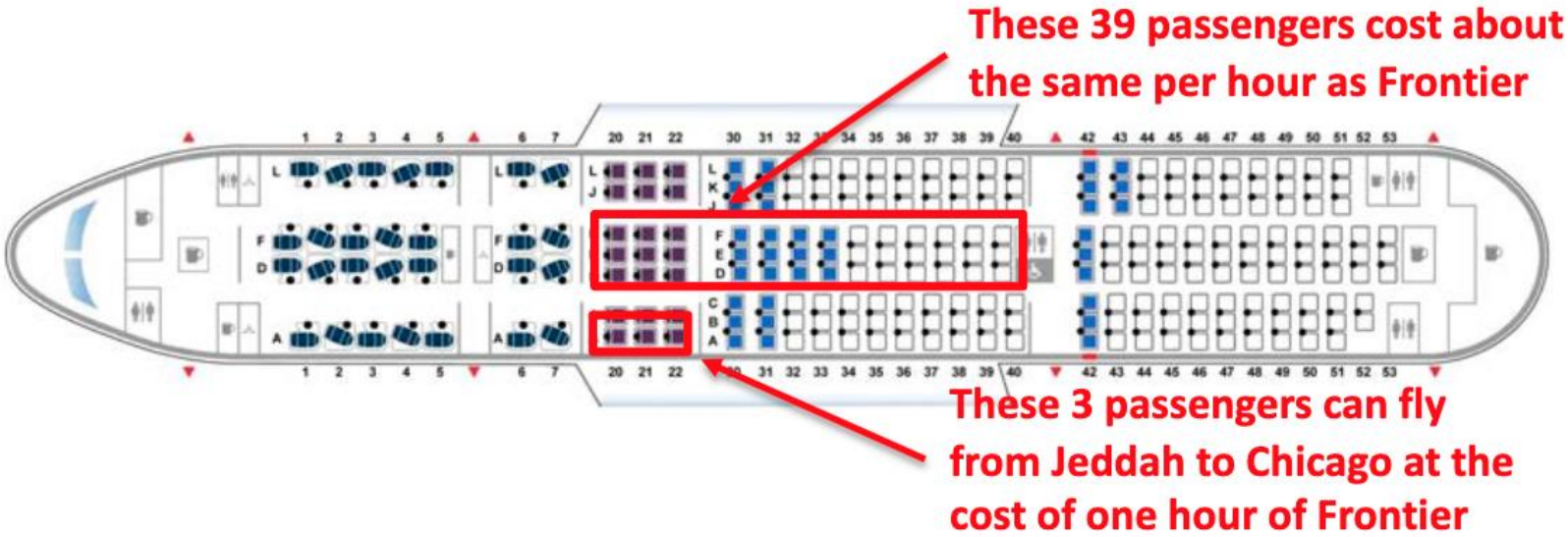
Architechture of a supercomputer



- Login nodes. To log into the system, cd into directories, look at files etc ...
- Compute node. Dedicated to do the computation
- The slurm scheduler controls access to the compute nodes to avoid a tragedy of the commons ...

Why Slurm?

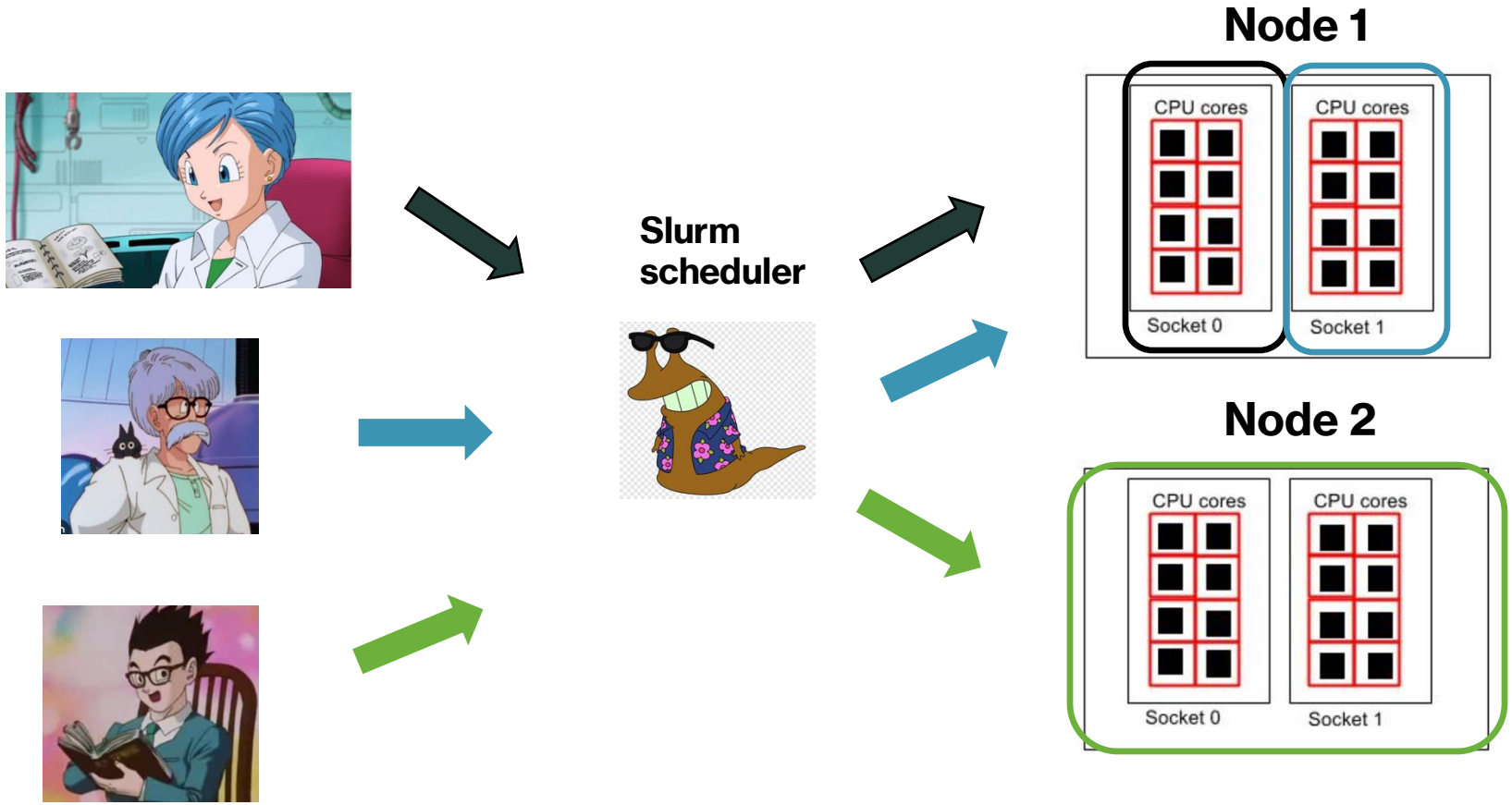
- Allows for better tracking of resource used per user
- Better management of of resource across user base
- More sustainable and less tragedy of the commons
- Widely used and supported by applications in case of bugs.



Source: https://extremecomputingtraining.anl.gov/wp-content/uploads/sites/96/2023/08/ATPESC-2023-Track-2b-Dinnerspeaker-Keyes-Efficient-

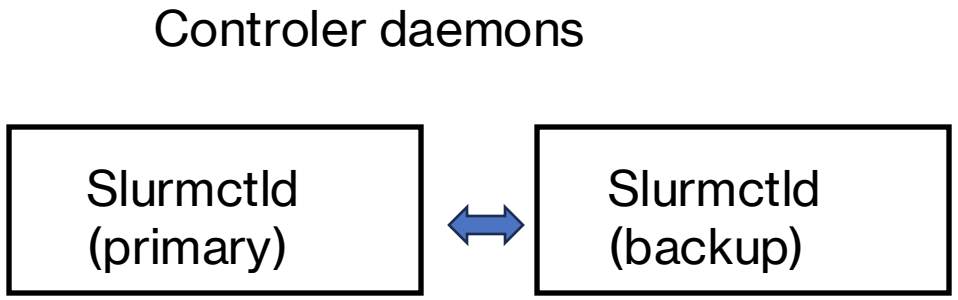
Why Slurm?

- Bulma request 8 cores from Node 1
- Dr. Brief requests 8 cores as well from Node 1
- Gohan requests 16 cores from Node 2
- No tragedy of the common!!!



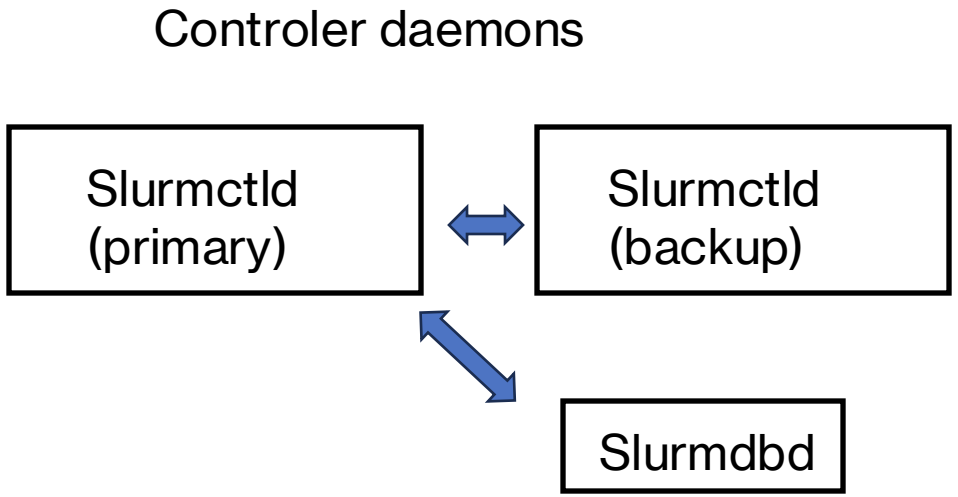
Slurm architechture

- Slurmctld are daemons that monitor the state of resources, decide when to initiate job states and process users command.



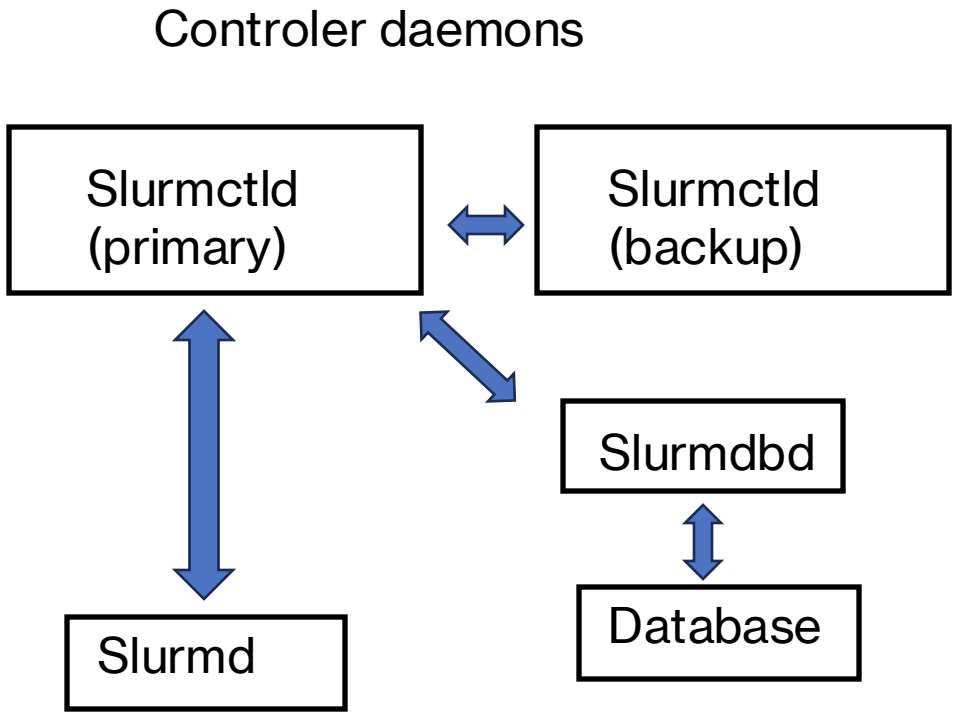
Slurm architechture

- Slurmctld are daemons that monitor the state of resources, decide when to initiate job states and process users command.
- Slurmdbd is the database daemon and records accounting information and manages some other config (qos, fair share ...)



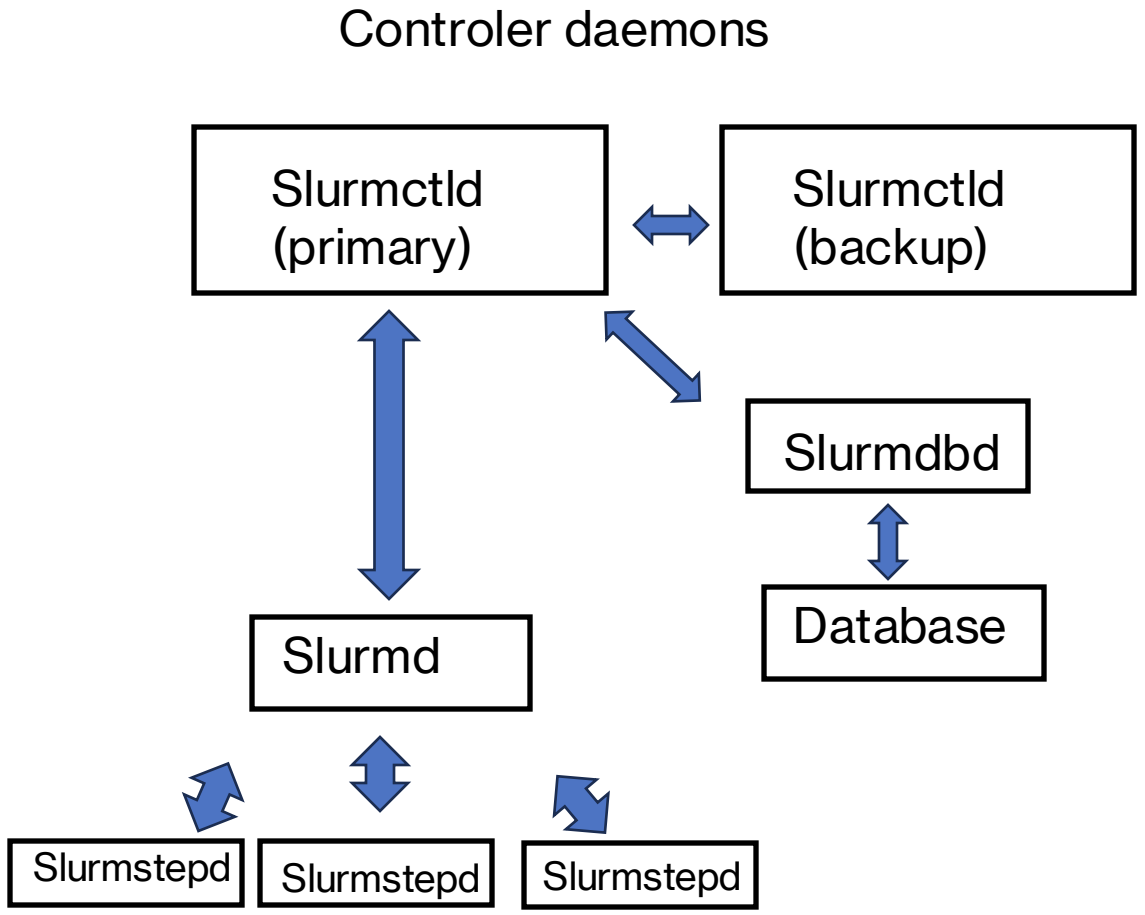
Slurm architechture

- Slurmctld are daemons that monitor the state of resources, decide when to initiate job states and process users command.
- Slurmdbd is the database daemon and records accounting information and manages some other config (qos, fair share ...).
- Slurmd is the slurm compute daemon on compute nodes. Spawn processes on behalf of users

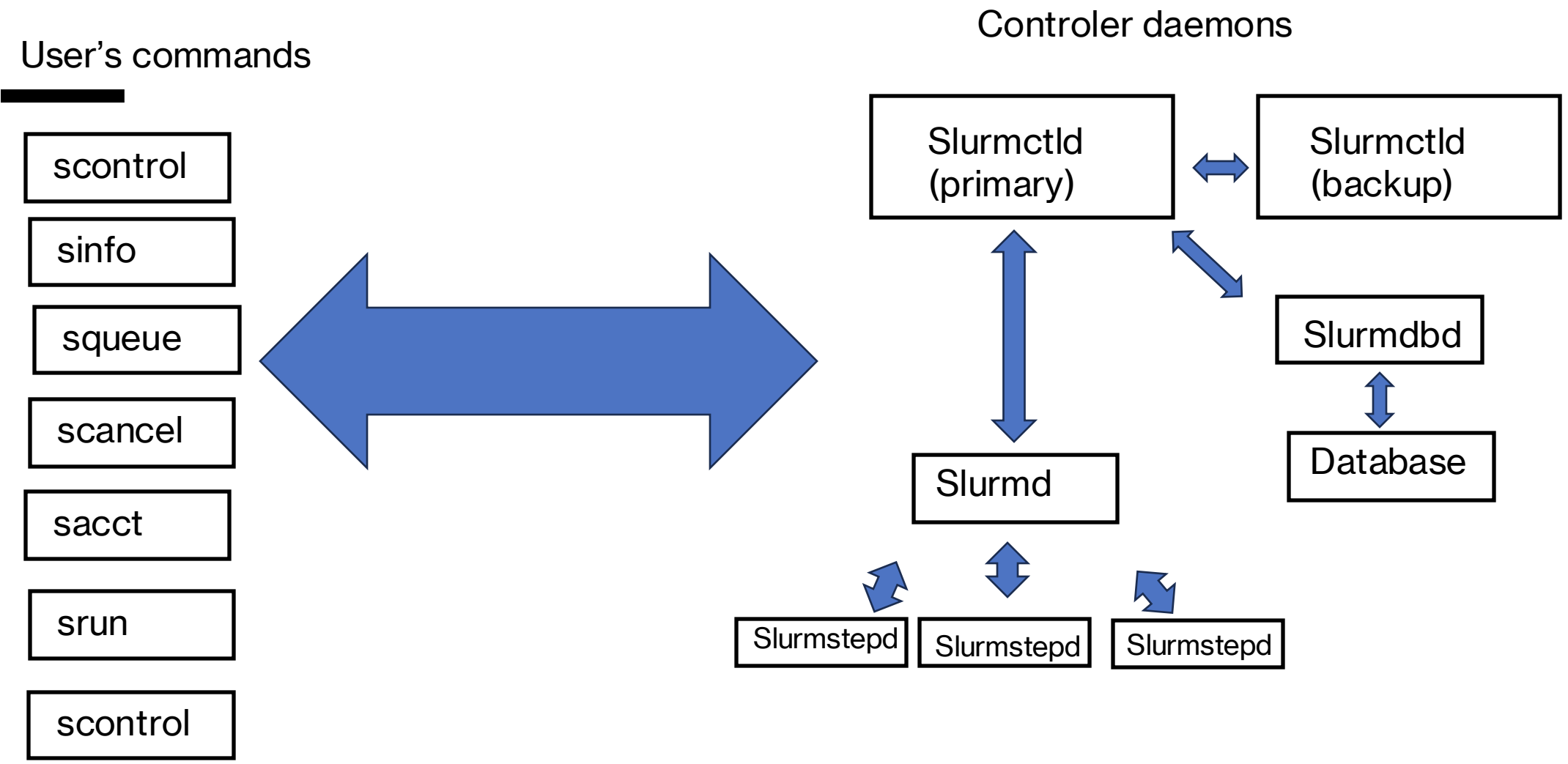


Slurm architechture

- Slurmctld are daemons that monitor the state of resources, decide when to initiate job states and process users command.
- Slurmdbd is the database daemon and records accounting information and manages some other config (qos, fair share ...).
- Slurmd is the slurm compute daemon on compute nodes. Spawn processes on behalf of users
- Slurmstepd are processes spawned by Slurmd that launch user's batch script, manage accounting, I/O, signals during compute.



Slurm architechture



Source: https://www.nersc.gov/assets/Uploads/SLURM-NUG-Nov2015.pdf ;
https://jsspp.org/papers23/JSSPP_2023_keynote_SLURM.pdf

Context[1]

- Our cluster named Alpine is made of 22,180 cores and 382 compute nodes

Context[1]

- Our cluster named Alpine is made of 22,180 cores and 382 compute nodes.
- We have (before hardware update) 8 NVIDIA A100 nodes and 8 AMD MI100 compute nodes.

Context[1]

- Our cluster named Alpine is made of 22,180 cores and 382 compute nodes.
- We have (before hardware update) 8 NVIDIA A100 nodes and 8 AMD MI100 compute nodes.
- Each GPU node is 2x25 Gb Ethernet+RoCE interconnected.

Context [1]

- Our cluster named Alpine is made of 22,180 cores and 382 compute nodes.
- We have (before hardware update) 8 NVIDIA A100 nodes and 8 AMD MI100 compute nodes.
- Each GPU node is 2x25 Gb Ethernet+RoCE interconnected.
- 3 gpus per node.

Context [1]

- Our cluster named Alpine is made of 22,180 cores and 382 compute nodes.
- We have (before hardware update) 8 NVIDIA A100 nodes and 8 AMD MI100 compute nodes.
- Each GPU node is 2x25 Gb Ethernet+RoCE interconnected.
- 3 gpus per node.
- The OS is RedHat 8.8

Context [1]

- Our cluster named Alpine is made of 22,180 cores and 382 compute nodes.
- We have (before hardware update) 8 NVIDIA A100 nodes and 8 AMD MI100 compute nodes.
- Each GPU node is 2x25 Gb Ethernet+RoCE interconnected.
- 3 gpus per node.
- The OS is RedHat 8.8
- Grace Hopper and L40 gpu nodes will be ready sometimes this year.

Software stack

- Our software versions on the cuda side are: cuda/11.2, cuda/11.3,cuda/11.4, cuda/11.8 and cuda/12.1.1

Software stack

- Our software versions on the cuda side are: cuda/11.2, cuda/11.3,cuda/11.4, cuda/11.8 and cuda/12.1.1

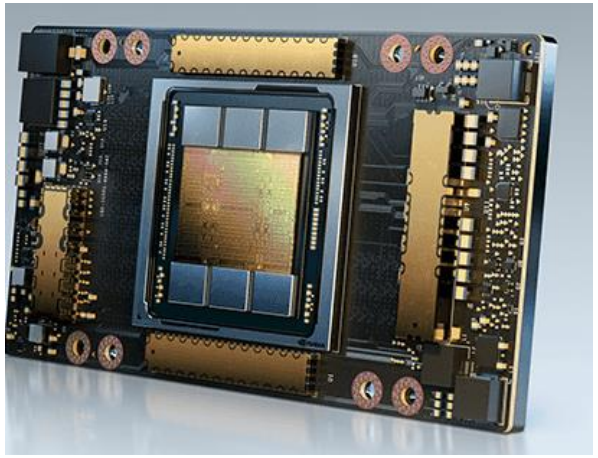
- Our software versions on the AMD side are: rocm/5.2.3, rocm/5.3.0, rocm/5.5.0, rocm/6.1.0

Computing situation

- Our users on the Medical Campus side have been used to computation on the A100 nodes

Alpine NVIDIA A100 Specs

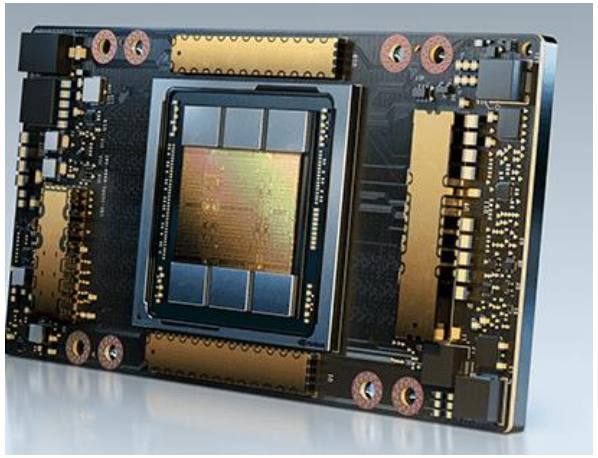
- 40GB or 80 GB global memory for each gpu.



Source: https://www.amd.com/en/products/accelerators/instinct/mi100.html

Alpine NVIDIA A100 Specs

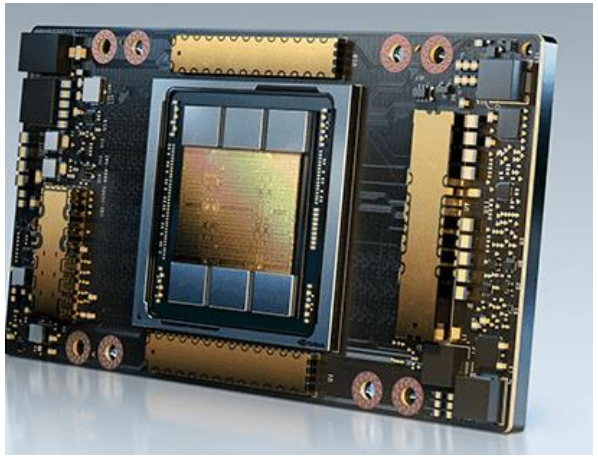
- 40GB or 80 GB global memory for each gpu.
- Peak memory bandwidth is >2TB/s.



Source: https://www.amd.com/en/products/accelerators/instinct/mi100.html

Alpine NVIDIA A100 Specs

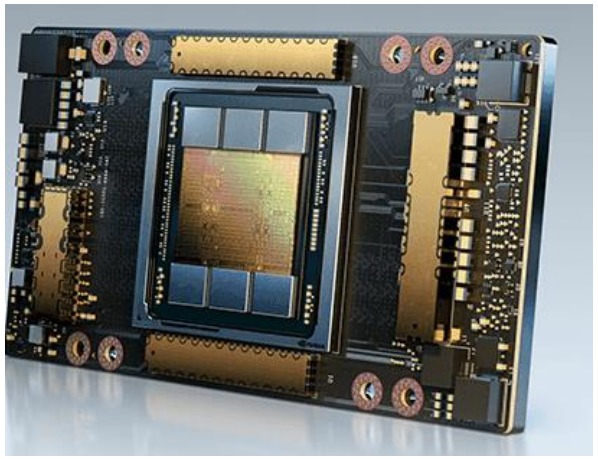
- 40GB or 80 GB global memory for each gpu.
- Peak memory bandwidth is >2TB/s.
- 3 NVIDIA gpus per node



Source: https://www.amd.com/en/products/accelerators/instinct/mi100.html

Alpine NVIDIA A100 Specs

- 40GB or 80 GB global memory for each gpu.
- Peak memory bandwidth is >2TB/s.
- 3 NVIDIA gpus per node
- Two partition names on Alpine: aa100 for batch submission and atesting_a1100 as a debugging partition



Source: https://www.amd.com/en/products/accelerators/instinct/mi100.html

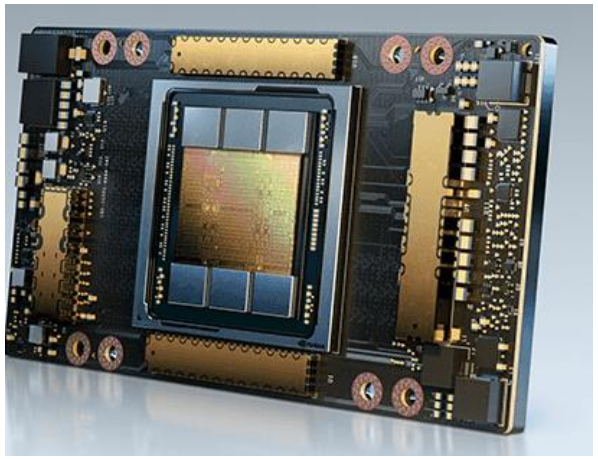
Alpine NVIDIA A100 Specs

- 40GB or 80 GB global memory for each gpu.
- Peak memory bandwidth is >2TB/s.
- 3 NVIDIA gpus per node
- Two partition names on Alpine: aa100 for batch submission and atesting_a1100 as a debugging partition

To access the A100 compute node with 80G please include

the following below in your slurm script:

- **#SBATCH --partition=aa100**
  #SBATCH --constraint=gpu80



Source: https://www.amd.com/en/products/accelerators/instinct/mi100.html

Alpine AMD MI100 Specs

- 32 GB global memory for each gpu (80 GB memory for the NVIDIA A100 gpu)

Source: https://www.amd.com/en/products/accelerators/instinct/mi100.html

Alpine AMD MI100 Specs

- 32 GB global memory for each gpu (80 GB memory for the NVIDIA A100 gpu)
- Peak memory bandwidth is 1.2 TB/s (>2TB/s for NVIDIA A100)

Source: https://www.amd.com/en/products/accelerators/instinct/mi100.html

Alpine AMD MI100 Specs

- 32 GB global memory for each gpu (80 GB memory for the NVIDIA A100 gpu)
- Peak memory bandwidth is 1.2 TB/s (>2TB/s for NVIDIA A100)
- 3 AMD gpus per node

Source: https://www.amd.com/en/products/accelerators/instinct/mi100.html

Alpine AMD MI100 Specs

- 32 GB global memory for each gpu (80 GB memory for the NVIDIA A100 gpu)
- Peak memory bandwidth is 1.2 TB/s (>2TB/s for NVIDIA A100)
- 3 AMD gpus per node
- Two partition names on Alpine: ami100 for batch submission and atesting_mi100 as a debugging partition

Source: https://www.amd.com/en/products/accelerators/instinct/mi100.html

Slurm & GPU partitions

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
```

Shows partition name, total nodes, and time limit

Slurm & GPU partitions

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
```

Shows current State, Default walltime, Billing, type of nodes etc ...

Slurm & GPU partitions

Requesting Alpine GPUs with Slurm

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

File display

```
--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
```

Slurm & GPU partitions

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
```

File display

Pay attention to defaults!

To access 80G on debug mode!!

Slurm & GPU partitions

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

File display

11/04/24

19

There are rules with how much GPUs you can get

Slurm & GPU partitions

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

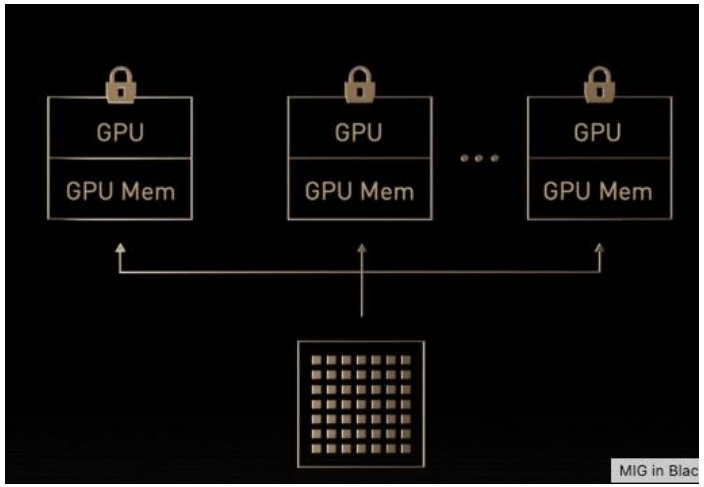
File display

There are rules with how much GPUs you can get

Source: https://github.com/ResearchComputing/Intro_GPU_Acceleration/blob/main/GPU_acceleration_Nov24.pdf

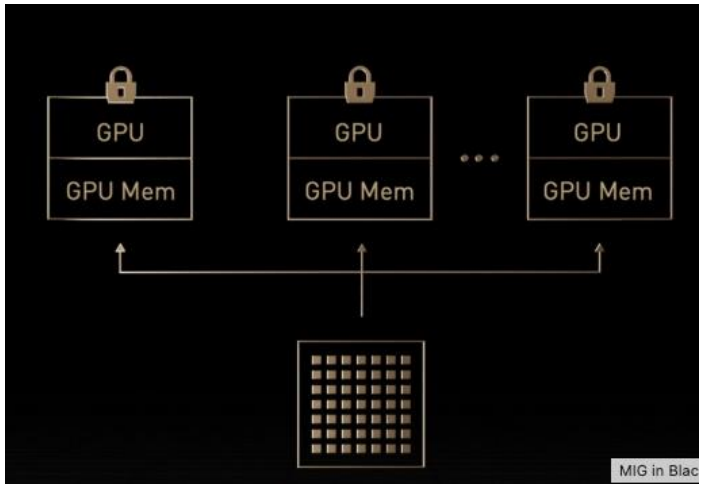
NVIDIA MIG

- The atesting_a100 partition utilizes NVIDIA's Multi-Instance GPU (MIG) feature, which can "slice" GPUs into multiple GPU instances.



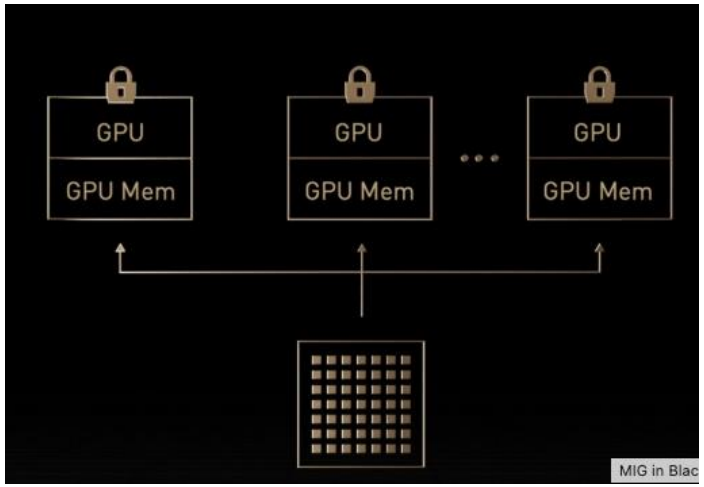
NVIDIA MIG

- The atesting_a100 partition utilizes NVIDIA's Multi-Instance GPU (MIG) feature, which can "slice" GPUs into multiple GPU instances.
- These GPU instances can be treated as a single GPU.



NVIDIA MIG

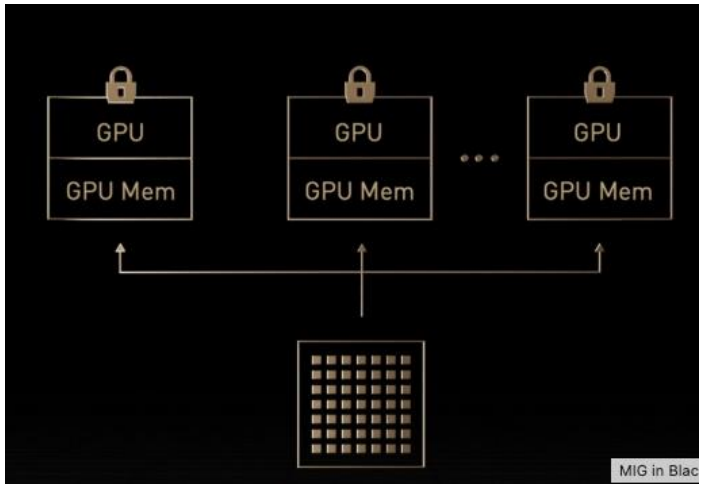
- The atesting_a100 partition utilizes NVIDIA's Multi-Instance GPU (MIG) feature, which can "slice" GPUs into multiple GPU instances.
- These GPU instances can be treated as a single GPU.
- One important limitation is that MIG does not allow for multiple GPU instances to communicate with each other.



NVIDIA MIG

- sinteractive --partition=atesting_a100 --gres=gpu --ntasks=10 --time=30:00

(Request 1 A100 MIG slice with 10 CPU cores for 30 minutes)



IV- Intro to GPU profiling NVIDIA

3 steps

- Analyze code: for assessing the blocks that need optimization

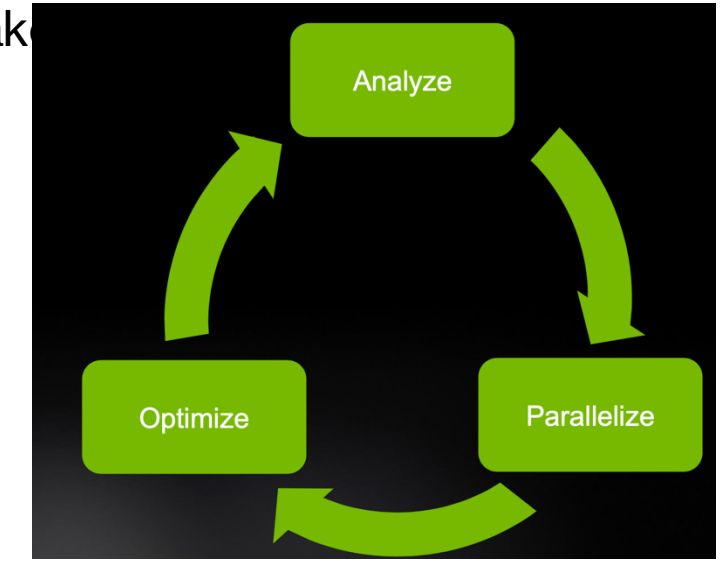
3 steps

- Analyze code: for assessing the blocks that need optimization

- Then parallelize the further by rewriting the parts that take the most time

3 steps

- Analyze code: for assessing the blocks that need optimization
- Then parallelize the further by rewriting the parts that tak
- Optimize the code
- Repeat



Source: Openhackaton NVIDIA GPU hackaton Princeton Spring 2023, Nsights

Benefits of nsights

- Allows to measure the bottlenecks of the pipeline to improve efficiency

- Allows to avoid work based on intuition or guess (is the pipeline CPU or GPU bound?)

- See how asynchronous the software is interacting.

Source: https://alcf.anl.gov/sites/default/files/2022-07/Nsight%20Systems%20-%20DL%20Profiling%20Argonne%20National%20Labs%202022-06-30.pdf

2 things necessary to profile

- You need to have nsys on the hpc

  module load cuda/11.8

- When running your GPU pipeline run it with nsys and it will create an output

- Export that output profile to your local laptop where you downloaded the nsight app.

- You may profile for a 30s to 2 min to get an idea of the bottleneck.

  You do not need to profile for the entire time if you have a heavy run

nsys command

```
nsys profile --trace=cuda,nvtx,osrt,cudnn,cublas --backtrace=dwarf
--capture-range=cudaProfilerApi --gpu-metrics-devices=all -output=oft-profile-dwarf4
sh scripts/expt-singleGPU.sh --profile 50 --profile_start 5000 --profile_epoch 1
```

Nsight Systems CLI command
Select APIs to trace
Enable GPU memory use tracking (but there is extra overhead)
Collect thread call-stack sample backtraces via DWARF info - deeper but more expensive to collect
Trigger collection on cudaProfilerStart API in application, or consider timer-based options
GPU metrics sampling at default 10khz
Name the report file
Application command - plus arguments for when to start profiling

https://docs.nvidia.com/nsight-systems/UserGuide/index.html#cli-profile-command-switch-options

srun nsys profile … required on multi-node or multi-container
nsys profile mpirun … optional on single node to produce a single report

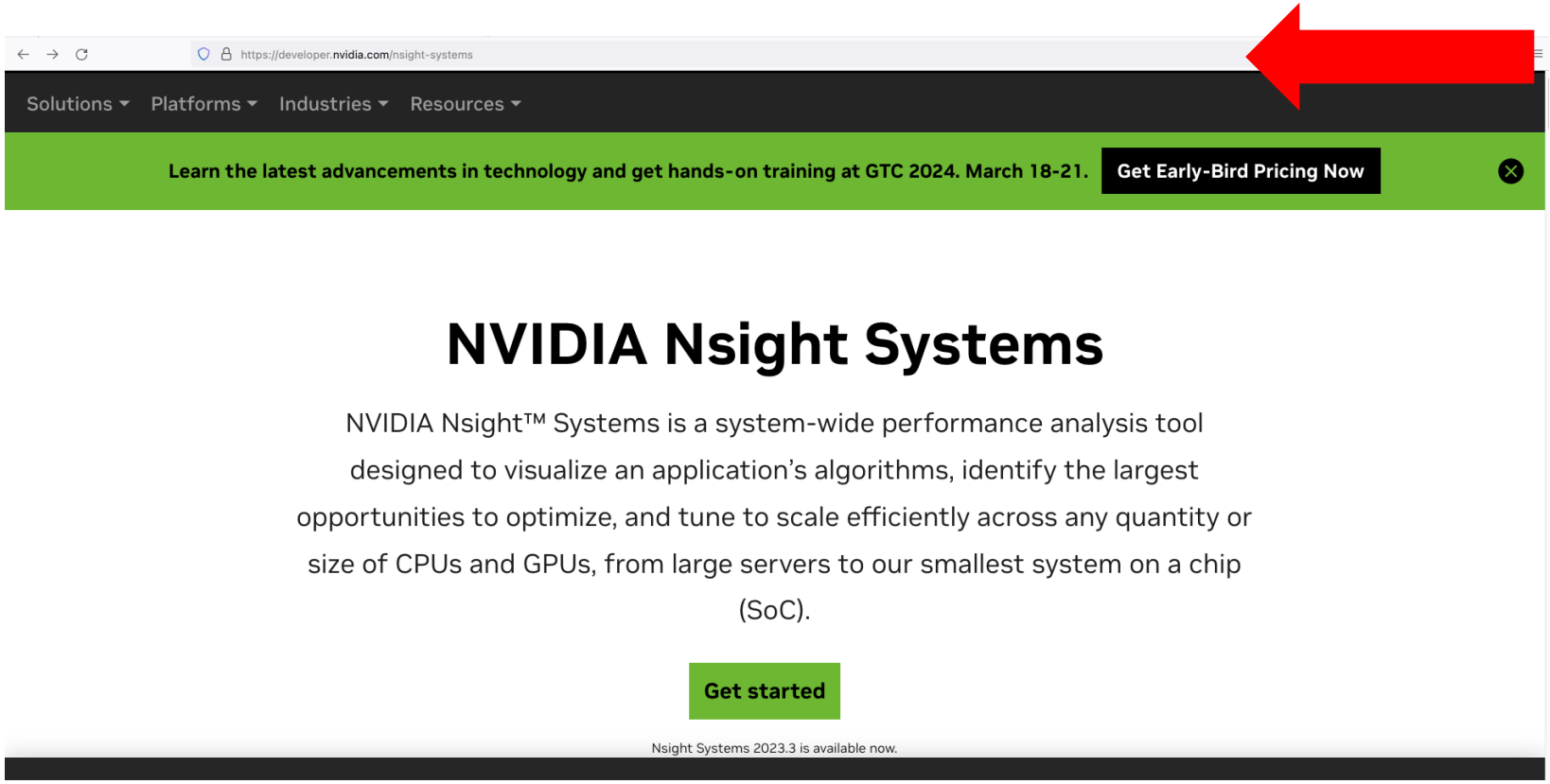
Nsys with Deep learning Horovod on Alpine (Slurm script)

```
# We test it with 4 GPUs
echo "Testing it with 4 GPUs"
nsys profile --trace=cuda,nvtx,osrt,cudnn,cublas \
        --capture-range=cudaProfilerApi \
        --gpu-metrics-device=all \
        --output=horovod_profile \
        horovodrun -np 4 -H $node1:2,$node2:2
        horovod_housing_tf.py
```

We use 2 GPUs
on node 1 and 2
on node 2

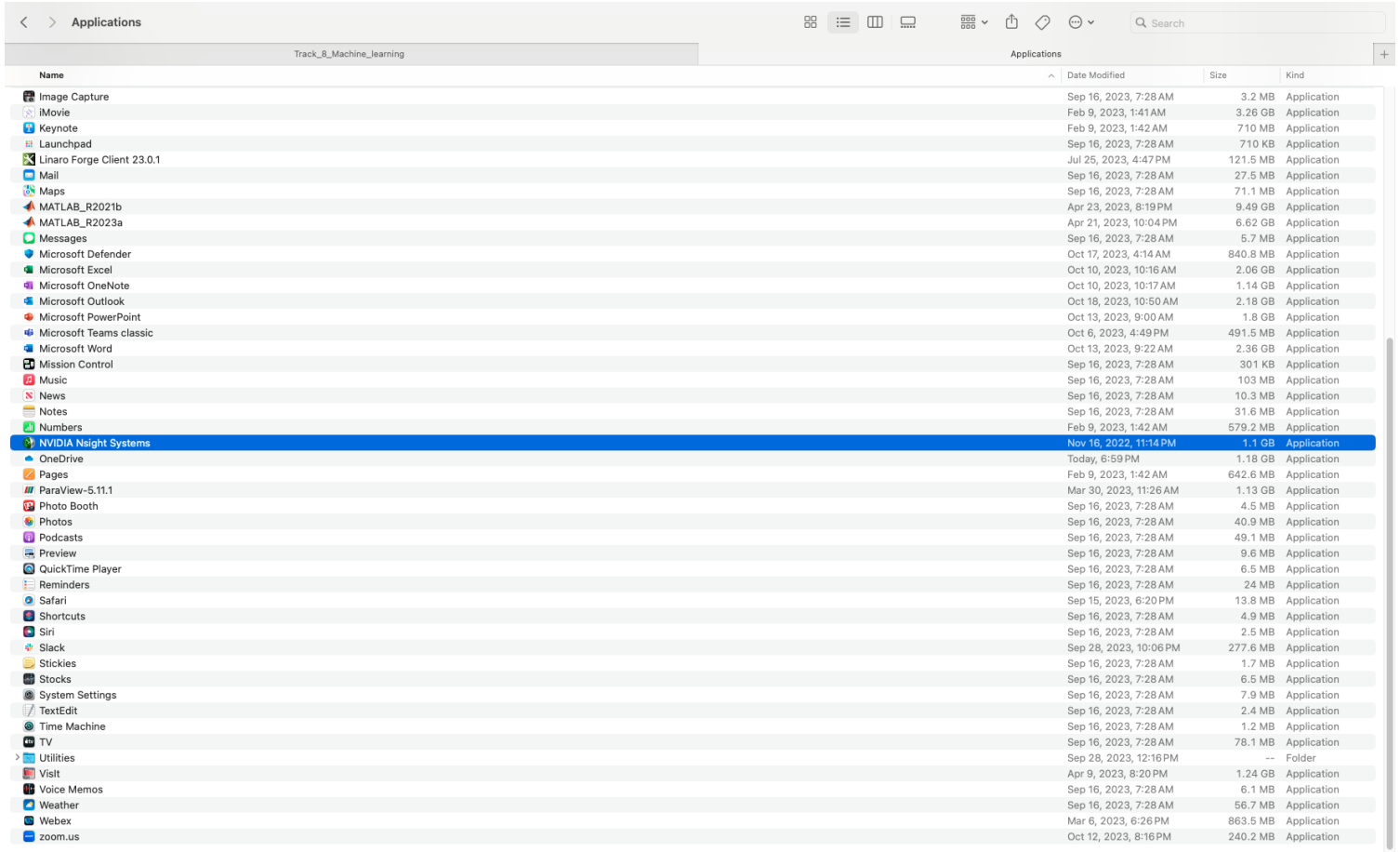
NVIDIA Nsight

- Download the NVIDIA Nsight system client tool to your local computer

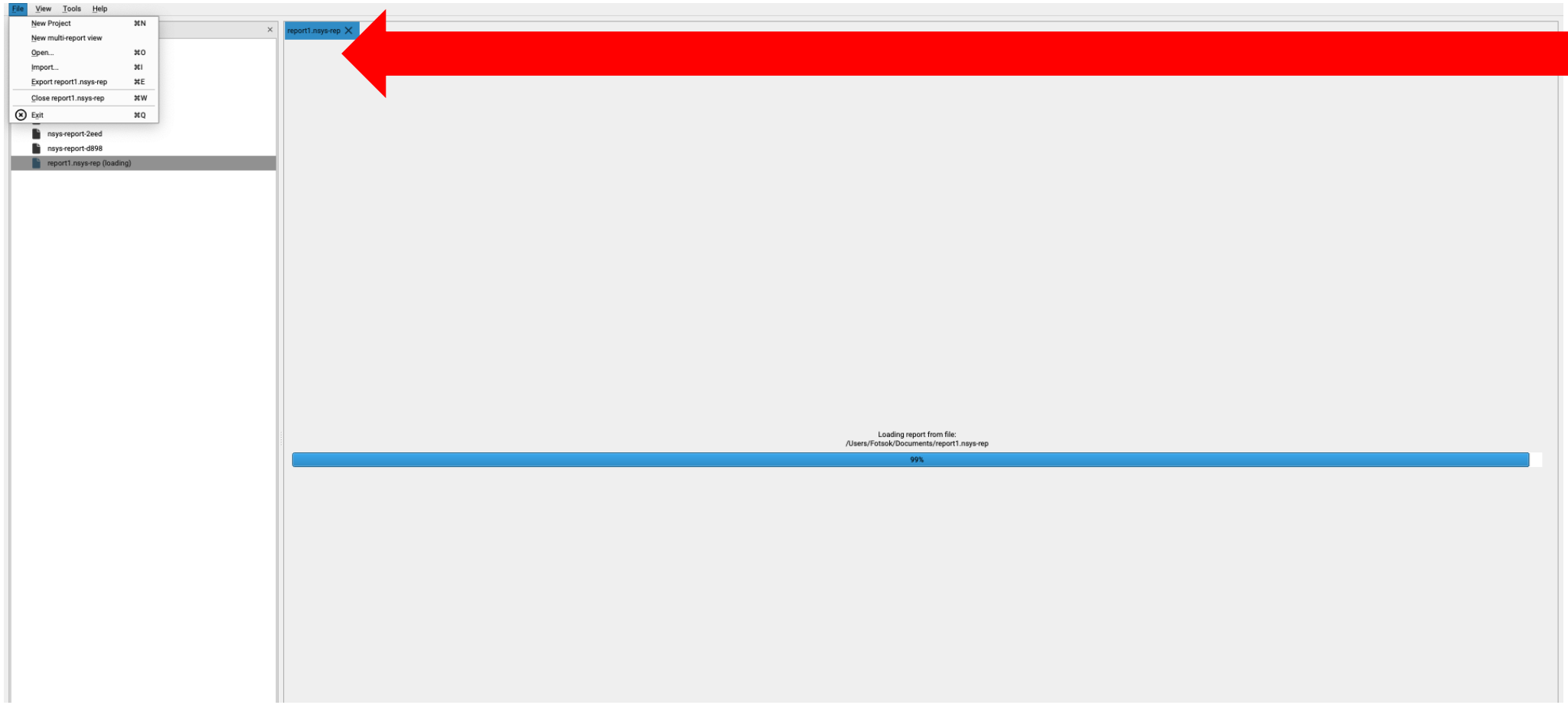


https://developer.nvidia.com/nsight-systems

NVIDIA Nsight (2)

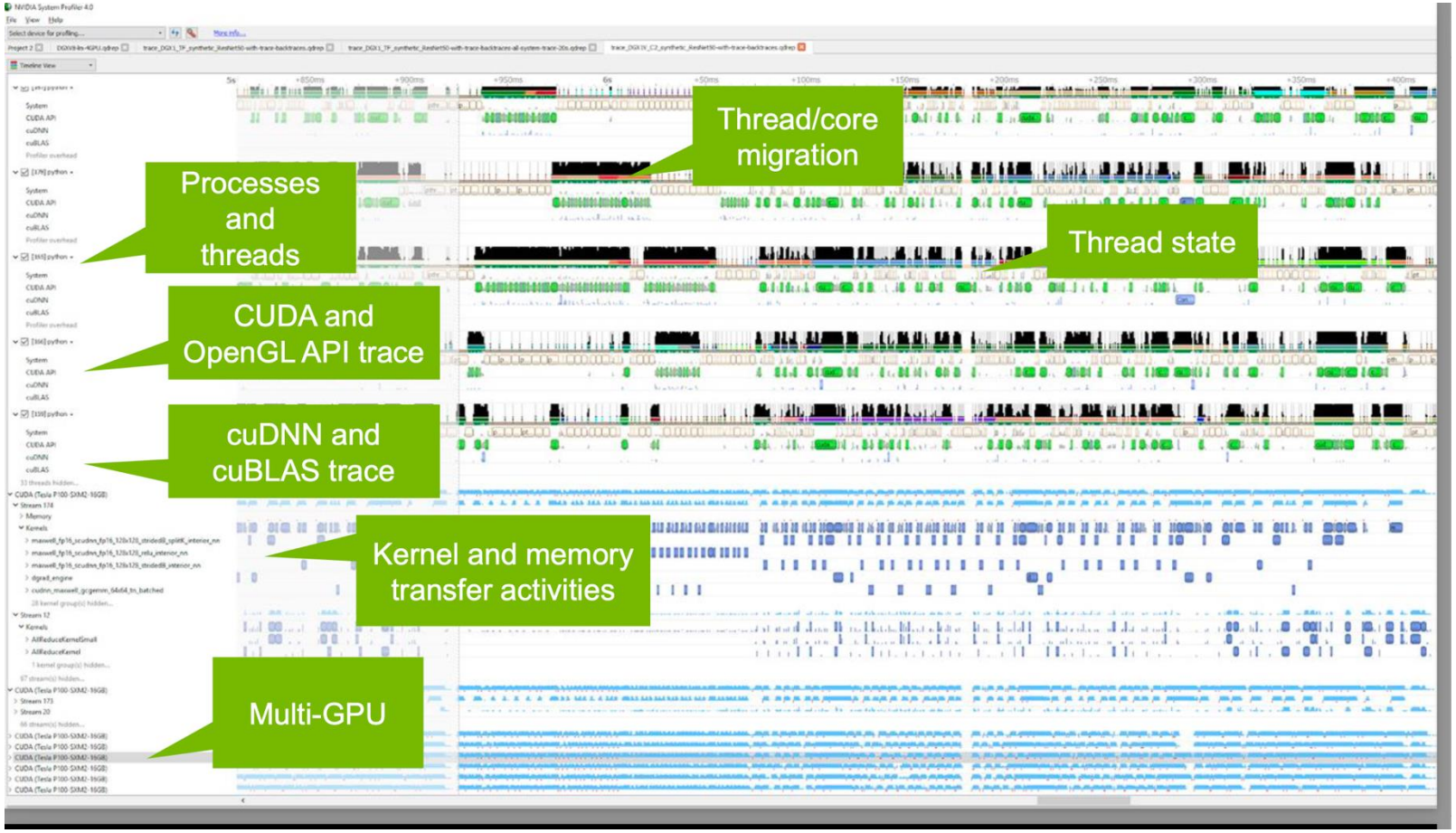


NVIDIA Nsight (3)



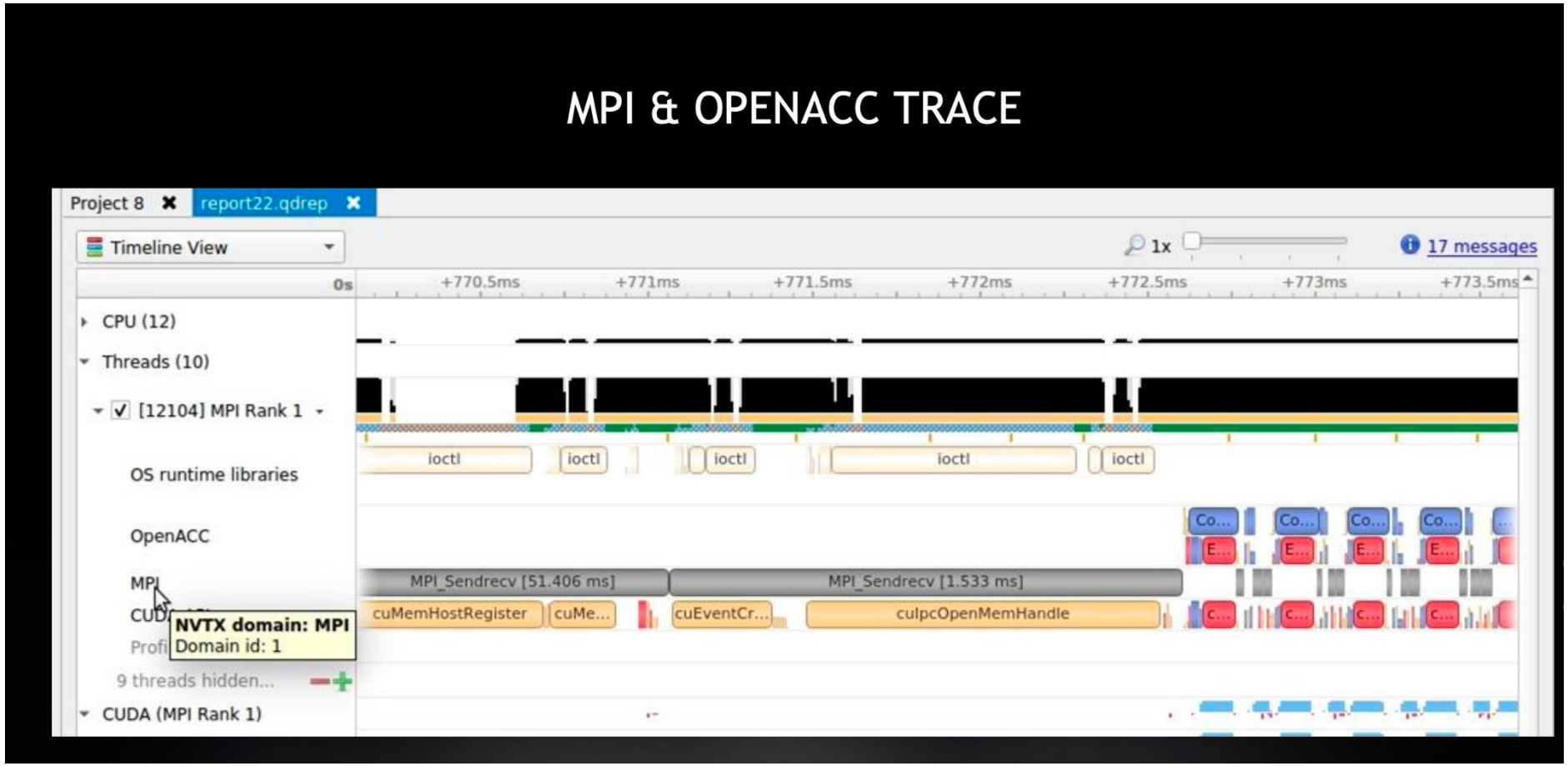
Click on open

What can be visualize with nsys



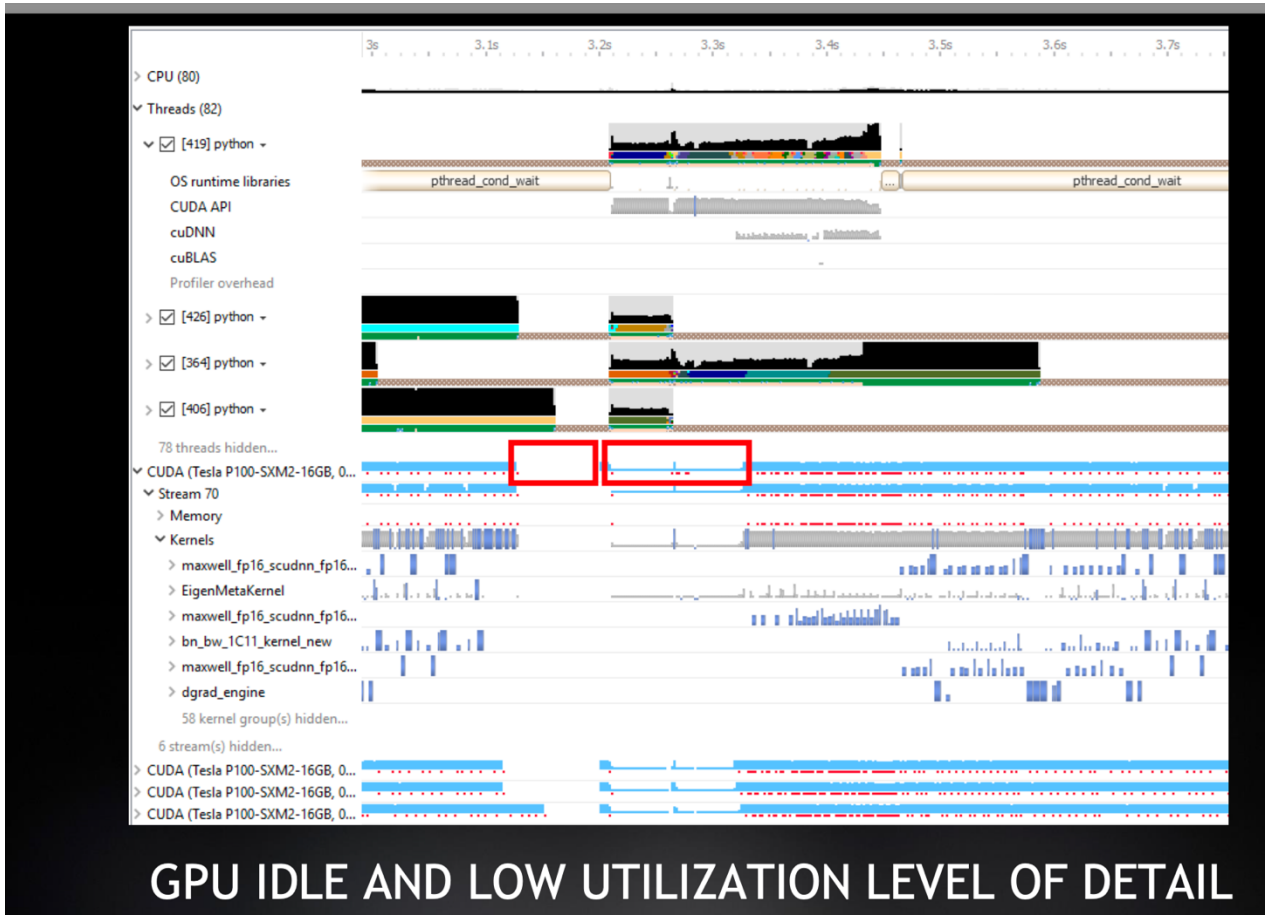
Source: Openhackaton NVIDIA GPU hackaton Princeton Spring 2023, Nsights

What can be visualize with nsys



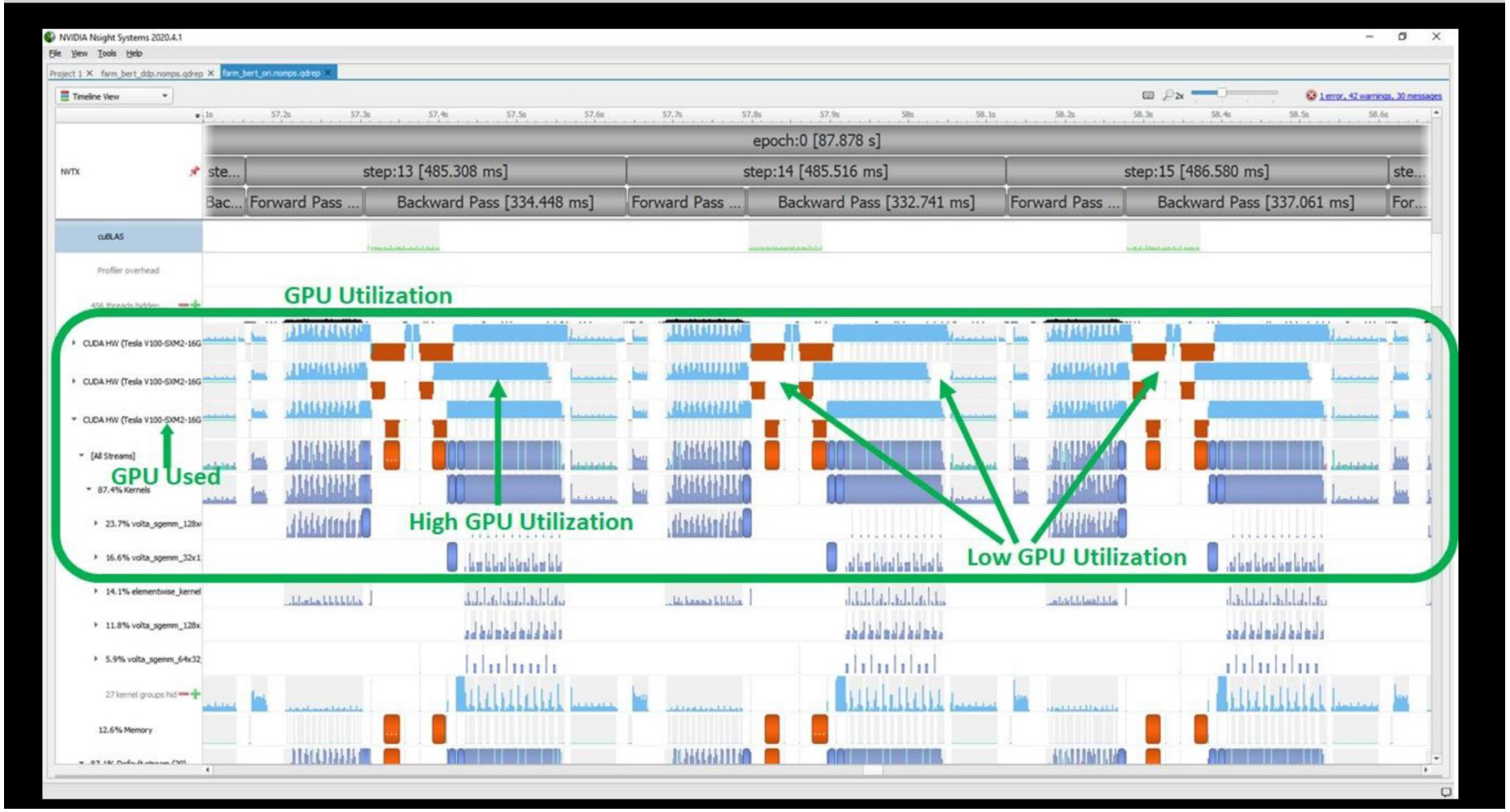
Source: Openhackaton NVIDIA GPU hackaton Princeton Spring 2023, Nsights

What can be visualize with nsys



Source: Openhackaton NVIDIA GPU hackaton Princeton Spring 2023, Nsights

High vs low GPU utilization



Source: https://alcf.anl.gov/sites/default/files/2022-

AMD GPU software stack

- Heterogeneous Computing interface (Hip) is a C++ software stack API and kernel language to automatically run, convert or create code on AMD or NVIDIA GPUs.

https://raw.githubusercontent.com/RadeonOpenCompute/ROCm/rocm-4.5.2/AMD_HIP_Programming_Guide.pdf

AMD GPU software stack

- Heterogeneous Computing interface (Hip) is a C++ software stack API and kernel language to automatically run, convert or create code on AMD or NVIDIA GPUs.
- Its equivalent with NVIDIA is cuda.

https://raw.githubusercontent.com/RadeonOpenCompute/ROCm/rocm-4.5.2/AMD_HIP_Programming_Guide.pdf

AMD GPU software stack

- Heterogeneous Computing interface (Hip) is a C++ software stack API and kernel language to automatically run, convert or create code on AMD or NVIDIA GPUs.
- Its equivalent with NVIDIA is cuda.
- Radeon Open Compute (ROCm) is an ensemble of software stack that incorporates HIP.

https://raw.githubusercontent.com/RadeonOpenCompute/ROCm/rocm-4.5.2/AMD_HIP_Programming_Guide.pdf

AMD GPU software stack

- Heterogeneous Computing interface (Hip) is a C++ software stack API and kernel language to automatically run, convert or create code on AMD or NVIDIA GPUs.
- Its equivalent with NVIDIA is cuda.
- Radeon Open Compute (ROCm) is an ensemble of software stack that incorporates HIP.
- It allows to port CUDA code, supports deep learning pipelines (Pytorch, Tensorflow) and math libraries (Blas, FFT, Sparse etc ...)

https://raw.githubusercontent.com/RadeonOpenCompute/ROCm/rocm-4.5.2/AMD_HIP_Programming_Guide.pdf

AMD GPU computing on Alpine

- Make sure that you are in the video group to use AMD GPUs!!!

AMD GPU computing on Alpine

- Make sure that you are in the video group to use AMD GPUs!!!
- Without the video group you will not be able to see the AMD GPUs

sinteractive --partition=ami100 –reservation=amc_workshop --time=02:00:00 --ntasks=4 –-gres=gpu:1

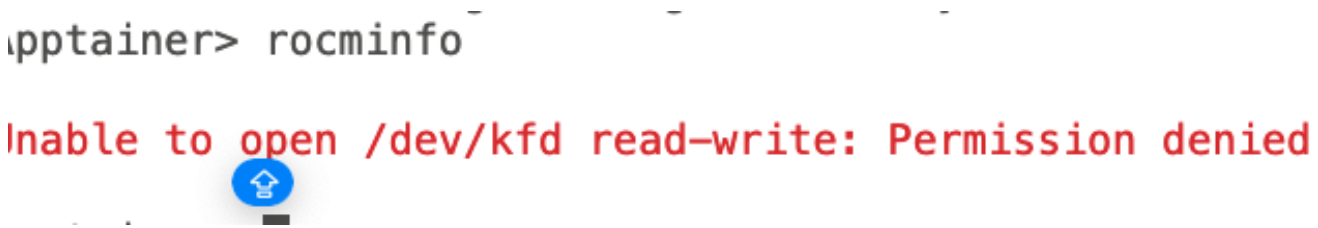
- groups $USER

AMD GPU computing on Alpine

- ROCMinfo gives information about the GPU model. If when running it you do not see anything then you will not be able to use the GPU

AMD GPU computing on Alpine

- ROCMinfo gives information about the GPU model. If when running it you do not see anything then you will not be able to use the GPU



AMD GPU computing on Alpine

- Your output should look more like this!

```
[kfotso@xsede.org@c3gpu-c2-u17 workshop_amd_gpu]$ rocminfo

=====================
HSA System Attributes
=====================
Runtime Version:         1.1
System Timestamp Freq.:  1000.000000MHz
Sig. Max Wait Duration:  18446744073709551615 (0xFFFFFFFFFFFFFFFF) (timestamp count)
Machine Model:           LARGE
System Endianness:       LITTLE

==========
HSA Agents
==========
*******
Agent 1
*******
  Name:                    AMD EPYC 7543 32-Core Processor
  Uuid:                    CPU-XX
  Marketing Name:          AMD EPYC 7543 32-Core Processor
  Vendor Name:             CPU
```

AMD GPU computing on Alpine

- Hipconfig provides useful information about the hip library

```
[kfotso@xsede.org@c3gpu-c2-u17 workshop_amd_gpu]$ hipconfig
HIP version  : 5.4.22801-aaa1e3d8

== hipconfig
HIP_PATH     : /opt/rocm-5.4.0
ROCM_PATH    : /opt/rocm-5.4.0
HIP_COMPILER : clang
HIP_PLATFORM : amd
HIP_RUNTIME  : rocclr
CPP_CONFIG   :  -D__HIP_PLATFORM_HCC__= -D__HIP_PLATFORM_AMD__= -I/opt/rocm-5.4.0/include -I/opt/rocm-5.4.0/llvm/bin/../lib/clang/15.0.0 -I/opt/rocm-5.4.0/hsa
/include

== hip-clang
HSA_PATH         : /opt/rocm-5.4.0/hsa
HIP_CLANG_PATH   : /opt/rocm-5.4.0/llvm/bin
```

Running our first model

- This example shows how we fit a 3rd order polynomial to a sine function
- Example from https://pytorch.org/tutorials/beginner/pytorch_with_examples.html
- Please run script **part2-run_amd_gpu_debug_level1.slurm** from the login node
- Make sure that you are logged out from the interactive session

Running our first model

```
#!/bin/bash
#SBATCH --job-name=run_amd_gpu_standard
#SBATCH --output=run_amd_gpu_standard.%j.out
#SBATCH --error=run_amd_gpu_standard.%j.err
#SBATCH --partition=ami100
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=6
#SBATCH --gres=gpu:1
#SBATCH --account=amc-general
#SBATCH --reservation=amc_workshop
#SBATCH --time=00:10:00


# We make sure that the user can run mambaforge
cp ~/.condarc ~/.mambarc

echo "********************************** Loading ROCM and pytorch + exporting useful paths  **********************************"
# We load ROCM which is the collection of drivers, libraries, tools and API to run
# on an AMD GPU : https://rocm.docs.amd.com/en/latest/what-is-rocm.html
# For NVIDIA it is NVIDIA CUDA
module load rocm/5.2.3

# We load pytorch 1.13.0 that has been built for the ROCM,
# which is compatible with ROCM5.2.X : https://pytorch.org/get-started/previous-versions/
module load pytorch/1.13.0
```

We give job name, output and error

Running our first model

```
#!/bin/bash
#SBATCH --job-name=run_amd_gpu_standard
#SBATCH --output=run_amd_gpu_standard.%j.out
#SBATCH --error=run_amd_gpu_standard.%j.err
#SBATCH --partition=ami100
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=6
#SBATCH --gres=gpu:1
#SBATCH --account=amc-general
#SBATCH --reservation=amc_workshop
#SBATCH --time=00:10:00


# We make sure that the user can run mambaforge
cp ~/.condarc ~/.mambarc

echo "********************************** Loading ROCM and pytorch + exporting useful paths  **********************************"
# We load ROCM which is the collection of drivers, libraries, tools and API to run
# on an AMD GPU : https://rocm.docs.amd.com/en/latest/what-is-rocm.html
# For NVIDIA it is NVIDIA CUDA
module load rocm/5.2.3

# We load pytorch 1.13.0 that has been built for the ROCM,
# which is compatible with ROCM5.2.X : https://pytorch.org/get-started/previous-versions/
module load pytorch/1.13.0
```

Name of the AMD GPU partition

Running our first model

```
#!/bin/bash
#SBATCH --job-name=run_amd_gpu_standard
#SBATCH --output=run_amd_gpu_standard.%j.out
#SBATCH --error=run_amd_gpu_standard.%j.err
#SBATCH --partition=ami100
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=6
#SBATCH --gres=gpu:1
#SBATCH --account=amc-general
#SBATCH --reservation=amc_workshop
#SBATCH --time=00:10:00


# We make sure that the user can run mambaforge
cp ~/.condarc ~/.mambarc

echo "********************************** Loading ROCM and pytorch + exporting useful paths  **********************************"
# We load ROCM which is the collection of drivers, libraries, tools and API to run
# on an AMD GPU : https://rocm.docs.amd.com/en/latest/what-is-rocm.html
# For NVIDIA it is NVIDIA CUDA
module load rocm/5.2.3

# We load pytorch 1.13.0 that has been built for the ROCM,
# which is compatible with ROCM5.2.X : https://pytorch.org/get-started/previous-versions/
module load pytorch/1.13.0
```

We ask for 1 node and a few cores since most computation will be on the gpu

Running our first model

```
#!/bin/bash
#SBATCH --job-name=run_amd_gpu_standard
#SBATCH --output=run_amd_gpu_standard.%j.out
#SBATCH --error=run_amd_gpu_standard.%j.err
#SBATCH --partition=ami100
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=6
#SBATCH --gres=gpu:1
#SBATCH --account=amc-gener
#SBATCH --reservation=amc_workshop
#SBATCH --time=00:10:00


# We make sure that the user can run mambaforge
cp ~/.condarc ~/.mambarc

echo "********************************** Loading ROCM and pytorch + exporting useful paths  **********************************"
# We load ROCM which is the collection of drivers, libraries, tools and API to run
# on an AMD GPU : https://rocm.docs.amd.com/en/latest/what-is-rocm.html
# For NVIDIA it is NVIDIA CUDA
module load rocm/5.2.3

# We load pytorch 1.13.0 that has been built for the ROCM,
# which is compatible with ROCM5.2.X : https://pytorch.org/get-started/previous-versions/
module load pytorch/1.13.0
```

We ask
for 1 GPU

Running our first model

```
#!/bin/bash
#SBATCH --job-name=run_amd_gpu_standard
#SBATCH --output=run_amd_gpu_standard.%j.out
#SBATCH --error=run_amd_gpu_standard.%j.err
#SBATCH --partition=ami100
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=6
#SBATCH --gres=gpu:1
#SBATCH --account=amc-general
#SBATCH --reservation=amc_workshop
#SBATCH --time=00:10:00


# We make sure that the user can run mambaforge
cp ~/.condarc ~/.mambarc

echo "********************************** Loading ROCM and pytorch + exporting useful paths  **********************************"
# We load ROCM which is the collection of drivers, libraries, tools and API to run
# on an AMD GPU : https://rocm.docs.amd.com/en/latest/what-is-rocm.html
# For NVIDIA it is NVIDIA CUDA
module load rocm/5.2.3

# We load pytorch 1.13.0 that has been built for the ROCM,
# which is compatible with ROCM5.2.X : https://pytorch.org/get-started/previous-versions/
module load pytorch/1.13.0
```

We load the modules

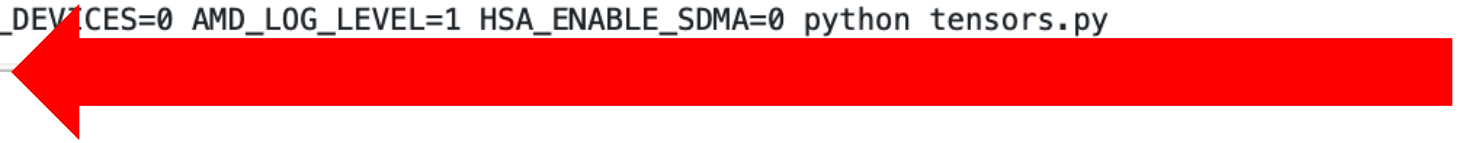
Running our first model

```
echo "********************************** Running the models  **********************************"
# We want to make sure that we can profile what is going on
# We want to set AMD_LOG_LEVEL for profiling and debugging
#CODE 0 means No log
#CODE 1 means log the error
#CODE 2 means log warnings
#CODE 3 means log information
#CODE 4 means debug mode
# More information here: https://rocm.docs.amd.com/projects/HIP/en/develop/developer_guide/logging.html

HIP_VISIBLE_DEVICES=0 ROCR_VISIBLE_DEVICES=0 AMD_LOG_LEVEL=1 HSA_ENABLE_SDMA=0 python tensors.py
```

Here is a way we can debug our code or even profile what is going on

Running our first model

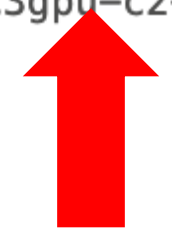
```
echo "************************************ Running the models  ********************************"
# We want to make sure that we can profile what is going on
# We want to set AMD_LOG_LEVEL for profiling and debugging
#CODE 0 means No log
#CODE 1 means log the error
#CODE 2 means log warnings
#CODE 3 means log information
#CODE 4 means debug mode
# More information here: https://rocm.docs.amd.com/projects/HIP/en/develop/developer_guide/logging.html

HIP_VISIBLE_DEVICES=0 ROCR_VISIBLE_DEVICES=0 AMD_LOG_LEVEL=1 HSA_ENABLE_SDMA=0 python tensors.py
```



AMD_LOG_LEVEL variable is very useful

While the job is running ...

```
[kfotso@xsede.org@login-ci1 workshop_amd_gpu]$ squeue --me
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           5187143  acompile acompile kfotso@x  R      10:03      1 c3cpu-c11-u3-4
           5187117    ami100 sinterac kfotso@x  R      20:16      1 c3gpu-c2-u17
[kfotso@xsede.org@login-ci1 workshop_amd_gpu]$
```



We look for node where the GPU job is running

While the job is running ...

```
[kfotso@xsede.org@login-ci1 workshop_amd_gpu]$ squeue --me
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           5187143  acompile acompile kfotso@x  R      10:03      1 c3cpu-c11-u3-4
           5187117    ami100 sinterac kfotso@x  R      20:16      1 c3gpu-c2-u17
[kfotso@xsede.org@login-ci1 workshop_amd_gpu]$ ssh c3gpu-c2-u17
```

We ssh to the node

While the job is running ...

```
[kfotso@xsede.org@c3gpu-c2-u17 ~]$ watch -n 1 rocm-smi --showmeminfo vram
```

This shows that we are on the gpu node

While the job is running ...

```
[kfotso@xsede.org@c3gpu-c2-u17 ~]$ watch -n 1 rocm-smi --showmeminfo vram
```

rocm-smi is the equivalent of nvidia-smi

While the job is running ...

```
very 1.0s: rocm-smi --showmeminfo vram


=======================ROCm System Management Interface=======================
================================================================================
PU[1]           : vram Total Memory (B): 34342961152
PU[1]           : vram Total Used Memory (B): 2590732288
PU[2]           : vram Total Memory (B): 34342961152
PU[2]           : vram Total Used Memory (B): 7012352
PU[3]           : vram Total Memory (B): 34342961152
PU[3]           : vram Total Used Memory (B): 7012352
================================================================================
============================End of ROCm SMI Log =================================
```

We are on GPU 1 and we can tell that the job is using the GPU memory

Now we run the job with LOG_LEVEL_4

- Only use this if you script is failing
- It is for debugging purpose

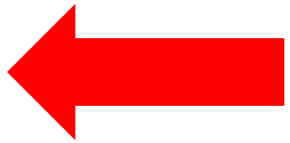
Now we run the job with LOG_LEVEL 4 or 5

- Only use this if you script is failing
- It is for debugging purpose
- Please run script **part3-run_amd_gpu_debug_level4.slurm** from the login node

```
export PYTHONPATH=$PIP_INSTALL_DIR:$PIP_INSTALL_DIR/bin:$PYTHONPATH
export PATH=$PIP_INTALL_DIR:$PATH
#export DEVICE_LIB_PATH=/curc/sw/install/rocm/5.2.3/amdgcn/bitcode
#export ROCM_DEVICE_LIB_PATH=/curc/sw/install/rocm/5.2.3/amdgcn/bitcode

# We need to specify the HIP platform
HIP_PLATFORM=amd
USE_ROCM=1
# We may now install the additional pytorch AMD compatible packages:
export PATH=$PATH:$PIP_INSTALL_DIR/bin

# Running the vision transformer example
# This example shows a simple implementation of Vision Transformer on the CIFAR-10 dataset.
# Original code from https://github.com/pytorch/examples/tree/main/vision_transformer

echo "********************************** Running the models  **********************************"
# We want to make sure that we can profile what is going on
HIP_VISIBLE_DEVICES=0 ROCR_VISIBLE_DEVICES=0 AMD_LOG_LEVEL=5 HSA_ENABLE_SDMA=0 python vision_transformer_example.py
```

Now we run the job with LOG_LEVEL 4 or 5

```
Extracting ./dataset/cifar-10-python.tar.gz to ./dataset
Files already downloaded and verified
:3:hip_device_runtime.cpp   :515 : 10340869225183 us: 3360697: [tid:0x7f790645a280] hipGetDeviceCount ( 0x7ffea9cabfa8 )
:3:hip_device_runtime.cpp   :517 : 10340869225203 us: 3360697: [tid:0x7f790645a280] hipGetDeviceCount: Returned hipSuccess :
:3:hip_device_runtime.cpp   :515 : 10340869225229 us: 3360697: [tid:0x7f790645a280] hipGetDeviceCount ( 0x7f78970657f4 )
:3:hip_device_runtime.cpp   :517 : 10340869225233 us: 3360697: [tid:0x7f790645a280] hipGetDeviceCount: Returned hipSuccess :
:3:hip_device.cpp           :346 : 10340869225240 us: 3360697: [tid:0x7f790645a280] hipGetDeviceProperties ( 0x7ffea9cabcb0, 0 )
:3:hip_device.cpp           :348 : 10340869225248 us: 3360697: [tid:0x7f790645a280] hipGetDeviceProperties: Returned hipSuccess :
:3:hip_device_runtime.cpp   :500 : 10340869225360 us: 3360697: [tid:0x7f790645a280] hipGetDevice ( 0x7ffea9cac2ac )
:3:hip_device_runtime.cpp   :508 : 10340869225367 us: 3360697: [tid:0x7f790645a280] hipGetDevice: Returned hipSuccess :
:3:hip_device_runtime.cpp   :500 : 10340869225406 us: 3360697: [tid:0x7f790645a280] hipGetDevice ( 0x7ffea9cab7ec )
:3:hip_device_runtime.cpp   :508 : 10340869225411 us: 3360697: [tid:0x7f790645a280] hipGetDevice: Returned hipSuccess :
:3:hip_device_runtime.cpp   :500 : 10340869225415 us: 3360697: [tid:0x7f790645a280] hipGetDevice ( 0x7ffea9cab70c )
:3:hip_device_runtime.cpp   :508 : 10340869225419 us: 3360697: [tid:0x7f790645a280] hipGetDevice: Returned hipSuccess :
:3:hip_device_runtime.cpp   :500 : 10340869225424 us: 3360697: [tid:0x7f790645a280] hipGetDevice ( 0x7ffea9cab5bc )
:3:hip_device_runtime.cpp   :508 : 10340869225429 us: 3360697: [tid:0x7f790645a280] hipGetDevice: Returned hipSuccess :
:3:hip_error.cpp            :27  : 10340869225442 us: 3360697: [tid:0x7f790645a280] hipGetLastError (  )
:3:hip_memory.cpp           :493 : 10340869225454 us: 3360697: [tid:0x7f790645a280] hipMalloc ( 0x7ffea9cab138, 2097152 )
:4:rocdevice.cpp            :2054: 10340869225580 us: 3360697: [tid:0x7f790645a280] Allocate hsa device memory 0x7f7649e00000, size 0x200000
:3:rocdevice.cpp            :2093: 10340869225585 us: 3360697: [tid:0x7f790645a280] device=0x55d237eaf930, freeMem_ = 0xfee00000
:3:hip_memory.cpp           :495 : 10340869225593 us: 3360697: [tid:0x7f790645a280] hipMalloc: Returned hipSuccess : 0x7f7649e00000: duration: 139 us
:3:hip_device_runtime.cpp   :530 : 10340869225602 us: 3360697: [tid:0x7f790645a280] hipSetDevice ( 0 )
:3:hip_device_runtime.cpp   :535 : 10340869225606 us: 3360697: [tid:0x7f790645a280] hipSetDevice: Returned hipSuccess :
:3:hip_device_runtime.cpp   :530 : 10340869225609 us: 3360697: [tid:0x7f790645a280] hipSetDevice ( 0 )
:3:hip_device_runtime.cpp   :535 : 10340869225612 us: 3360697: [tid:0x7f790645a280] hipSetDevice: Returned hipSuccess :
:3:hip_device_runtime.cpp   :500 : 10340869225636 us: 3360697: [tid:0x7f790645a280] hipGetDevice ( 0x7ffea9cab54c )
:3:hip_device_runtime.cpp   :508 : 10340869225639 us: 3360697: [tid:0x7f790645a280] hipGetDevice: Returned hipSuccess :
:3:hip_device_runtime.cpp   :500 : 10340869225643 us: 3360697: [tid:0x7f790645a280] hipGetDevice ( 0x7ffea9cab56c )
:3:hip_device_runtime.cpp   :508 : 10340869225644 us: 3360697: [tid:0x7f790645a280] hipGetDevice: Returned hipSuccess :
:3:hip_memory.cpp           :566 : 10340869225661 us: 3360697: [tid:0x7f790645a280] hipMemcpyWithStream ( 0x7f7649e00000, 0x55d238356fc0, 12288, hipMemcpyHostToDevice, stream:<null> )
:3:rocdevice.cpp            :2686: 10340869225670 us: 3360697: [tid:0x7f790645a280] number of allocated hardware queues with low priority: 0, with normal priority: 0, with high priority: 0, maximum
per priority is: 4
```



Hip can see the gpu

Now we run the job with LOG_LEVEL 4 or 5

```
Extracting ./dataset/cifar-10-python.tar.gz to ./dataset
Files already downloaded and verified
:3:hip_device_runtime.cpp   :515 : 10340869225183 us: 3360697: [tid:0x7f790645a280] hipGetDeviceCount ( 0x7ffea9cabfa8 )
:3:hip_device_runtime.cpp   :517 : 10340869225203 us: 3360697: [tid:0x7f790645a280] hipGetDeviceCount: Returned hipSuccess :
:3:hip_device_runtime.cpp   :515 : 10340869225229 us: 3360697: [tid:0x7f790645a280] hipGetDeviceCount ( 0x7f78970657f4 )
:3:hip_device_runtime.cpp   :517 : 10340869225233 us: 3360697: [tid:0x7f790645a280] hipGetDeviceCount: Returned hipSuccess :
:3:hip_device.cpp           :346 : 10340869225240 us: 3360697: [tid:0x7f790645a280] hipGetDeviceProperties ( 0x7ffea9cabcb0
:3:hip_device.cpp           :348 : 10340869225248 us: 3360697: [tid:0x7f790645a280] hipGetDeviceProperties: Returned hipSuc
:3:hip_device_runtime.cpp   :500 : 10340869225360 us: 3360697: [tid:0x7f790645a280] hipGetDevice ( 0x7ffea9cac2ac )
:3:hip_device_runtime.cpp   :508 : 10340869225367 us: 3360697: [tid:0x7f790645a280] hipGetDevice: Returned hipSuccess :
:3:hip_device_runtime.cpp   :500 : 10340869225406 us: 3360697: [tid:0x7f790645a280] hipGetDevice ( 0x7ffea9cab7ec )
:3:hip_device_runtime.cpp   :508 : 10340869225411 us: 3360697: [tid:0x7f790645a280] hipGetDevice: Returned hipSuccess :
:3:hip_device_runtime.cpp   :500 : 10340869225415 us: 3360697: [tid:0x7f790645a280] hipGetDevice ( 0x7ffea9cab70c )
:3:hip_device_runtime.cpp   :508 : 10340869225419 us: 3360697: [tid:0x7f790645a280] hipGetDevice: Returned hipSuccess :
:3:hip_device_runtime.cpp   :500 : 10340869225424 us: 3360697: [tid:0x7f790645a280] hipGetDevice ( 0x7ffea9cab5bc )
:3:hip_device_runtime.cpp   :508 : 10340869225429 us: 3360697: [tid:0x7f790645a280] hipGetDevice: Returned hipSuccess :
:3:hip_error.cpp            :27  : 10340869225442 us: 3360697: [tid:0x7f790645a280] hipGetLastError (  )
:3:hip_memory.cpp           :493 : 10340869225454 us: 3360697: [tid:0x7f790645a280] hipMalloc ( 0x7ffea9cab138, 2097152 )
:4:rocdevice.cpp            :2054: 10340869225580 us: 3360697: [tid:0x7f790645a280] Allocate hsa device memory 0x7f7649e00000, size 0x200000
:3:rocdevice.cpp            :2093: 10340869225585 us: 3360697: [tid:0x7f790645a280] device=0x55d237eaf930, freeMem_ = 0xfee00000
:3:hip_memory.cpp           :495 : 10340869225593 us: 3360697: [tid:0x7f790645a280] hipMalloc: Returned hipSuccess : 0x7f7649e00000: duration: 139 us
:3:hip_device_runtime.cpp   :530 : 10340869225602 us: 3360697: [tid:0x7f790645a280] hipSetDevice ( 0 )
:3:hip_device_runtime.cpp   :535 : 10340869225606 us: 3360697: [tid:0x7f790645a280] hipSetDevice: Returned hipSuccess :
:3:hip_device_runtime.cpp   :530 : 10340869225609 us: 3360697: [tid:0x7f790645a280] hipSetDevice ( 0 )
:3:hip_device_runtime.cpp   :535 : 10340869225612 us: 3360697: [tid:0x7f790645a280] hipSetDevice: Returned hipSuccess :
:3:hip_device_runtime.cpp   :500 : 10340869225636 us: 3360697: [tid:0x7f790645a280] hipGetDevice ( 0x7ffea9cab54c )
:3:hip_device_runtime.cpp   :508 : 10340869225639 us: 3360697: [tid:0x7f790645a280] hipGetDevice: Returned hipSuccess :
:3:hip_device_runtime.cpp   :500 : 10340869225643 us: 3360697: [tid:0x7f790645a280] hipGetDevice ( 0x7ffea9cab56c )
:3:hip_device_runtime.cpp   :508 : 10340869225644 us: 3360697: [tid:0x7f790645a280] hipGetDevice: Returned hipSuccess :
:3:hip_memory.cpp           :566 : 10340869225661 us: 3360697: [tid:0x7f790645a280] hipMemcpyWithStream ( 0x7f7649e00000, 0x55d238356fc0, 12288, hipMemcpyHostToDevice, stream:<null> )
:3:rocdevice.cpp            :2686: 10340869225670 us: 3360697: [tid:0x7f790645a280] number of allocated hardware queues with low priority: 0, with normal priority: 0, with high priority: 0, maximum
per priority is: 4
```

Hip can see the gpu

We get information about runtime libraries, and data movement.

Now we want to introduce rocprof

- You may use rocprof to profile your job as well.
- It will output a file that you can upload into chrome.

Now we want to introduce rocprof

- You may use rocprof to profile your job as well.
- It will output a file that you can upload into chrome.
- Please run script: **part4-run_amd_gpu_debug_rocprof.slurm**

```
[kfotso@xsede.org@c3gpu-c2-u17 workshop_amd_gpu]$ rocprof  --stats --basenames --hip-trace python vision_transformer_example.py
RPL: on '240229_074832' from '/curc/sw/install/rocm/5.2.3' in '/scratch/alpine/kfotso@xsede.org/workshop_amd_gpu'
RPL: profiling '"python" "vision_transformer_example.py"'
RPL: input file ''
RPL: output dir '/tmp/rpl_data_240229_074832_3367074'
RPL: result dir '/tmp/rpl_data_240229_074832_3367074/input_results_240229_074832'
Files already downloaded and verified
Files already downloaded and verified
EPOCH[TRAIN]1/10:   4%|▌         | 486/12500 [00:35<12:48, 15.63it/s, Loss=4.853929]
```

Now we want to introduce rocprof

- You may use rocprof to profile your job as well.
- It will output a file that you can upload into chrome.
- Please run script: **part4-run_amd_gpu_debug_rocprof.slurm**

```
1799 10.972630500793457
1899 10.250618934631348
1999 9.770797729492188
Result: y = 0.00992796290665865 + 0.8281400203704834 x + -0.001712737255729735 x^2 + -0.0892621725
hsa_copy_deps: 0
scan ops data 62009:62010
e.org/workshop_amd_gpu/results.copy_stats.csv' is generating
dump json 2003:2004
File '/scratch/alpine/kfotso@xsede.org/workshop_amd_gpu/results.json' is generating
File '/scratch/alpine/kfotso@xsede.org/workshop_amd_gpu/results.hip_stats.csv' is generating
dump json 628111:628112
File '/scratch/alpine/kfotso@xsede.org/workshop_amd_gpu/results.json' is generating
File '/scratch/alpine/kfotso@xsede.org/workshop_amd_gpu/results.stats.csv' is generating
dump json 60005:60006
File '/scratch/alpine/kfotso@xsede.org/workshop_amd_gpu/results.json' is generating
```

Can upload JSON file into chrome

Let’s run more ambitious models

- Let’s try a more ambitious model
- Change walltime to 01:00:00
- Please run script **part5-run_amd_gpu_debug_vision_example.slurm**
- This example shows a simple implementation of Vision Transformer on the CIFAR-10 dataset.
- Feel free to use rocm-smi

```
# This example shows a simple implementation of Vision Transformer on the CIFAR-10 dataset.
# Original code from https://github.com/pytorch/examples/tree/main/vision_transformer
python vision_transformer_example.py
```

Now let’s see how to build a container that can run on AMD GPU

- Take a look at **part7-run-container.sh**

Now let’s see how to build a container that can run on AMD GPU

- Take a look at **part7-run-container.sh**
- Make sure that your ~/.bashrc does not contain any $TMP_DIR variables
- This script is intended to be run interactively after you’ve logged into a gpu node

sinteractive --partition=atesting_mi100 --qos=testing --time=01:00:00 --gres=gpu:1 --ntasks=4

sinteractive --partition=ami100 --nodes=1 --ntasks=4 --gres=gpu:1 --time=00:10:00 –reservation=amc_workshop

Now let’s see how to build a container that can run on AMD GPU

- Take a look at **part7-run-container.sh**
- Make sure that your ~/.bashrc does not contain any $TMP_DIR variables
- We have already built the container for you so that you do not have to do it

sinteractive --partition=atesting_mi100 --qos=testing --time=01:00:00 --gres=gpu:1 --ntasks=4

sinteractive --partition=ami100 --nodes=1 --ntasks=4 --gres=gpu:1 --time=00:10:00 –reservation=amc_workshop

Now let's see how to build a container that can run on AMD GPU

- |

```
#I am putting in comment the command I used to download and run the container:
# Make sure to comment all the export TMP in your ~/.bashrc
# apptainer build dev-ubuntu-20.04_latest.sif docker://rocm/dev-ubuntu-20.04
# apptainer overlay create --fakeroot --sparse --size 100000 sparse_3_simple_overlay.img
# Make sure to comment all the export TMP in your ~/.bashrc

export ALPINE_SCRATCH=/gpfs/alpine1/scratch/$USER
export APPTAINER_TMPDIR=$ALPINE_SCRATCH/singularity/tmp
export APPTAINER_CACHEDIR=$ALPINE_SCRATCH/singularity/cache
mkdir -pv $APPTAINER_CACHEDIR $APPTAINER_TMPD
```

To build the container we had to create an overlay

Now let's see how to build a container that can run on AMD GPU

- |

```
#I am putting in comment the command I used to download and run the container:
# Make sure to comment all the export TMP in your ~/.bashrc
# apptainer build dev-ubuntu-20.04_latest.sif docker://rocm/dev-ubuntu-20.04
# apptainer overlay create --fakeroot --sparse --size 100000 sparse_3_simple_overlay.img
# Make sure to comment all the export TMP in your ~/.bashrc

export ALPINE_SCRATCH=/gpfs/alpine1/scratch/$USER
export APPTAINER_TMPDIR=$ALPINE_SCRATCH/singularity/tmp
export APPTAINER_CACHEDIR=$ALPINE_SCRATCH/singularity/cache
mkdir -pv $APPTAINER_CACHEDIR $APPTAINER_TMPD
```

An overlay allows to create a filesystem on top of the immutable .sif container

Now let’s see how to build a container that can run on AMD GPU

- |

```
#I am putting in comment the command I used to download and run the container:
# Make sure to comment all the export TMP in your ~/.bashrc
# apptainer build dev-ubuntu-20.04_latest.sif docker://rocm/dev-ubuntu-20.04
# apptainer overlay create --fakeroot --sparse --size 100000 sparse_3_simple_overlay.img
# Make sure to comment all the export TMP in your ~/.bashrc

export ALPINE_SCRATCH=/gpfs/alpine1/scratch/$USER
export APPTAINER_TMPDIR=$ALPINE_SCRATCH/singularity/tmp
export APPTAINER_CACHEDIR=$ALPINE_SCRATCH/singularity/cache
mkdir -pv $APPTAINER_CACHEDIR $APPTAINER_TMPD
```

Make sure to export the apptainer related tmp and cache dir

Now let's see how to build a container that can run on AMD GPU

```
echo "********************************* Copying the .sif and .img files  ************************"
cp /scratch/alpine/kfotso@xsede.org/test_container_docker/sparse_3_simple_overlay.img .
cp /scratch/alpine/kfotso@xsede.org/test_container_docker/build-dev-ubuntu-20.04_latest.sif .
export CONTAIN_DIR=${PWD}

echo "********************************* Running the bui  container ************************"
# --containall is very important as it allows to transf  video permission to the container
# Otherwise you will not see the AMD GPU
#--overlay allows to create a filesystem on top of the  table .sif container
# so that you can modify it at will with the --fakeroot  ion
# To install packages and run pipeline inside the conta  we run:
apptainer shell -H $CONTAIN_DIR --bind=/dev/kfd,/dev/dr  fakeroot --rocm --containall --overlay sparse_3_simple_overlay.img build-dev-ubuntu-20.04_
```

Copy the container related images and overlay to the directory where you cloned the repo

Now let's see how to build a container that can run on AMD GPU

```
echo "********************************** Copying the .sif and .img files  ************************"
cp /scratch/alpine/kfotso@xsede.org/test_container_docker/sparse_3_simple_overlay.img .
cp /scratch/alpine/kfotso@xsede.org/test_container_docker/build-dev-ubuntu-20.04_latest.sif .
export CONTAIN_DIR=${PWD}

echo "**********************************  Running the built container ************************"
# --contai     is very important as it allows to transfer video permission to the container
# Otherwis     u will not see the AMD GPU
#--overlay     ows to create a filesystem on top of the immutable .sif container
# so that      can modify it at will with the --fakeroot option
# To insta     ackages and run pipeline inside the container we run:
apptainer     l -H $CONTAIN_DIR --bind=/dev/kfd,/dev/dri --fakeroot --rocm --containall --overlay sparse_3_simple_overlay.img build-dev-ubuntu-20.04_
```

Export the current directory env

Now let's see how to build a container that can run on AMD GPU

```
echo "********************************** Copying the .sif and .img files  ************************"
cp /scratch/alpine/kfotso@xsede.org/test_container_docker/sparse_3_simple_overlay.img .
cp /scratch/alpine/kfotso@xsede.org/test_container_docker/build-dev-ubuntu-20.04_latest.sif .
export CONTAIN_DIR=${PWD}

echo "********************************** Running the built container ************************"
# --containall is very important as it allows to transfer video permission to the container
# Otherwise you will not see the AMD GPU
#--overlay allows to create a filesystem on top of the immutable .sif container
# so that you can modify it at will with the --fakeroot option
# To install packages and run pipeline inside the container we run:
apptainer shell -H $CONTAIN_DIR --bind=/dev/kfd,/dev/dri --fakeroot --rocm --containall --overlay sparse_3_simple_overlay.img build-dev-ubuntu-20.04_
```

Call the container

Now let's see how to build a container that can run on AMD GPU

- In the apptainer folder go to /usr/bin and run the tensor.py model

```
exit
[kfotso@xsede.org@c3gpu-c2-u17 test_container_docker]$ apptainer shell -H $CONTAIN_DIR --bind=/dev/kfd,/dev/dri --fakeroot --rocm --contain
e_3_simple_overlay.img build-dev-ubuntu-20.04_latest.sif
]INFO:    User not listed in /etc/subuid, trying root-mapped namespace
INFO:    Using fakeroot command combined with root-mapped namespace
WARNING: Skipping /dev/kfd bind mount: already mounted
WARNING: Skipping /dev/dri bind mount: already mounted
INFO:    unknown argument ignored: lazytime
Apptainer> ]     ds
Apptainer>
Apptainer> cp tensors.py /usr/bin/
```

Notes:

- Although the AMD GPU computation has been demonstrated with pytorch, there is a high chance that it can be done with tensorflow as well.
- You can also use other pytorch ROCM compatible versions to install packages: https://pytorch.org/get-started/previous-versions/

Notes:

- Although the AMD GPU computation has been demonstrated with pytorch, there is a high chance that it can be done with tensorflow as well.

- When investigating, we found that Princeton supports it.

What is Horovod? (1)

A distributed deep learning training framework for Tensorflow, Keras, Pytorch, and Apache MXNet.

Originally developed by Uber to train ML models in hours and mins instead of days and weeks.

What is Horovod? (2)

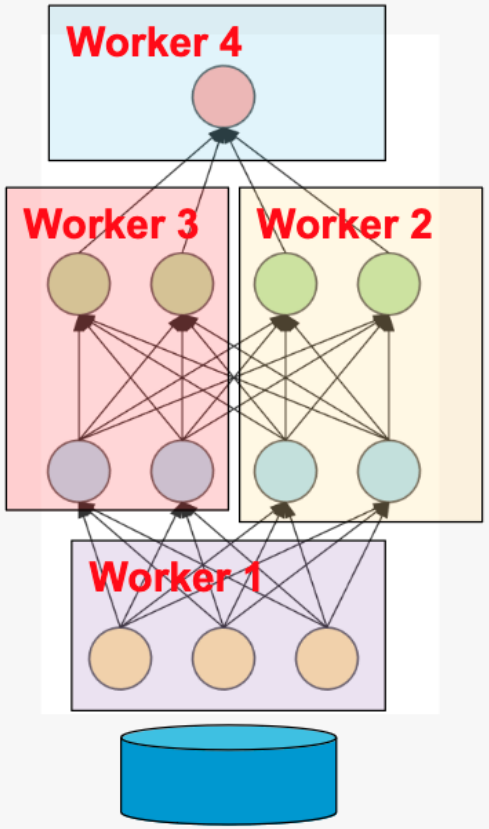
WITH HOROVOD, ML ALGORITHM CAN BE SCALED TO RUN ON 100 OF GPUS WITH JUST FEW LINE OF PYTHON CODE.

PORTABLE, RELATIVELY EASY AND FAST.

IT IS HOSTED UNDER THE LINUX FOUNDATION

Parallelization schemes (1)

Parallelization schemes – Model Parallelism (MP)



Model parallelism

```python
import torch
import torch.nn as nn
import torch.optim as optim


class ToyModel(nn.Module):
    def __init__(self):
        super(ToyModel, self).__init__()
        self.net1 = torch.nn.Linear(10, 10).to('cuda:0')
        self.relu = torch.nn.ReLU()
        self.net2 = torch.nn.Linear(10, 5).to('cuda:1')

    def forward(self, x):
        x = self.relu(self.net1(x.to('cuda:0')))
        return self.net2(x.to('cuda:1'))
```

PyTorch multiple GPU model parallelism within a node

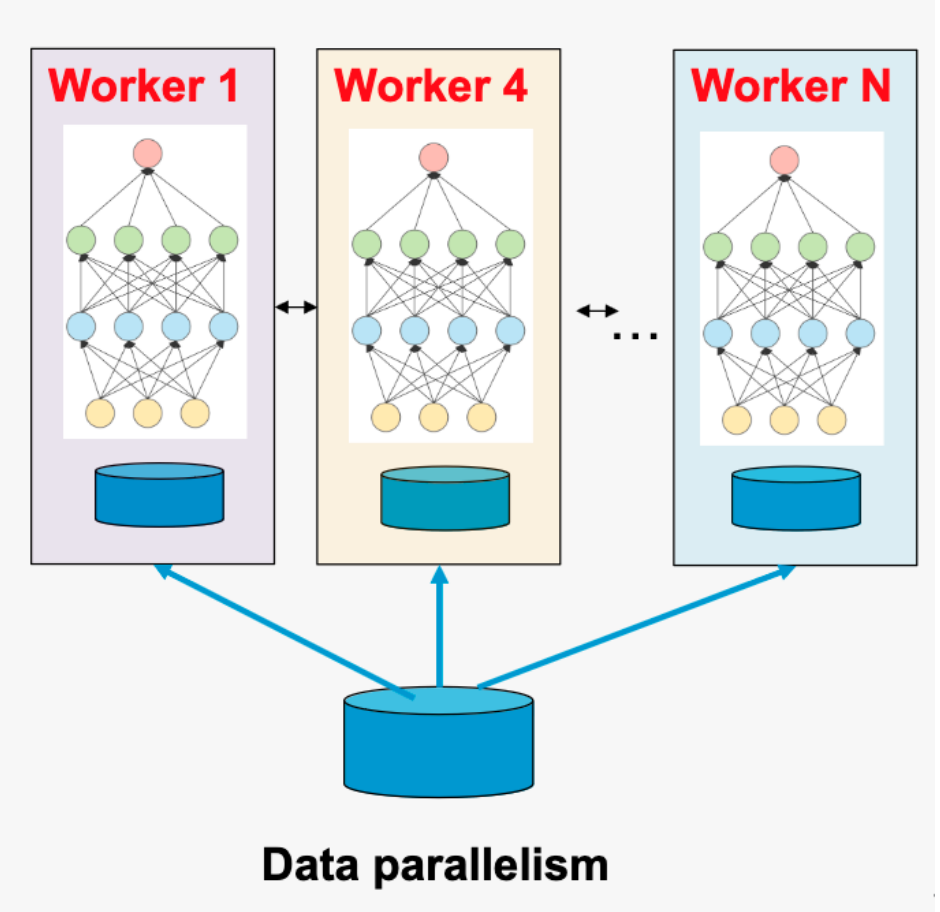
Worker 1

Worker 2

```python
model = ToyModel()
loss_fn = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=0.001)

optimizer.zero_grad()
outputs = model(torch.randn(20, 10))
labels = torch.randn(20, 5).to('cuda:1')
loss_fn(outputs, labels).backward()
optimizer.step()
```

Source: ATPESC 2023, https://extremecomputingtraining.anl.gov/wp-content/uploads/sites/96/2023/08/ATPESC-2023-Track-8-Talk-7-Zheng-Distributed-Deep-Learnng.pdf

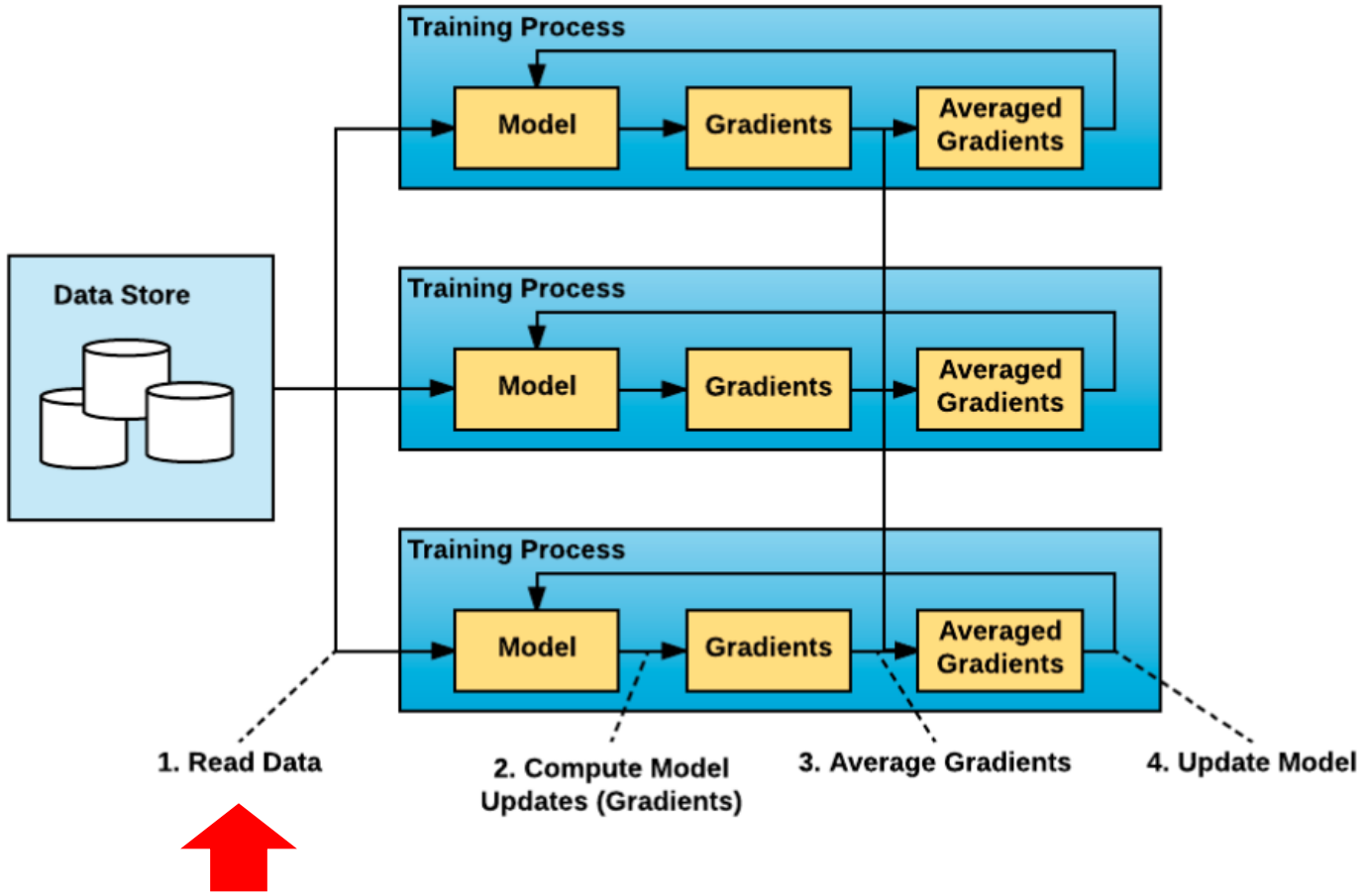
Parallelization schemes (2)



- Data parallelism
- Model is replicated on each worker
- Each worker processes a subset of the minibatch
- Weights are synchronized before updating the model
- Widely supported.

Source: ATPESC 2023, https://extremecomputingtraining.anl.gov/wp-content/uploads/sites/96/2023/08/ATPESC-2023-Track-8-Talk-7-Zheng-Distributed-Deep-Learnng.pdf

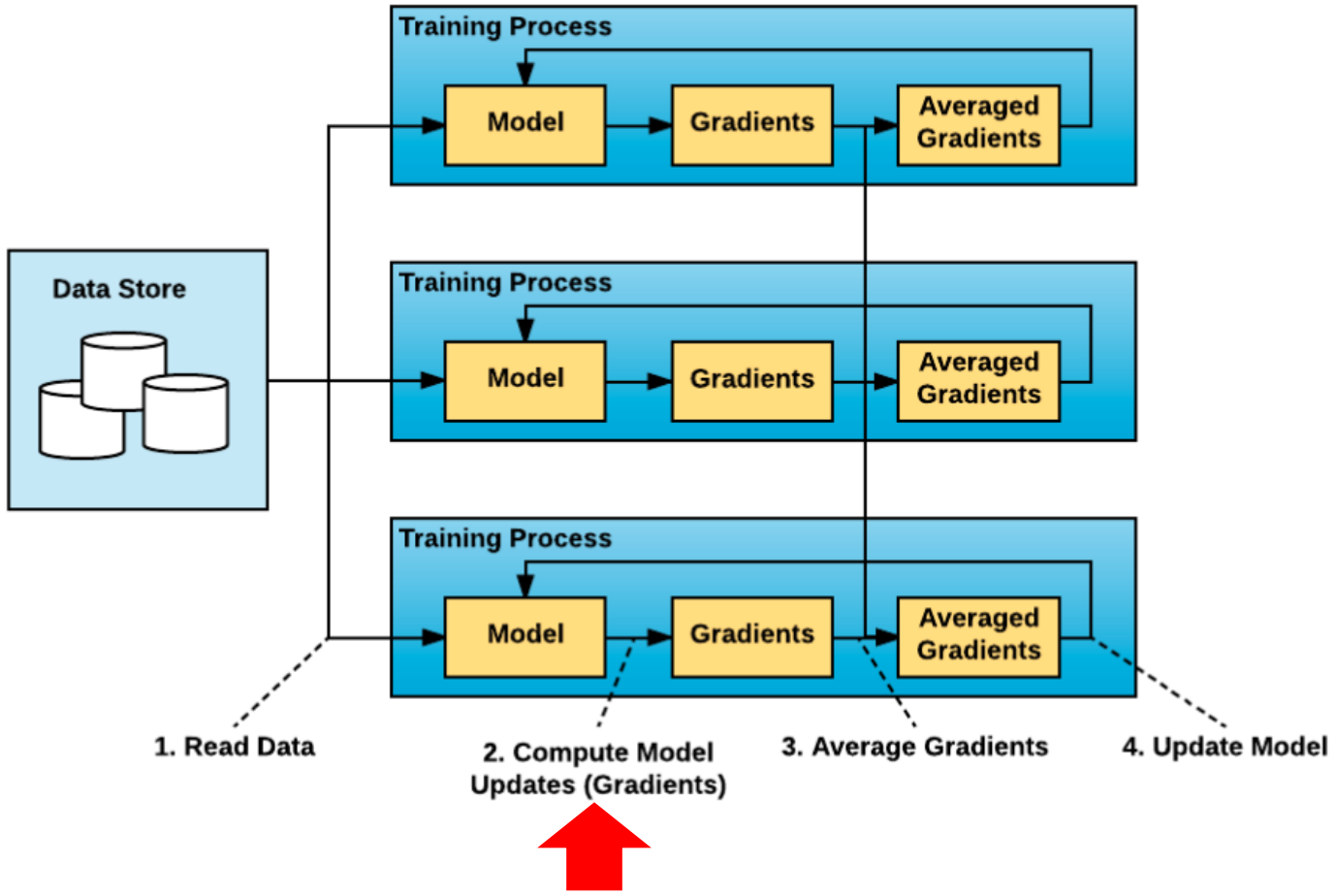
Overview of the Horovod training model



Reads chunk of data (batch)

Source: "Horovod: fast and easy distributed deep learning in Tensorflow" by Sergeev and Balso

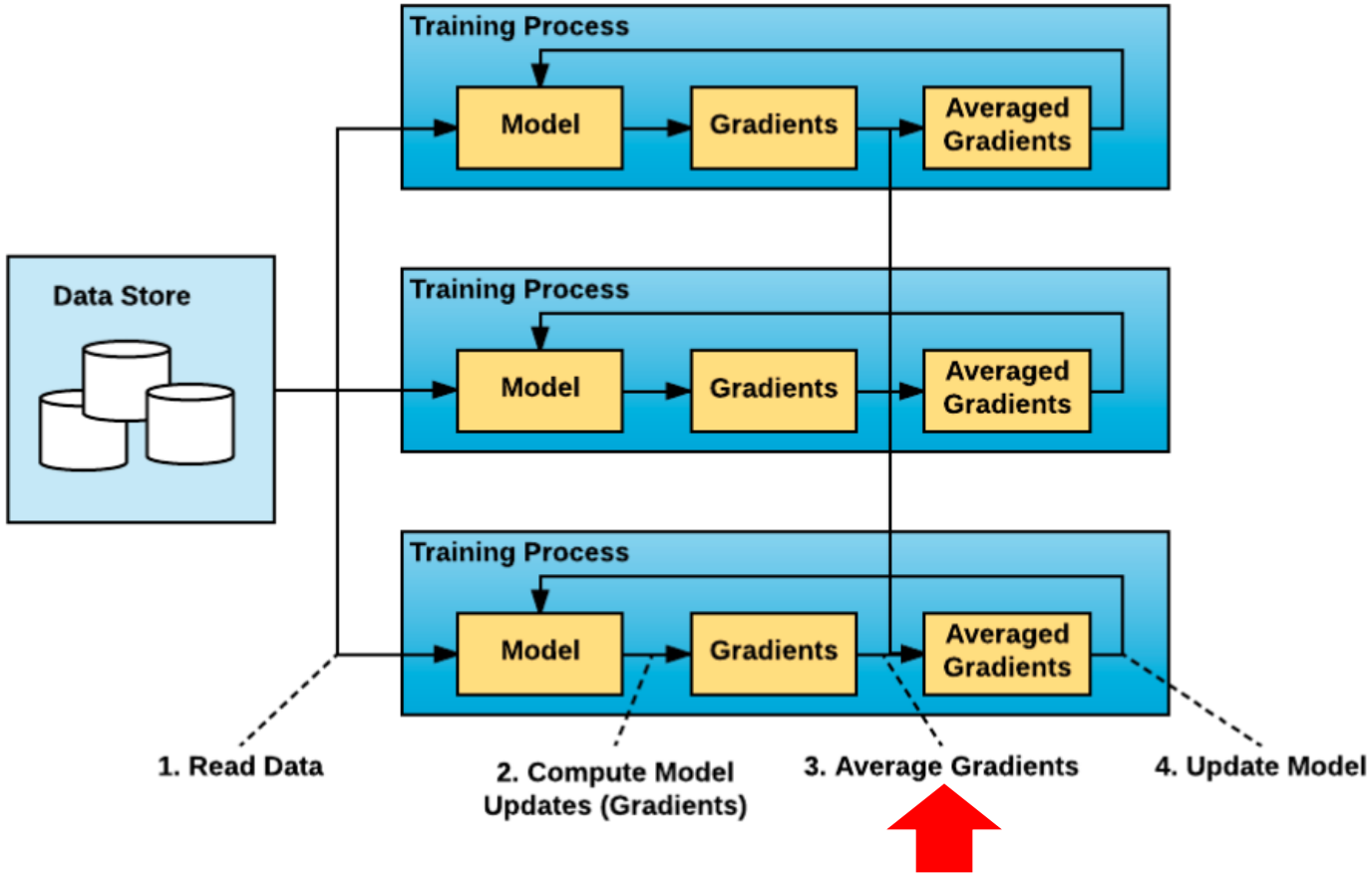
Overview of the Horovod training model (1)



Gradients of different batches calculated separately

Source: "Horovod: fast and easy distributed deep learning in Tensorflow" by Sergeev and Balso

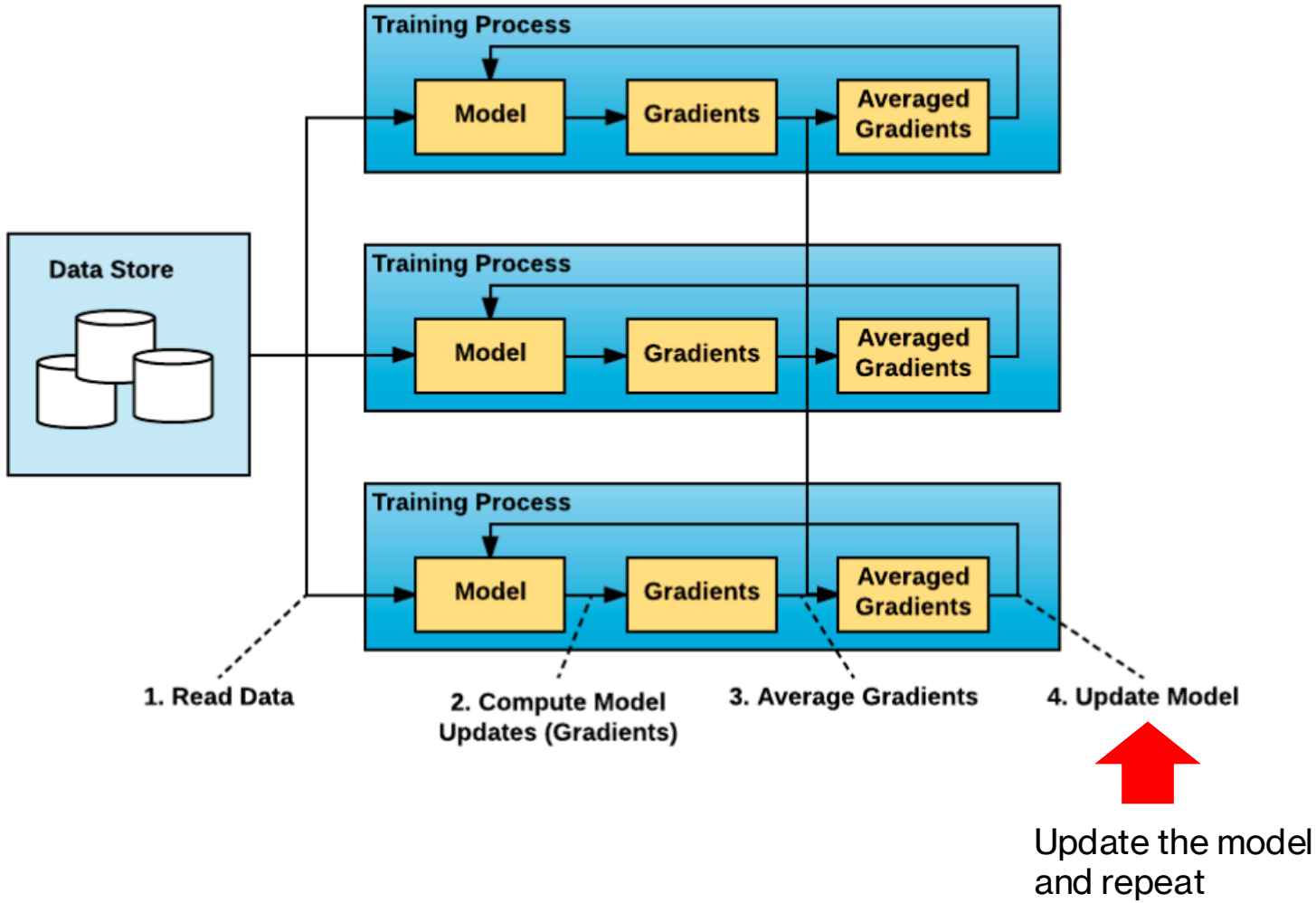
Overview of the Horovod training model(1)



Average across nodes to apply consistent updates to model copy in each node

Source: "Horovod: fast and easy distributed deep learning in Tensorflow" by Sergeev and Balso

Overview of the Horovod training model (1)



Source: "Horovod: fast and easy distributed deep learning in Tensorflow" by Sergeev and Balso

Snippet of Horovod basic code

```
import tensorflow as tf
import horovod.tensorflow as hvd

# Initialize Horovod
hvd.init()

# Pin GPU to be used to process local rank (one GPU per process)
config = tf.ConfigProto()
config.gpu_options.visible_device_list = str(hvd.local_rank())

# Build model...
loss = ...
opt = tf.train.AdagradOptimizer(0.01)

# Add Horovod Distributed Optimizer
opt = hvd.DistributedOptimizer(opt)

# Add hook to broadcast variables from rank 0 to all other processes
# during initialization.
hooks = [hvd.BroadcastGlobalVariablesHook(0)]

# Make training operation
train_op = opt.minimize(loss)

# The MonitoredTrainingSession takes care of session initialization,
# restoring from a checkpoint, saving to a checkpoint, and closing
# when done or an error occurs.
```

Import horovod & tensorflow

Source: "Horovod: fast and easy distributed deep learning in Tensorflow" by Sergeev and Balso

Snippet of Horovod basic code

```
import tensorflow as tf
import horovod.tensorflow as hvd

# Initialize Horovod
hvd.init()

# Pin GPU to be used to process local rank (one GPU per process)
config = tf.ConfigProto()
config.gpu_options.visible_device_list = str(hvd.local_rank())

# Build model...
loss = ...
opt = tf.train.AdagradOptimizer(0.01)

# Add Horovod Distributed Optimizer
opt = hvd.DistributedOptimizer(opt)

# Add hook to broadcast variables from rank 0 to all other processes
# during initialization.
hooks = [hvd.BroadcastGlobalVariablesHook(0)]

# Make training operation
train_op = opt.minimize(loss)

# The MonitoredTrainingSession takes care of session initialization,
# restoring from a checkpoint, saving to a checkpoint, and closing
# when done or an error occurs.
```

Wraps any regular Tensorflow opt with Horovod optimizer which will take care of averaging the gradients automatically

Source: "Horovod: fast and easy distributed deep learning in Tensorflow" by Sergeev and Balso

Snippet of Horovod basic code

```
import tensorflow as tf
import horovod.tensorflow as hvd

# Initialize Horovod
hvd.init()

# Pin GPU to be used to process local rank (one GPU per process)
config = tf.ConfigProto()
config.gpu_options.visible_device_list = str(hvd.local_rank())

# Build model...
loss = ...
opt = tf.train.AdagradOptimizer(0.01)

# Add Horovod Distributed Optimizer
opt = hvd.DistributedOptimizer(opt)

# Add hook to broadcast variables from rank 0 to all other processes
# during initialization.
hooks = [hvd.BroadcastGlobalVariablesHook(0)]

# Make training operation
train_op = opt.minimize(loss)

# The MonitoredTrainingSession takes care of session initialization,
# restoring from a checkpoint, saving to a checkpoint, and closing
# when done or an error occurs.
```

Broadcast variable across all ranks to ensure consistent initialization

Source: "Horovod: fast and easy distributed deep learning in Tensorflow" by Sergeev and Balso

Snippet of Horovod basic code

```
with tf.train.MonitoredTrainingSession(checkpoint_dir="/tmp/train_logs",
                                       config=config,
                                       hooks=hooks) as mon_sess:
 while not mon_sess.should_stop():
   # Perform synchronous training.
   mon_sess.run(train_op)
```

Monitoring session for automatic session init, checkpoint accommodation & closing

Source: "Horovod: fast and easy distributed deep learning in Tensorflow" by Sergeev and Balso

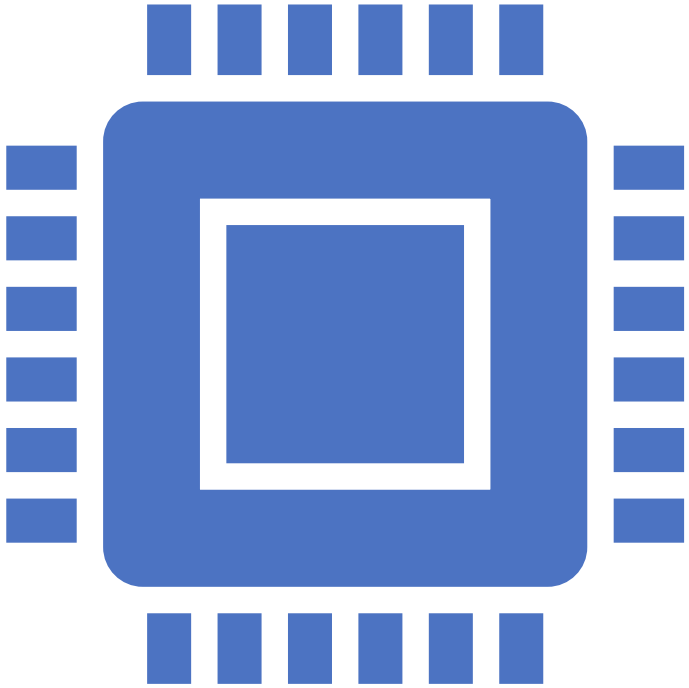
Collective communication with Horovod

- Message Passing Interface (MPI) for CPUs and GPUs in general.

- NVIDIA Collective Communication Library (NCCL) with multi-node & multi GPU.

- GLOO developed by Facebook AI

- Allow to average the gradients automatically (ring-allreduce operation)



Horovod on Alpine (Slurm script)

```
#!/bin/bash

#SBATCH --job-name=horovod_mult_gpu_
#SBATCH --output=horovod_mult_gpu_.%j.out
#SBATCH --error=horovod_mult_gpu_.%j.err
#SBATCH --partition=aa100
#SBATCH --nodes=2
#SBATCH --ntasks=30
#SBATCH --gres=gpu:2
#SBATCH --account=amc-general
#SBATCH --time=00:20:00



ml gcc openmpi/4.1.1
ml cuda/11.8

module load anaconda
conda activate prs_gpu
```

Job name, output and error

Horovod on Alpine (Slurm script)

```
#!/bin/bash

#SBATCH --job-name=horovod_mult_gpu_
#SBATCH --output=horovod_mult_gpu_.%j.out
#SBATCH --error=horovod_mult_gpu_.%j.err
#SBATCH --partition=aa100
#SBATCH --nodes=2
#SBATCH --ntasks=30
#SBATCH --gres=gpu:2
#SBATCH --account=amc-general
#SBATCH --time=00:20:00



ml gcc openmpi/4.1.1
ml cuda/11.8

module load anaconda
conda activate prs_gpu
```

NVIDIA partition name

Horovod on Alpine (Slurm script)

```
#!/bin/bash

#SBATCH --job-name=horovod_mult_gpu_
#SBATCH --output=horovod_mult_gpu_.%j.out
#SBATCH --error=horovod_mult_gpu_.%j.err
#SBATCH --partition=aa100
#SBATCH --nodes=2
#SBATCH --ntasks=30
#SBATCH --gres=gpu:2
#SBATCH --account=amc-general
#SBATCH --time=00:20:00



ml gcc openmpi/4.1.1
ml cuda/11.8

module load anaconda
conda activate prs_gpu
```

We ask for 2 nodes

Horovod on Alpine (Slurm script)

```
#!/bin/bash

#SBATCH --job-name=horovod_mult_gpu_
#SBATCH --output=horovod_mult_gpu_.%j.out
#SBATCH --error=horovod_mult_gpu_.%j.err
#SBATCH --partition=aa100
#SBATCH --nodes=2
#SBATCH --ntasks=30
#SBATCH --gres=gpu:2
#SBATCH --account=amc-general
#SBATCH --time=00:20:00



ml gcc openmpi/4.1.1
ml cuda/11.8

module load anaconda
conda activate prs_gpu
```

We ask for 30 CPU cores total.
NOT GPU cores !!!

Horovod on Alpine (Slurm script)

```
#!/bin/bash

#SBATCH --job-name=horovod_mult_gpu_
#SBATCH --output=horovod_mult_gpu_.%j.out
#SBATCH --error=horovod_mult_gpu_.%j.err
#SBATCH --partition=aa100
#SBATCH --nodes=2
#SBATCH --ntasks=30
#SBATCH --gres=gpu:2
#SBATCH --account=amc-general
#SBATCH --time=00:20:00



ml gcc openmpi/4.1.1
ml cuda/11.8

module load anaconda
conda activate prs_gpu
```

We ask for 2 GPUs per node

Horovod on Alpine (Slurm script)

```
#!/bin/bash

#SBATCH --job-name=horovod_mult_gpu_
#SBATCH --output=horovod_mult_gpu_.%j.out
#SBATCH --error=horovod_mult_gpu_.%j.err
#SBATCH --partition=aa100
#SBATCH --nodes=2
#SBATCH --ntasks=30
#SBATCH --gres=gpu:2
#SBATCH --account=amc-general
#SBATCH --time=00:20:00



ml gcc openmpi/4.1.1
ml cuda/11.8

module load anaconda
conda activate prs_gpu
```

Anschutz users account

Horovod on Alpine (Slurm script)

```
#!/bin/bash

#SBATCH --job-name=horovod_mult_gpu_
#SBATCH --output=horovod_mult_gpu_.%j.out
#SBATCH --error=horovod_mult_gpu_.%j.err
#SBATCH --partition=aa100
#SBATCH --nodes=2
#SBATCH --ntasks=30
#SBATCH --gres=gpu:2
#SBATCH --account=amc-general
#SBATCH --time=00:20:00

ml gcc openmpi/4.1.1
ml cuda/11.8

module load anaconda
conda activate prs_gpu
```

Walltime

Horovod on Alpine (Slurm script)

```
#!/bin/bash

#SBATCH --job-name=horovod_mult_gpu_
#SBATCH --output=horovod_mult_gpu_.%j.out
#SBATCH --error=horovod_mult_gpu_.%j.err
#SBATCH --partition=aa100
#SBATCH --nodes=2
#SBATCH --ntasks=30
#SBATCH --gres=gpu:2
#SBATCH --account=amc-general
#SBATCH --time=00:20:00



ml gcc openmpi/4.1.1
ml cuda/11.8

module load anaconda
conda activate prs_gpu
```

Load gcc to load OpenMPI

Horovod on Alpine (Slurm script)

```
#!/bin/bash

#SBATCH --job-name=horovod_mult_gpu_
#SBATCH --output=horovod_mult_gpu_.%j.out
#SBATCH --error=horovod_mult_gpu_.%j.err
#SBATCH --partition=aa100
#SBATCH --nodes=2
#SBATCH --ntasks=30
#SBATCH --gres=gpu:2
#SBATCH --account=amc-general
#SBATCH --time=00:20:00



ml gcc openmpi/4.1.1
ml cuda/11.8

module load anaconda
conda activate prs_gpu
```

Load cuda to use nsys

Horovod on Alpine (Slurm script)

```
#!/bin/bash

#SBATCH --job-name=horovod_mult_gpu_
#SBATCH --output=horovod_mult_gpu_.%j.out
#SBATCH --error=horovod_mult_gpu_.%j.err
#SBATCH --partition=aa100
#SBATCH --nodes=2
#SBATCH --ntasks=30
#SBATCH --gres=gpu:2
#SBATCH --account=amc-general
#SBATCH --time=00:20:00



ml gcc openmpi/4.1.1
ml cuda/11.8

module load anaconda
conda activate prs_gpu
```

Load anaconda

Horovod on Alpine (Slurm script)

```
#!/bin/bash

#SBATCH --job-name=horovod_mult_gpu_
#SBATCH --output=horovod_mult_gpu_.%j.out
#SBATCH --error=horovod_mult_gpu_.%j.err
#SBATCH --partition=aa100
#SBATCH --nodes=2
#SBATCH --ntasks=30
#SBATCH --gres=gpu:2
#SBATCH --account=amc-general
#SBATCH --time=00:20:00



ml gcc openmpi/4.1.1
ml cuda/11.8

module load anaconda
conda activate prs_gpu
```

Activate environment

Horovod on Alpine (Slurm script)

NCCL library export

```
# We export the NCCL library
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/projects/kefo9343/software/spack/opt/spack/linux-rhel8-zen/gcc-8.4.1/nccl-2.18.3-1-2pmue

# We want the XLA flags and tensorrt for optimization
export XLA_FLAGS=--xla_gpu_cuda_data_dir /projects/kfotso@xsede.org/software/anaconda/envs/prs_gpu

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$CONDA_PREFIX/lib/python3.8/site-packages/tensorrt


# If I want to run on multiple nodes, I will need to know their hostnames
scontrol show hostname > $SLURM_SUBMIT_DIR/nodelist.txt
node1=`sed -n 1p $SLURM_SUBMIT_DIR/nodelist.txt`

# An example in case you want to add a new node, corresponding to the line number
node2=`sed -n 2p $SLURM_SUBMIT_DIR/nodelist.txt`
```

Horovod on Alpine (Slurm script)

```
# We export the NCCL library
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/projects/kefo9343/software/spack/opt/spack/linux-rhel8-zen/gcc-8.4.1/nccl-2.18.3-1-2pmuetqeeecftln5wn6jqy6xcvyzak6d/lib

# We want the XLA flags and tensorrt for optimization
export XLA_FLAGS=--xla_gpu_cuda_data_dir /projects/kfotso@xsede.org/software/anaconda/envs/prs_gpu

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$CONDA_PREFIX/lib/python3.8/site-packages/tensorrt


# If I want to run on multiple nodes, I will need to know their hostnames
scontrol show hostname > $SLURM_SUBMIT_DIR/nodelist.txt
node1=`sed -n 1p $SLURM_SUBMIT_DIR/nodelist.txt`

# An example in case you want to add a new node, corresponding to the line number
node2=`sed -n 2p $SLURM_SUBMIT_DIR/nodelist.txt`
```

XLA flags and tensorRT for optimization

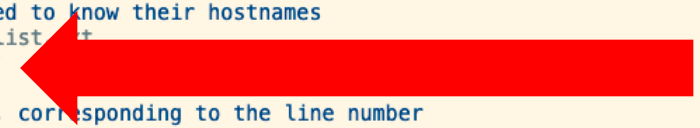
Horovod on Alpine (Slurm script)

```
# We export the NCCL library
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/projects/kefo9343/software/spack/opt/spack/linux-rhel8-zen/gcc-8.4.1/nccl-2.18.3-1-2pmuetqeeecftln5wn6jqy6xcvyzak6d/lib

# We want the XLA flags and tensorrt for optimization
export XLA_FLAGS=--xla_gpu_cuda_data_dir /projects/kfotso@xsede.org/software/anaconda/envs/prs_gpu

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$CONDA_PREFIX/lib/python3.8/site-packages/tensorrt


# If I want to run on multiple nodes, I will need to know their hostnames
scontrol show hostname > $SLURM_SUBMIT_DIR/nodelist.txt
node1=`sed -n 1p $SLURM_SUBMIT_DIR/nodelist.txt`

# An example in case you want to add a new node, corresponding to the line number
node2=`sed -n 2p $SLURM_SUBMIT_DIR/nodelist.txt`
```

We create a nodelist with scontrol that we store in nodelist.txt

Horovod on Alpine (Slurm script)

```
# We export the NCCL library
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/projects/kefo9343/software/spack/opt/spack/linux-rhel8-zen/gcc-8.4.1/nccl-2.18.3-1-2pmuetqeeecftln5wn6jqy6xcvyzak6d/lib

# We want the XLA flags and tensorrt for optimization
export XLA_FLAGS=--xla_gpu_cuda_data_dir /projects/kfotso@xsede.org/software/anaconda/envs/prs_gpu

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$CONDA_PREFIX/lib/python3.8/site-packages/tensorrt


# If I want to run on multiple nodes, I will need to know their hostnames
scontrol show hostname > $SLURM_SUBMIT_DIR/nodelist.txt
node1=`sed -n 1p $SLURM_SUBMIT_DIR/nodelist.txt`

# An example in case you want to add a new node, corresponding to the line number
node2=`sed -n 2p $SLURM_SUBMIT_DIR/nodelist.txt`
```



We assign the content of nodelist.txt into variables

Horovod on Alpine (Basic Neural Net)

- California housing dataset
- Target variable is the median house value for California districts
- Simple regression neural network
- Code adapted from the Hands On Machine learning book chapter 10.

Horovod on Alpine (Python script)

```
# This horovod test comes from the original chapter 10 of Hands on Machine learning
# https://github.com/ageron/handson-ml3/blob/main/10_neural_nets_with_keras.ipynb
print(4)
# Get the current timestamp
timestamp1 = time.time()

# Horovod: initialize Horovod.
hvd.init()

# Selecting the GPU devices on the node
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)
if gpus:
    tf.config.experimental.set_visible_devices(gpus[hvd.local_rank()], 'GPU')
```

Timestamp to measure the duration

Horovod on Alpine (Python script)

```
# This horovod test comes from the original chapter 10 of Hands on Machine learning
# https://github.com/ageron/handson-ml3/blob/main/10_neural_nets_with_keras.ipynb
print(4)
# Get the current timestamp
timestamp1 = time.time()

# Horovod: initialize Horovod.
hvd.init()

# Selecting the GPU devices on the node
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)
if gpus:
    tf.config.experimental.set_visible_devices(gpus[hvd.local_rank()], 'GPU')
```

Initialize Horovod
and set multi GPUs

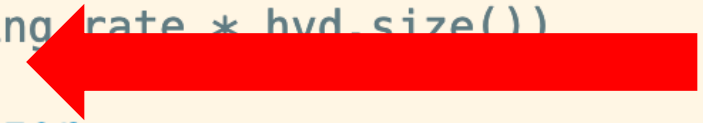
Horovod on Alpine (Python script)

```
tf.random.set_seed(42)
norm_layer = tf.keras.layers.Normalization(input_shape=X_train.shape[1:])
model = tf.keras.Sequential([
    norm_layer,
    tf.keras.layers.Dense(50, activation="relu"),
    tf.keras.layers.Dense(50, activation="relu"),
    tf.keras.layers.Dense(50, activation="relu"),
    tf.keras.layers.Dense(1)
])
```

Simple Neural Net

Horovod on Alpine (Python script)

```
# Tune the learning rate
learning_rate =  1e-3

#K.set_value(model.optimizer.learning_rate, learning_rate)
# Horovod: adjust learning rate based on number of GPUs.
opt = tf.keras.optimizers.Adadelta(learning_rate * hvd.size())

# Horovod: add Horovod Distributed Optimizer.
opt = hvd.DistributedOptimizer(opt)# Horovod: add Horovod Distributed Optimizer.

model.compile(loss="mse", optimizer=opt, metrics=["RootMeanSquaredError"])
norm_layer.adapt(X_train)

# Defining batch size and epochs as a function of horovod
batch_size = 8
```



Learning rate is scaled by the number of GPUs

Horovod on Alpine (Python script)

```
total_epochs_ = int(math.ceil(6.0 / hvd.size()))


callbacks = [
    # Horovod: broadcast initial variable states from rank 0 to all other processes.
    # This is necessary to ensure consistent initialization of all workers when
    # training is started with random weights or restored from a checkpoint.
    hvd.callbacks.BroadcastGlobalVariablesCallback(0),
]

# Horovod: save checkpoints only on worker 0 to prevent other workers from corrupting them.
if hvd.rank() == 0:
    callbacks.append(tf.keras.callbacks.ModelCheckpoint('./checkpoint-{epoch}.h5'))


# Model fitting across multiple GPUs
model.fit(X_train,
          y_train,
          epoch=total_epochs_,
          verbose=1,
          callbacks=callbacks,
          validation_data=(X_valid, y_valid))
```

We scale the number of epoch by the number of GPUs

Horovod on Alpine (Python script)

```
total_epochs_ = int(math.ceil(6.0 / hvd.size()))


callbacks = [
    # Horovod: broadcast initial variable states from rank 0 to all other processes.
    # This is necessary to ensure consistent initialization of all workers when
    # training is started with random weights or restored from a checkpoint.
    hvd.callbacks.BroadcastGlobalVariablesCallback(0),
]

# Horovod: save checkpoints only on worker 0 to prevent other wor
if hvd.rank() == 0:
    callbacks.append(tf.keras.callbacks.ModelCheckpoint('./checkpoint-{epoch}.h5'))


# Model fitting across multiple GPUs
model.fit(X_train,
          y_train,
          epoch=total_epochs_,
          verbose=1,
          callbacks=callbacks,
          validation_data=(X_valid, y_valid))
```

checkpointing

Run slurm script

- sbatch **horovod_mpi.sh**
- Check the nodes where the job is running

```
e.org@login-ci2 (                                    ]$ squeue --me
 JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
481362     aa100 horovod_ kfotso@x  R       2:34      2 c3gpu-a9-u31-1,c3gpu-a
479946  acompile acompile kfotso@x  R    2:03:48      1 c3cpu-c9-u5-2
481778 atesting_ sinterac kfotso@x  R       7:56      1 c3gpu-a9-u29-1
e.org@login-ci2                                      ]$ ssh c3gpu-a9-u29-1^C
```

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

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 |
+-----------------------------------------------------------------------------+
```

We can confirm that we are using 2 GPUs!!!

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 |
+-----------------------------------------------------------------------------+
```

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.

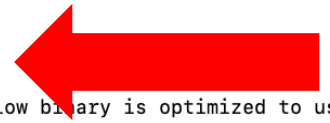
GPU monitoring

- 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 %
```

Slurm job (2 GPUs)

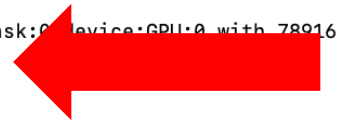


The profiling sees the 2 GPUs.

```
tput=horovod_profile horovodrun -np 2 -H localhost:2 horovod_cnn_mnist.py
GPU 0: General Metrics for NVIDIA GA100 (any frequency)
GPU 1: General Metrics for NVIDIA GA100 (any frequency)
2023-10-23 21:47:46.810582: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critic
al operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
[1,0]<stderr>:2023-10-23 21:47:52.457092: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in perf
ormance-critical operations.
[1,0]<stderr>:To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
[1,1]<stderr>:2023-10-23 21:47:52.485600: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in perf
ormance-critical operations.
[1,1]<stderr>:To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
[1,1]<stdout>:4
[1,0]<stdout>:4
[1,0]<stderr>:2023-10-23 21:47:57.693575: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1635] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 78916 MB
 memory:  -> device: 0, name: NVIDIA A100 80GB PCIe, pci bus id: 0000:25:00.0, compute capability: 8.0
[1,1]<stderr>:2023-10-23 21:47:57.708825: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1635] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 78916 MB
 memory:  -> device: 1, name: NVIDIA A100 80GB PCIe, pci bus id: 0000:81:00.0, compute capability: 8.0
[1,0]<stdout>:Epoch 1/3
[1,1]<stdout>:Epoch 1/3
[1,0]<stderr>:2023-10-23 21:48:01.104191: I tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.cc:424] Loaded cuDNN version 8600
[1,1]<stderr>:2023-10-23 21:48:01.117719: I tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.cc:424] Loaded cuDNN version 8600
[1,0]<stderr>:2023-10-23 21:48:01.793663: I tensorflow/compiler/xla/stream_executor/cuda/cuda_blas.cc:637] TensorFloat-32 will be used for the matrix multiplication. This w
ill only be logged once.
[1,1]<stderr>:2023-10-23 21:48:01.813954: I tensorflow/compiler/xla/stream_executor/cuda/cuda_blas.cc:637] TensorFloat-32 will be used for the matrix multiplication. This w
ill only be logged once.
[1,1]<stderr>:2023-10-23 21:48:01.846349: I tensorflow/compiler/xla/service/service.cc:169] XLA service 0x7f6a85723010 initialized for platform CUDA (this does not guarante
e that XLA will be used). Devices:
[1,1]<stderr>:2023-10-23 21:48:01.846381: I tensorflow/compiler/xla/service/service.cc:177]   StreamExecutor device (0): NVIDIA A100 80GB PCIe, Compute Capability 8.0
[1,0]<stderr>:2023-10-23 21:48:01.854430: I tensorflow/compiler/xla/service/service.cc:169] XLA service 0x7f0deed4b800 initialized for platform CUDA (this does not guarante
e that XLA will be used). Devices:
[1,0]<stderr>:2023-10-23 21:48:01.854486: I tensorflow/compiler/xla/service/service.cc:177]   StreamExecutor device (0): NVIDIA A100 80GB PCIe, Compute Capability 8.0
[1,1]<stderr>:2023-10-23 21:48:01.857037: I tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:269] disabling MLIR crash reproducer, set env var `MLIR_CRASH_REPROD
UCER_DIRECTORY` to enable.
[1,0]<stderr>:2023-10-23 21:48:01.861230: I tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:269] disabling MLIR crash reproducer, set env var `MLIR_CRASH_REPROD
UCER_DIRECTORY` to enable.
[1,1]<stderr>:2023-10-23 21:48:02.027518: I ./tensorflow/compiler/jit/device_compiler.h:180] Compiled cluster using XLA!  This line is logged at most once for the lifetime
of the process.
[1,0]<stderr>:2023-10-23 21:48:02.034153: I ./tensorflow/compiler/jit/device_compiler.h:180] Compiled cluster using XLA!  This line is logged at most once for the lifetime
of the process.
```

Slurm job (2 GPUs)

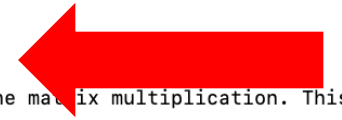
```
tput=horovod_profile horovodrun -np 2 -H localhost:2 horovod_cnn_mnist.py
GPU 0: General Metrics for NVIDIA GA100 (any frequency)
GPU 1: General Metrics for NVIDIA GA100 (any frequency)
2023-10-23 21:47:46.810582: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critic
al operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
[1,0]<stderr>:2023-10-23 21:47:52.457092: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in perf
ormance-critical operations.
[1,0]<stderr>:To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
[1,1]<stderr>:2023-10-23 21:47:52.485600: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in perf
ormance-critical operations.
[1,1]<stderr>:To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
[1,1]<stdout>:4
[1,0]<stdout>:4
[1,0]<stderr>:2023-10-23 21:47:57.693575: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1635] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 78916 MB
 memory:  -> device: 0, name: NVIDIA A100 80GB PCIe, pci bus id: 0000:25:00.0, compute capability: 8.0
[1,1]<stderr>:2023-10-23 21:47:57.708825: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1635] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 78916 MB
 memory:  -> device: 1, name: NVIDIA A100 80GB PCIe, pci bus id: 0000:81:00.0, compute capability: 8.0
[1,0]<stdout>:Epoch 1/3
[1,1]<stdout>:Epoch 1/3
[1,0]<stderr>:2023-10-23 21:48:01.104191: I tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.cc:424] Loaded cuDNN version 8600
[1,1]<stderr>:2023-10-23 21:48:01.117719: I tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.cc:424] Loaded cuDNN version 8600
[1,0]<stderr>:2023-10-23 21:48:01.793663: I tensorflow/compiler/xla/stream_executor/cuda/cuda_blas.cc:637] TensorFloat-32 will be used for the matrix multiplication. This w
ill only be logged once.
[1,1]<stderr>:2023-10-23 21:48:01.813954: I tensorflow/compiler/xla/stream_executor/cuda/cuda_blas.cc:637] TensorFloat-32 will be used for the matrix multiplication. This w
ill only be logged once.
[1,1]<stderr>:2023-10-23 21:48:01.846349: I tensorflow/compiler/xla/service/service.cc:169] XLA service 0x7f6a85723010 initialized for platform CUDA (this does not guarante
e that XLA will be used). Devices:
[1,1]<stderr>:2023-10-23 21:48:01.846381: I tensorflow/compiler/xla/service/service.cc:177]   StreamExecutor device (0): NVIDIA A100 80GB PCIe, Compute Capability 8.0
[1,0]<stderr>:2023-10-23 21:48:01.854430: I tensorflow/compiler/xla/service/service.cc:169] XLA service 0x7f0deed4b800 initialized for platform CUDA (this does not guarante
e that XLA will be used). Devices:
[1,0]<stderr>:2023-10-23 21:48:01.854486: I tensorflow/compiler/xla/service/service.cc:177]   StreamExecutor device (0): NVIDIA A100 80GB PCIe, Compute Capability 8.0
[1,1]<stderr>:2023-10-23 21:48:01.857037: I tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:269] disabling MLIR crash reproducer, set env var `MLIR_CRASH_REPROD
UCER_DIRECTORY` to enable.
[1,0]<stderr>:2023-10-23 21:48:01.861230: I tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:269] disabling MLIR crash reproducer, set env var `MLIR_CRASH_REPROD
UCER_DIRECTORY` to enable.
[1,1]<stderr>:2023-10-23 21:48:02.027518: I ./tensorflow/compiler/jit/device_compiler.h:180] Compiled cluster using XLA!  This line is logged at most once for the lifetime
of the process.
[1,0]<stderr>:2023-10-23 21:48:02.034153: I ./tensorflow/compiler/jit/device_compiler.h:180] Compiled cluster using XLA!  This line is logged at most once for the lifetime
of the process.
```



2 GPU got assigned

Slurm job (2 GPUs)

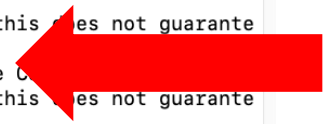
```
tput=horovod_profile horovodrun -np 2 -H localhost:2 horovod_cnn_mnist.py
GPU 0: General Metrics for NVIDIA GA100 (any frequency)
GPU 1: General Metrics for NVIDIA GA100 (any frequency)
2023-10-23 21:47:46.810582: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critic
al operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
[1,0]<stderr>:2023-10-23 21:47:52.457092: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in perf
ormance-critical operations.
[1,0]<stderr>:To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
[1,1]<stderr>:2023-10-23 21:47:52.485600: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in perf
ormance-critical operations.
[1,1]<stderr>:To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
[1,1]<stdout>:4
[1,0]<stdout>:4
[1,0]<stderr>:2023-10-23 21:47:57.693575: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1635] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 78916 MB
 memory:  -> device: 0, name: NVIDIA A100 80GB PCIe, pci bus id: 0000:25:00.0, compute capability: 8.0
[1,1]<stderr>:2023-10-23 21:47:57.708825: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1635] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 78916 MB
 memory:  -> device: 1, name: NVIDIA A100 80GB PCIe, pci bus id: 0000:81:00.0, compute capability: 8.0
[1,0]<stdout>:Epoch 1/3
[1,1]<stdout>:Epoch 1/3
[1,0]<stderr>:2023-10-23 21:48:01.104191: I tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.cc:424] Loaded cuDNN version 8600
[1,1]<stderr>:2023-10-23 21:48:01.117719: I tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.cc:424] Loaded cuDNN version 8600
[1,0]<stderr>:2023-10-23 21:48:01.793663: I tensorflow/compiler/xla/stream_executor/cuda/cuda_blas.cc:637] TensorFloat-32 will be used for the matrix multiplication. This w
ill only be logged once.
[1,1]<stderr>:2023-10-23 21:48:01.813954: I tensorflow/compiler/xla/stream_executor/cuda/cuda_blas.cc:637] TensorFloat-32 will be used for the matrix multiplication. This w
ill only be logged once.
[1,1]<stderr>:2023-10-23 21:48:01.846349: I tensorflow/compiler/xla/service/service.cc:169] XLA service 0x7f6a85723010 initialized for platform CUDA (this does not guarante
e that XLA will be used). Devices:
[1,1]<stderr>:2023-10-23 21:48:01.846381: I tensorflow/compiler/xla/service/service.cc:177]   StreamExecutor device (0): NVIDIA A100 80GB PCIe, Compute Capability 8.0
[1,0]<stderr>:2023-10-23 21:48:01.854430: I tensorflow/compiler/xla/service/service.cc:169] XLA service 0x7f0deed4b800 initialized for platform CUDA (this does not guarante
e that XLA will be used). Devices:
[1,0]<stderr>:2023-10-23 21:48:01.854486: I tensorflow/compiler/xla/service/service.cc:177]   StreamExecutor device (0): NVIDIA A100 80GB PCIe, Compute Capability 8.0
[1,1]<stderr>:2023-10-23 21:48:01.857037: I tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:269] disabling MLIR crash reproducer, set env var `MLIR_CRASH_REPROD
UCER_DIRECTORY` to enable.
[1,0]<stderr>:2023-10-23 21:48:01.861230: I tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:269] disabling MLIR crash reproducer, set env var `MLIR_CRASH_REPROD
UCER_DIRECTORY` to enable.
[1,1]<stderr>:2023-10-23 21:48:02.027518: I ./tensorflow/compiler/jit/device_compiler.h:180] Compiled cluster using XLA!  This line is logged at most once for the lifetime
of the process.
[1,0]<stderr>:2023-10-23 21:48:02.034153: I ./tensorflow/compiler/jit/device_compiler.h:180] Compiled cluster using XLA!  This line is logged at most once for the lifetime
of the process.
```



CUDNN gets loaded for each

Slurm job (2 GPUs)

```
tput=horovod_profile horovodrun -np 2 -H localhost:2 horovod_cnn_mnist.py
GPU 0: General Metrics for NVIDIA GA100 (any frequency)
GPU 1: General Metrics for NVIDIA GA100 (any frequency)
2023-10-23 21:47:46.810582: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critic
al operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
[1,0]<stderr>:2023-10-23 21:47:52.457092: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in perf
ormance-critical operations.
[1,0]<stderr>:To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
[1,1]<stderr>:2023-10-23 21:47:52.485600: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in perf
ormance-critical operations.
[1,1]<stderr>:To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
[1,1]<stdout>:4
[1,0]<stdout>:4
[1,0]<stderr>:2023-10-23 21:47:57.693575: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1635] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 78916 MB
 memory:  -> device: 0, name: NVIDIA A100 80GB PCIe, pci bus id: 0000:25:00.0, compute capability: 8.0
[1,1]<stderr>:2023-10-23 21:47:57.708825: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1635] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 78916 MB
 memory:  -> device: 1, name: NVIDIA A100 80GB PCIe, pci bus id: 0000:81:00.0, compute capability: 8.0
[1,0]<stdout>:Epoch 1/3
[1,1]<stdout>:Epoch 1/3
[1,0]<stderr>:2023-10-23 21:48:01.104191: I tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.cc:424] Loaded cuDNN version 8600
[1,1]<stderr>:2023-10-23 21:48:01.117719: I tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.cc:424] Loaded cuDNN version 8600
[1,0]<stderr>:2023-10-23 21:48:01.793663: I tensorflow/compiler/xla/stream_executor/cuda/cuda_blas.cc:637] TensorFloat-32 will be used for the matrix multiplication. This w
ill only be logged once.
[1,1]<stderr>:2023-10-23 21:48:01.813954: I tensorflow/compiler/xla/stream_executor/cuda/cuda_blas.cc:637] TensorFloat-32 will be used for the matrix multiplication. This w
ill only be logged once.
[1,1]<stderr>:2023-10-23 21:48:01.846349: I tensorflow/compiler/xla/service/service.cc:169] XLA service 0x7f6a85723010 initialized for platform CUDA (this does not guarante
e that XLA will be used). Devices:
[1,1]<stderr>:2023-10-23 21:48:01.846381: I tensorflow/compiler/xla/service/service.cc:177]   StreamExecutor device (0): NVIDIA A100 80GB PCIe, Compute C
[1,0]<stderr>:2023-10-23 21:48:01.854430: I tensorflow/compiler/xla/service/service.cc:169] XLA service 0x7f0deed4b800 initialized for platform CUDA (this does not guarante
e that XLA will be used). Devices:
[1,0]<stderr>:2023-10-23 21:48:01.854486: I tensorflow/compiler/xla/service/service.cc:177]   StreamExecutor device (0): NVIDIA A100 80GB PCIe, Compute Capability 8.0
[1,1]<stderr>:2023-10-23 21:48:01.857037: I tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:269] disabling MLIR crash reproducer, set env var `MLIR_CRASH_REPROD
UCER_DIRECTORY` to enable.
[1,0]<stderr>:2023-10-23 21:48:01.861230: I tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:269] disabling MLIR crash reproducer, set env var `MLIR_CRASH_REPROD
UCER_DIRECTORY` to enable.
[1,1]<stderr>:2023-10-23 21:48:02.027518: I ./tensorflow/compiler/jit/device_compiler.h:180] Compiled cluster using XLA!  This line is logged at most once for the lifetime
of the process.
[1,0]<stderr>:2023-10-23 21:48:02.034153: I ./tensorflow/compiler/jit/device_compiler.h:180] Compiled cluster using XLA!  This line is logged at most once for the lifetime
of the process.
```



XLA opt is initialize and the stream is executed

Special Thanks

- Special thanks to Brandon Reyes from the Boulder team for his contribution on the container aspect of AMD GPU computation

Questions?