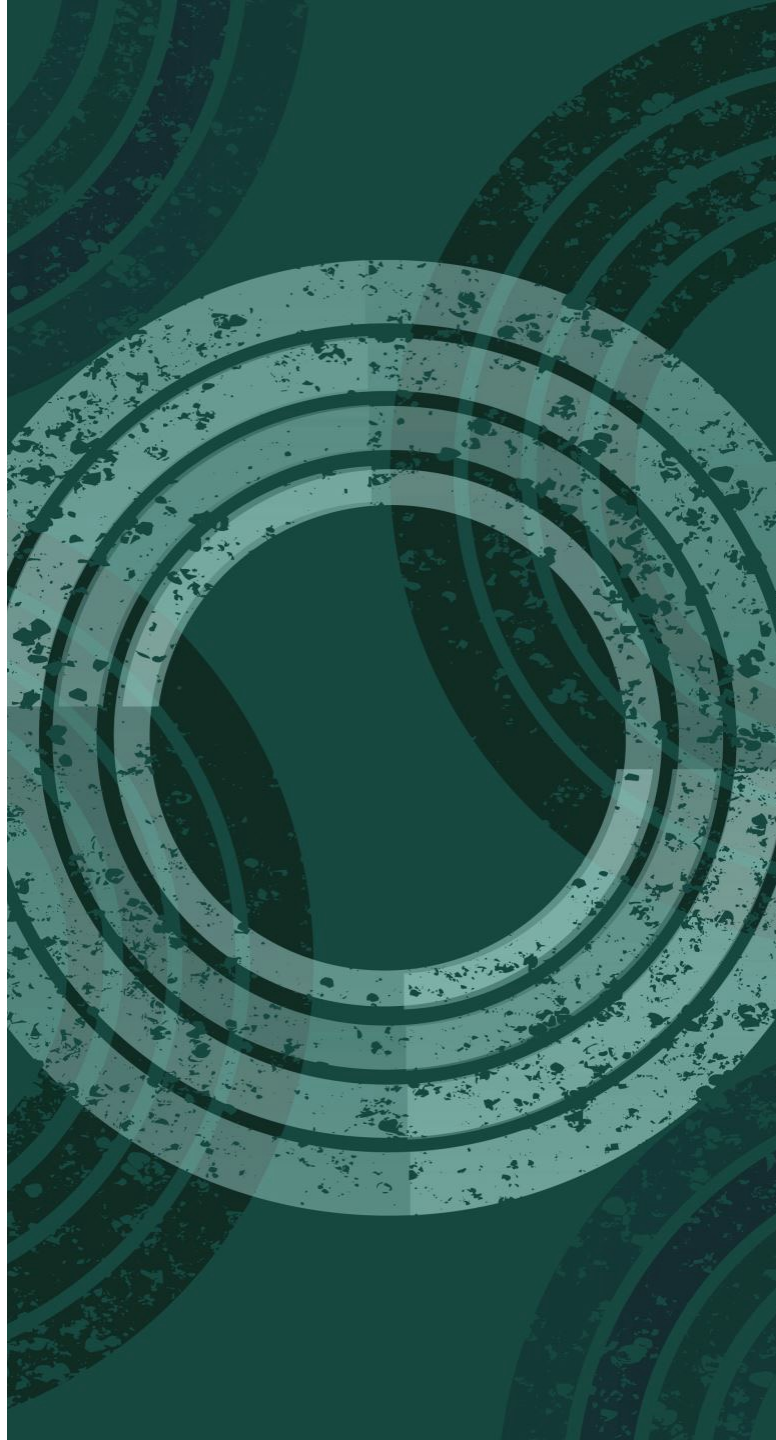
INTRODUCTION TO GPU COMPUTING ON AMD GPU

By Kevin Fotso



Before we start!!!

- Make sure to log into the AMD GPU node with the following:

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

What are GPUs?

- GPU are graphics processing units.

What are GPUs?

- GPU are graphics processing units.

- Originally was mainly used in the video game and cinema industry.

What are GPUs?

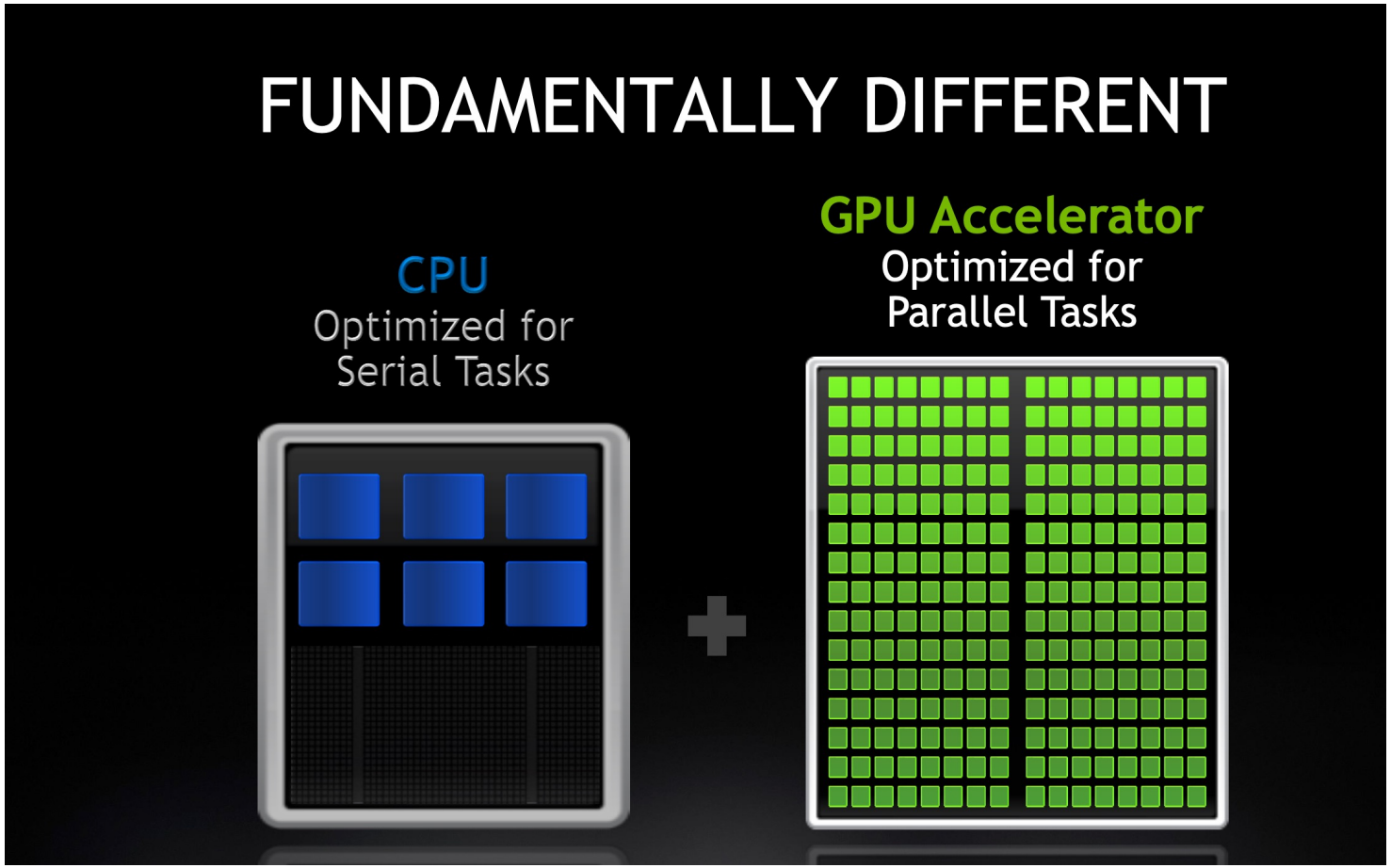
- GPU are graphics processing units.

- Originally was mainly used in the video game and cinema industry.

Jurassic Park

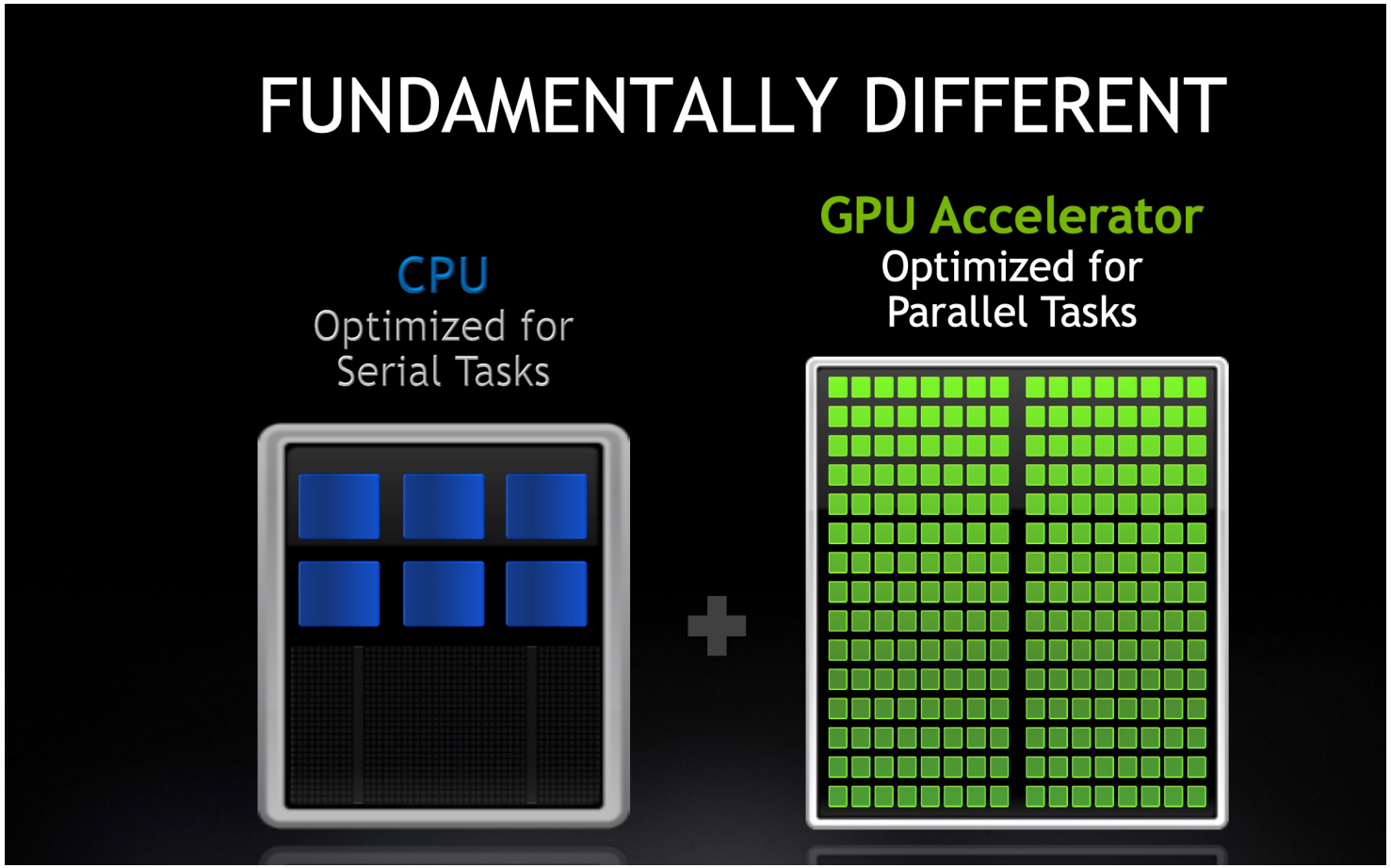
Zelda Ocarina of Time

CPUs vs GPUs comparison



Source: Openhackaton @ Princeton 2023 “Introduction to GPU computing

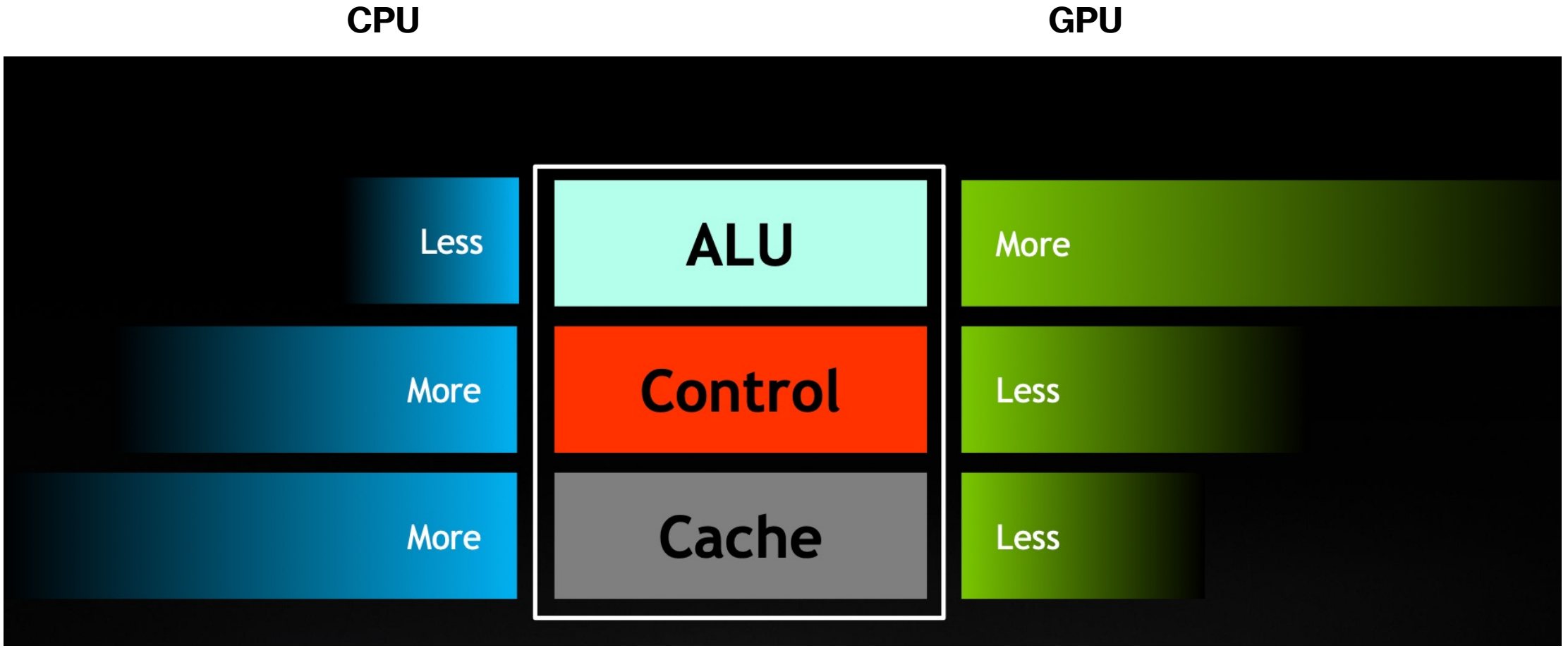
CPUs vs GPUs comparison



GPU follow SIMT pattern (Single Instruction multiple threads)

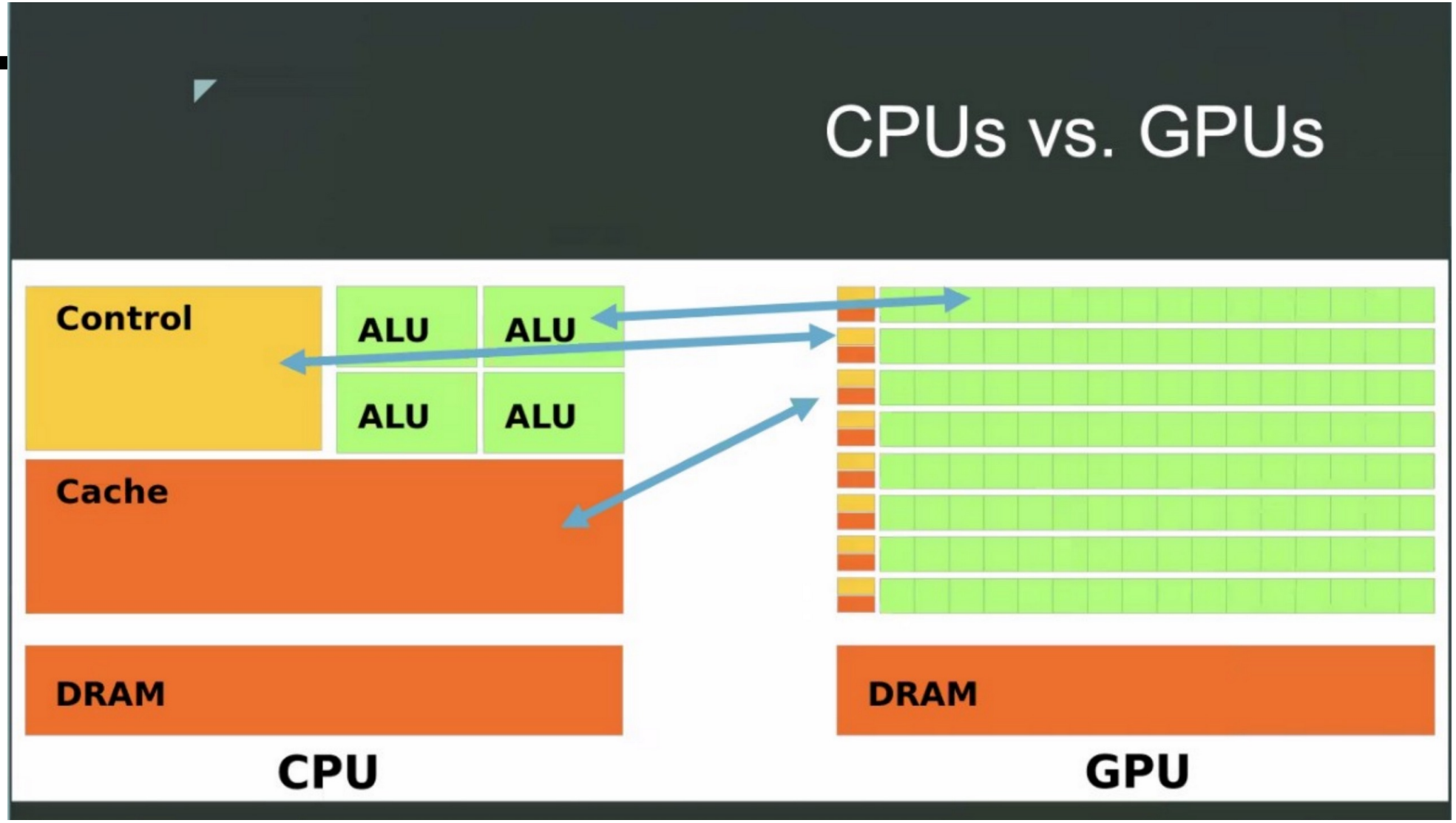
Source: Openhackaton @ Princeton 2023 "Introduction to GPU computing

CPUs vs GPUs comparison



Source: Openhackaton @ Princeton 2023 “Introduction to GPU computing

CPU vs GPU comparison



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

CPU vs GPU comparison

- DRAM is the global memory
- CPUs have few cores (ALU) when GPUs have a large number of cores.

Source: https://thedatafrog.com/en/articles/cuda-kernel-python/

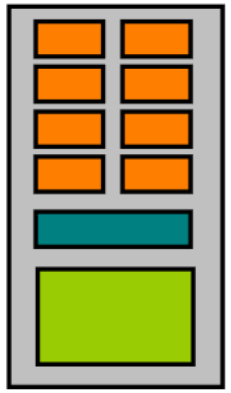
CPU vs GPU comparison

- DRAM is the global memory
- CPUs have few cores (ALU) when GPUs have a large number of cores.
- The CPU has lower latency and lower memory bandwidth compared to the GPU
- The GPU has high throughput and a higher memory bandwidth compared to the CPU

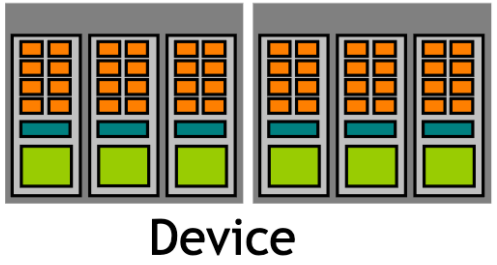
Source: https://thedatafrog.com/en/articles/cuda-kernel-python/

AMD hardware specs

Arithmetic logical unit (ALU)



Compute unit made of many ALUs



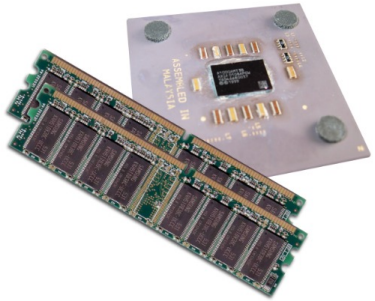
Many compute units make across the AMD GPU

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

CPU vs GPU programming model

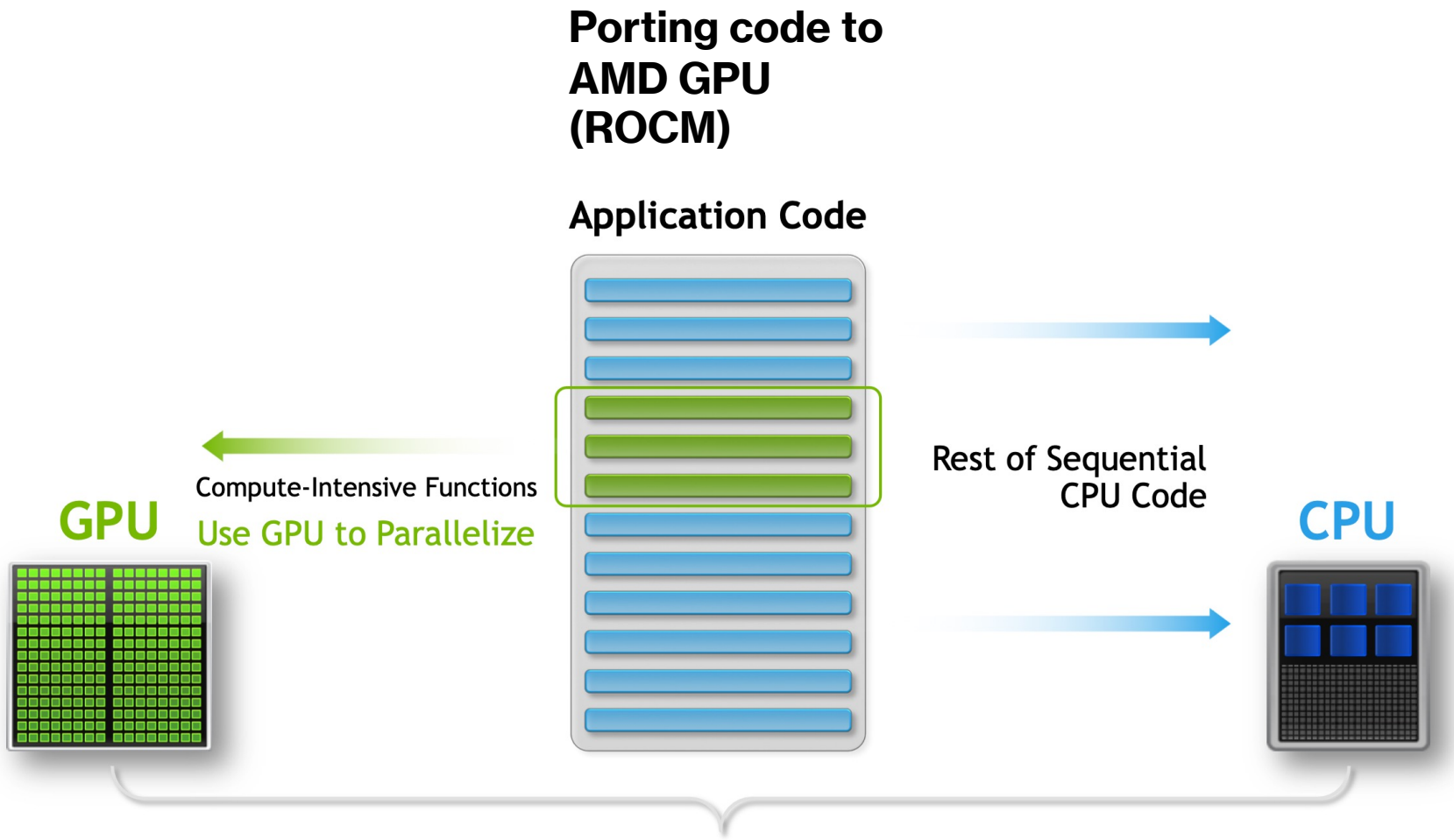
HETEROGENEOUS COMPUTING

- *Host* The CPU and its memory (host memory)
- *Device* The GPU and its memory (device memory)



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

CPU vs GPU programming model



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

CPU vs GPU programming model

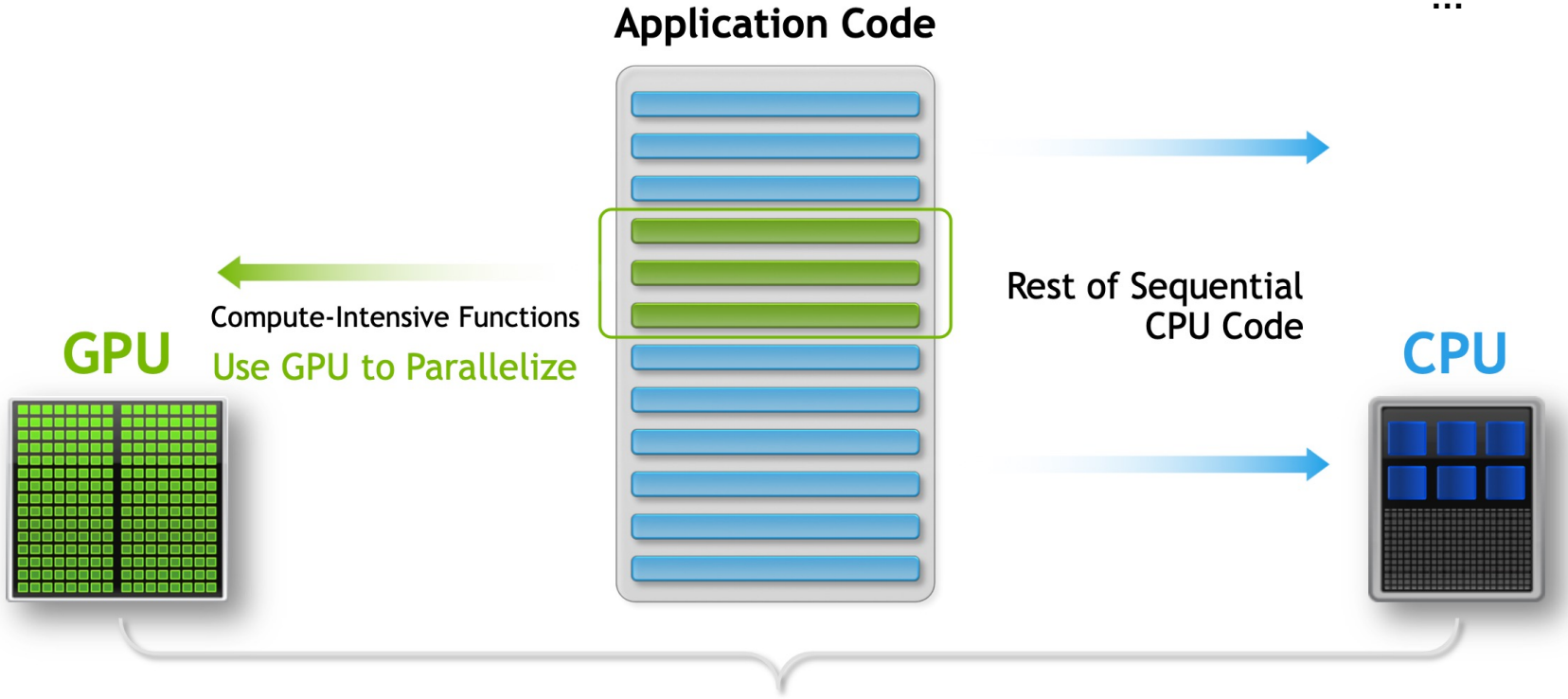
GPU: Matrix operations. (e.g. FFT)

Porting code to AMD GPU (ROCM)

CPU:

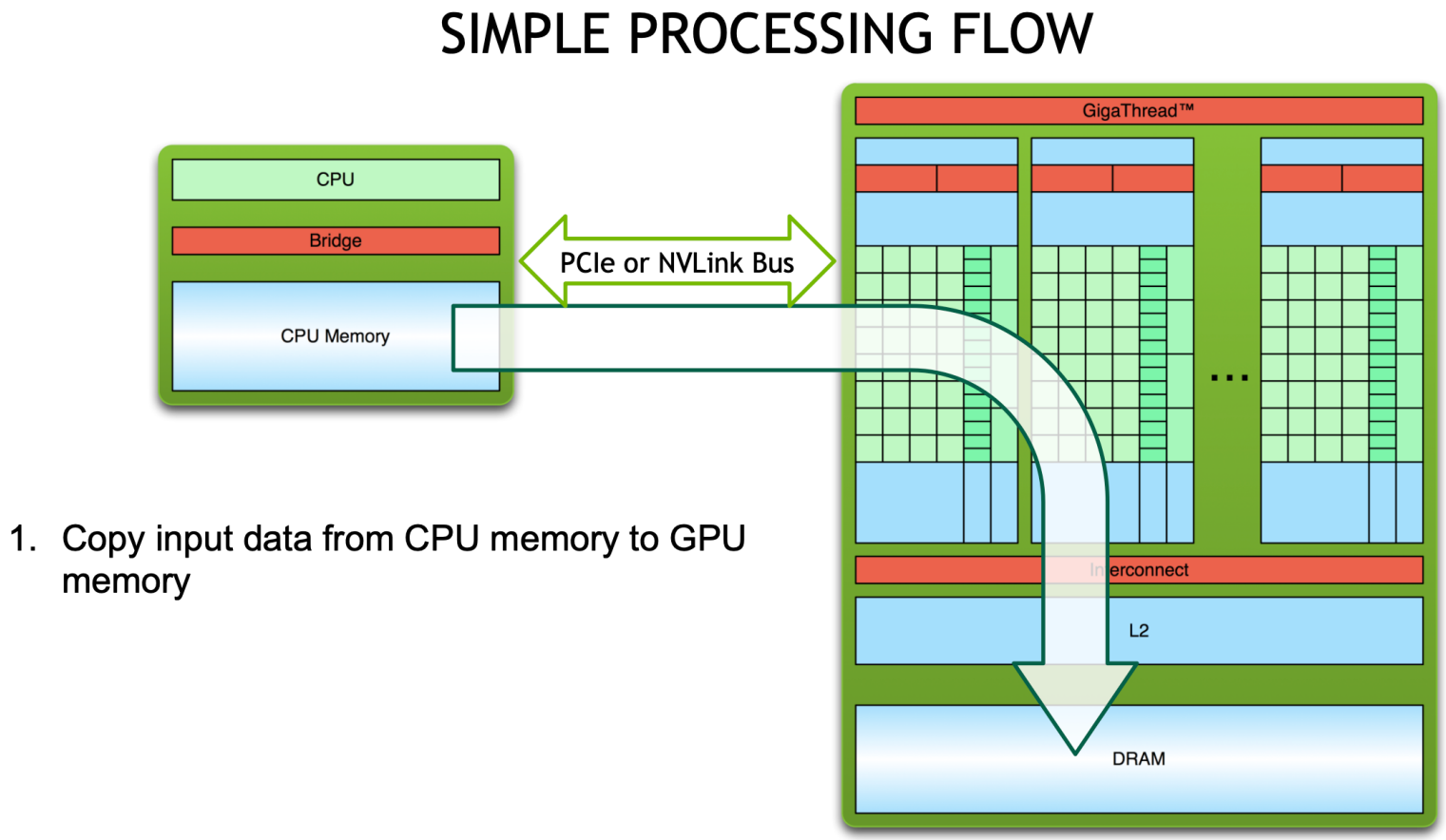
OS, security etc

...



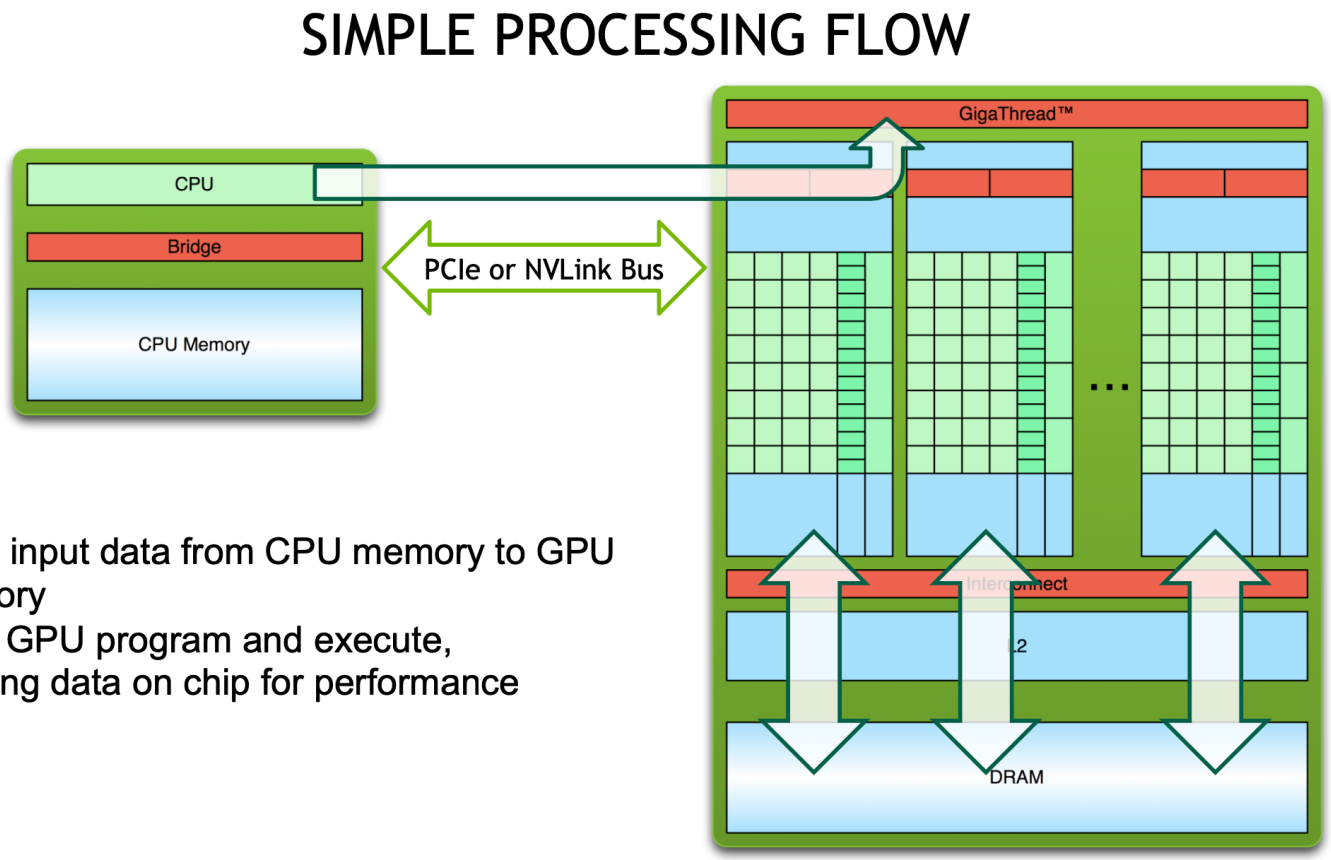
Source: https://www.olcf.ornl.gov/cuda-training-series/ ; CUDA C++

CPU vs GPU programming model



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

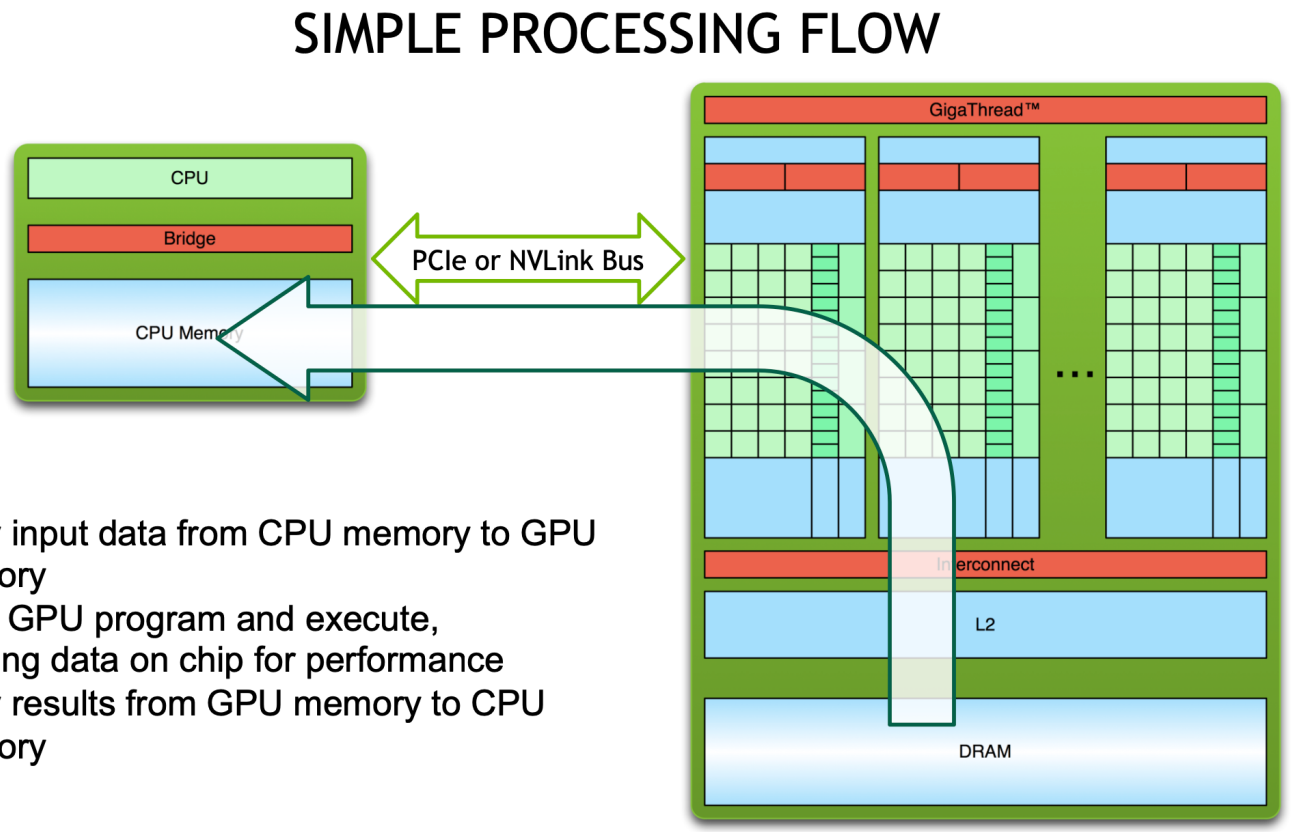
CPU vs GPU programming model



1. Copy input data from CPU memory to GPU memory
2. Load GPU program and execute, caching data on chip for performance

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

CPU vs GPU programming model



1. Copy input data from CPU memory to GPU memory
2. Load GPU program and execute, caching data on chip for performance
3. Copy results from GPU memory to CPU memory

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

AMD GPU programming model

- A kernel is a function launched to the GPU accelerated .

https://www.olcf.ornl.gov/wp-content/uploads/2019/10/ORNL_Application_Readiness_Workshop-AMD_GPU_Basics.pdf

AMD GPU programming model

- A kernel is a function launched to the GPU accelerated .
- It will execute multiple workers on the GPU

https://www.olcf.ornl.gov/wp-content/uploads/2019/10/ORNL_Application_Readiness_Workshop-AMD_GPU_Basics.pdf

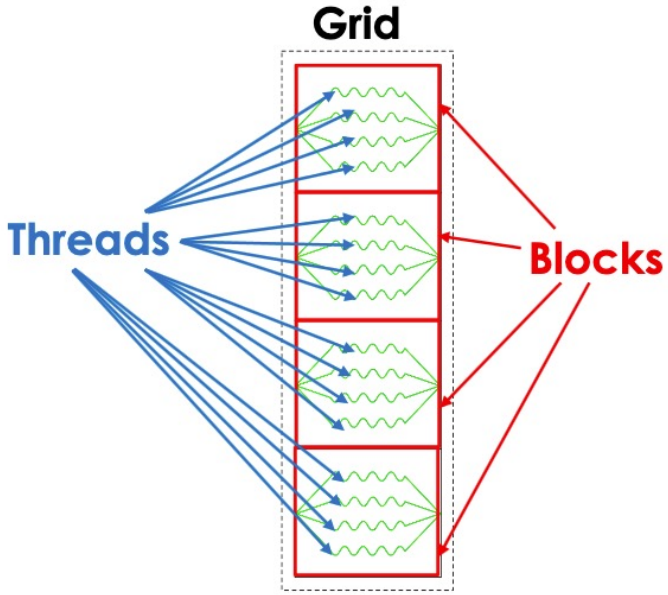
AMD GPU programming model

- A kernel is a function launched to the GPU accelerated .
- It will execute multiple workers on the GPU.
- A block of threads that will be executed by the compute unit is spawned

Source: https://www.olcf.ornl.gov/wp-content/uploads/2019/10/ORNL_Application_Readiness_Workshop-AMD_GPU_Basics.pdf; https://www.olcf.ornl.gov/wp-content/uploads/intro_to_hip.pdf

AMD GPU programming model

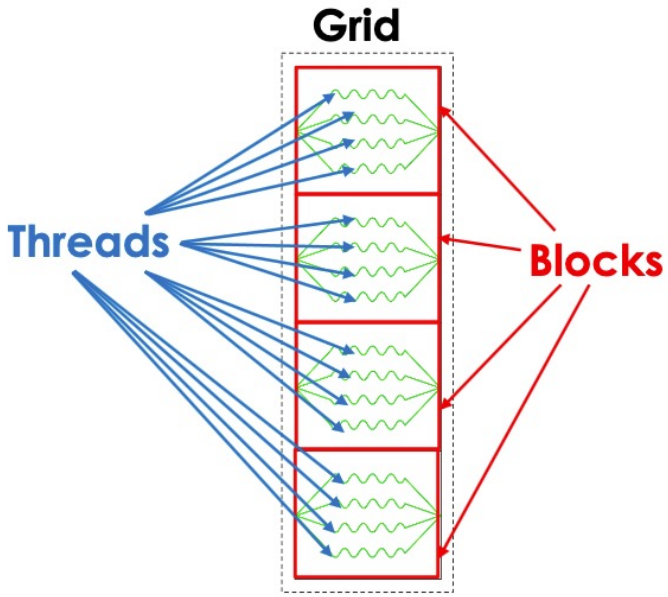
- A kernel is a function launched to the GPU accelerated .
- It will execute multiple workers on the GPU.
- A block of threads that will be executed by the compute unit is spawned
- Threads within a block can cooperate when performing computation



Source: https://www.olcf.ornl.gov/wp-content/uploads/2019/10/ORNL_Application_Readiness_Workshop-AMD_GPU_Basics.pdf; https://www.olcf.ornl.gov/wp-content/uploads/intro_to_hip.pdf

AMD GPU programming model

- A kernel is a function launched to the GPU accelerated .
- It will execute multiple workers on the GPU.
- A block of threads that will be executed by the compute unit is spawned
- Threads within a block can cooperate when performing computation

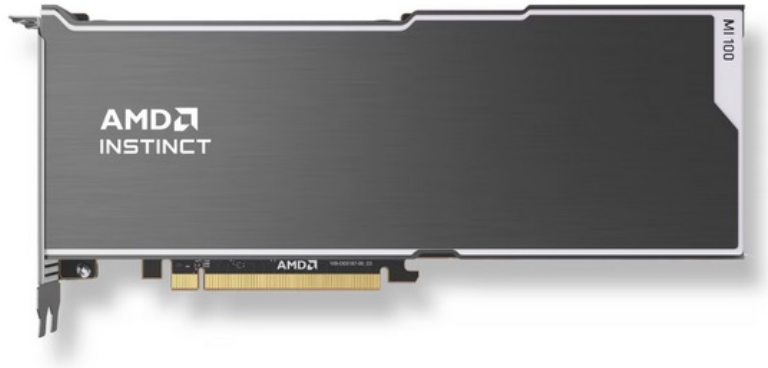


16 threads and 4 threads per block

Source: https://www.olcf.ornl.gov/wp-content/uploads/2019/10/ORNL_Application_Readiness_Workshop-AMD_GPU_Basics.pdf; https://www.olcf.ornl.gov/wp-content/uploads/intro_to_hip.pdf

Alpine AMD MI100 Specs

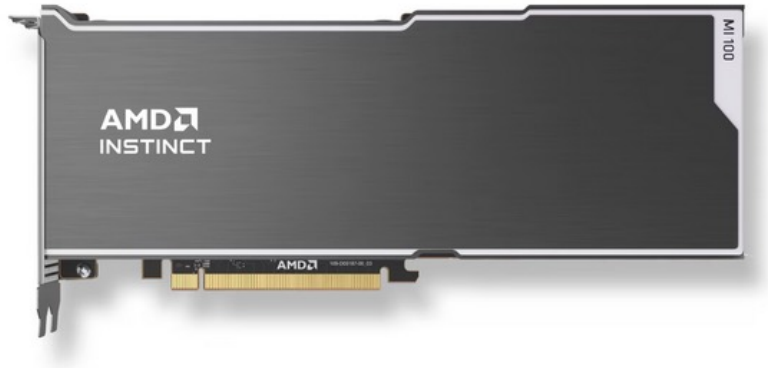
- 32 GB global memory for each gpu (80 GB memory for the NVIDIA A100 gpu)



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

Alpine AMD MI100 Specs

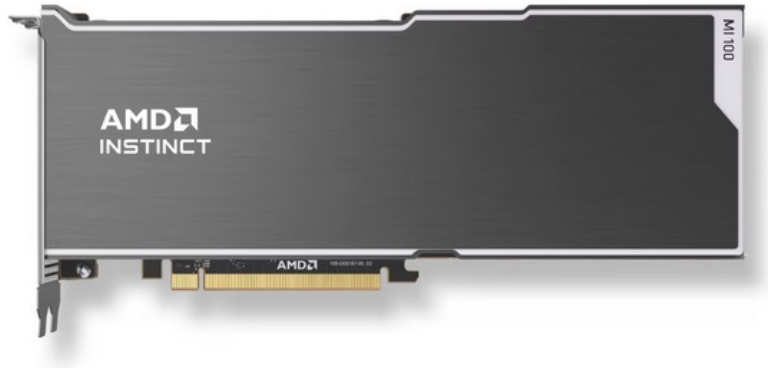
- 32 GB global memory for each gpu (80 GB memory for the NVIDIA A100 gpu)
- Peak memory bandwidth is 1.2 TB/s (>2TB/s for NVIDIA A100)



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

Alpine AMD MI100 Specs

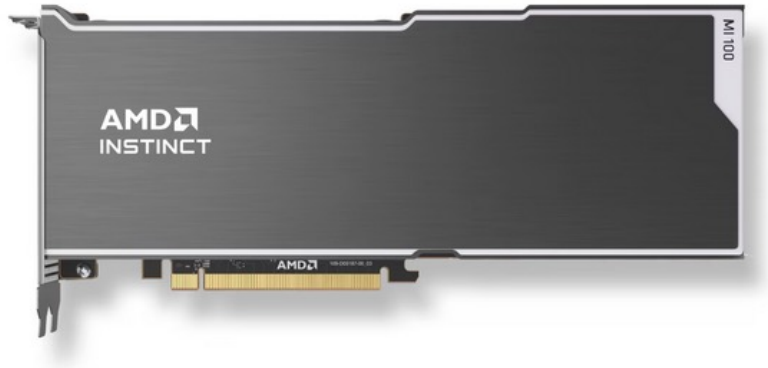
- 32 GB global memory for each gpu (80 GB memory for the NVIDIA A100 gpu)
- Peak memory bandwidth is 1.2 TB/s (>2TB/s for NVIDIA A100)
- 3 AMD gpus per node



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

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.

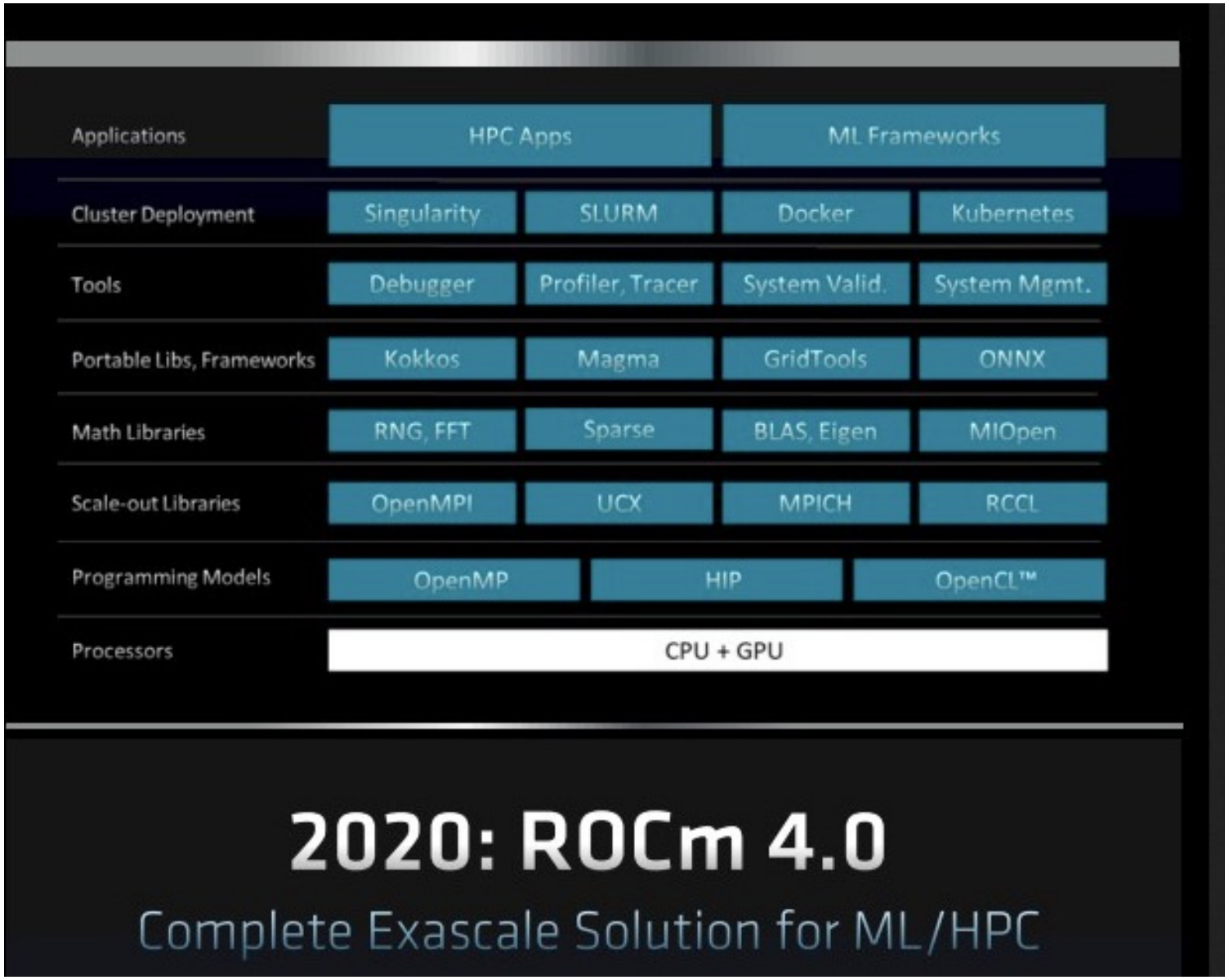
https://raw.githubusercontent.com/RadeonOpenCompute/ROCm/rocm-4.5.2/AMD_HIP_Programming_Guide.pdf

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 ROCM software stack



https://www.nextplatform.com/2022/03/08/with-rocm-software-and-instinct-mi200-gpus-amd-has-ecosystem-critical-mass/

AMD ROCM software stack

Current ROCm version on Alpine are 5.2.3, 5.3.0 and 5.5.0

https://www.nextplatform.com/2022/03/08/with-rocm-software-and-instinct-mi200-gpus-amd-has-ecosystem-critical-mass/

AMD ROCM software stack

CUDA Library	ROCm Library	Description
cuBLAS	rocBLAS	Basic Linear Algebra Subroutines
cuFFT	rocFFT	Fast Fourier Transfer Library
cuSPARSE	rocSPARSE	Sparse BLAS + SPMV
cuSolver	rocSolver	Lapack Library
AMG-X	rocALUTION	Sparse iterative solvers & preconditioners with Geometric & Algebraic MultiGrid
Thrust	rocThrust	C++ parallel algorithms library
CUB	rocPRIM	Low Level Optimized Parallel Primitives
cuDNN	MIOpen	Deep learning Solver Library
cuRAND	rocRAND	Random Number Generator Library
EIGEN	EIGEN	C++ template library for linear algebra: matrices, vectors, numerical solvers
NCCL	RCCL	Communications Primitives Library based on the MPI equivalents

https://www.nextplatform.com/2022/03/08/with-rocm-software-and-instinct-mi200-gpus-amd-has-ecosystem-critical-mass/

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

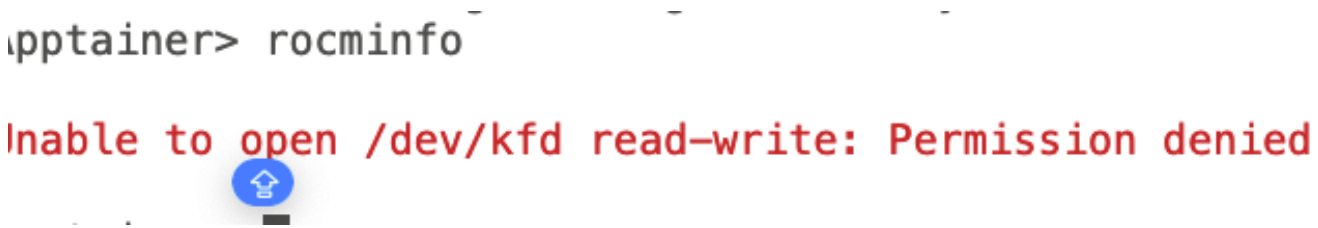
- groups $USER

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
```

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
```

ROCM standard to install packages

- git clone git@github.com:kf-cuanschutz/amd_gpu_workshop_022924.git
- chmod +x part1-install_amd_gpu_standard.sh
- bash part1-install_amd_gpu_standard.sh

ROCM standard to install packages

```
#!/bin/bash

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


# 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

echo "********************************** Loading ROCM and pytorch-rocm **********************************"
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 export the TMP_DIR so that we do not fill /tmp space
echo "********************************** Exporting TMP related directories  **********************************"
export TMP=/gpfs/alpine1/scratch/$USER/cache_dir
mkdir -pv $TMP
export TEMP=$TMP
export TMPDIR=$TMP
export TEMPDIR=$TMP
export PIP_CACHE_DIR=$TMP
```

In case we use mambaforge

ROCM standard to install packages

```
#!/bin/bash

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


# 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

echo "**********  Loading ROCM and pytorch-rocm  **********"
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 export the TMP_DIR so that we do not fill /tmp space
echo "**********  Exporting TMP related directories  **********"
export TMP=/gpfs/alpine1/scratch/$USER/cache_dir
mkdir -pv $TMP
export TEMP=$TMP
export TMPDIR=$TMP
export TEMPDIR=$TMP
export PIP_CACHE_DIR=$TMP
```

We load ROCM that comes bundled with pytorch

ROCM standard to install packages

```
#!/bin/bash

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


# 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

echo "********************************** Loading ROCM and pytorch-rocm **********************************"
module load rocm/5.2.3

# We load pytorch 1.13.0 that has been built for the ROCM,
# which is compatible with ROCM 5.2.X : https://pytorch.org/get-started/previous-versions/
module load pytorch/1.13.0

# We export the TMP_DIR so that we do not fill /tmp space
echo "********************************** Exporting TMP related directories  **********************************"
export TMP=/gpfs/alpine1/scratch/$USER/cache_dir
mkdir -pv $TMP
export TEMP=$TMP
export TMPDIR=$TMP
export TEMPDIR=$TMP
export PIP_CACHE_DIR=$TMP
```

We load pytorch

ROCM standard to install packages

```
#!/bin/bash

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


# 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

echo "************************************ Loading ROCM and pytorch-rocm ********************************"
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 export the TMP_DIR so that we do not fill /tmp space
echo "************************************ Exporting TMP related directories  ********************************"
export TMP=/gpfs/alpine1/scratch/$USER/cache_dir
mkdir -pv $TMP
export TEMP=$TMP
export TMPDIR=$TMP
export TEMPDIR=$TMP
export PIP_CACHE_DIR=$TMP
```

Exporting TMP_DIR

ROCM standard to install packages

```
#!/bin/bash

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


# 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

echo "************************************ Loading ROCM and pytorch-rocm ********************************"
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 export the TMP_DIR so that we do not fill /tmp space
echo "************************************ Exporting TMP related directories  ********************************"
export TMP=/gpfs/alpine1/scratch/$USER/cache_dir
mkdir -pv $TMP
export TEMP=$TMP
export TMPDIR=$TMP
export TEMPDIR=$TMP
export PIP_CACHE_DIR=$TMP
```

Exporting TMP_DIR

ROCM standard to install packages

```
# We point the pip cache dir to /scratch as well
echo "********************************** Exporting PIP related variables  **********************************"
export PIP_CACHE_DIR=/gpfs/alpine1/scratch/$USER/cache_dir
mkdir $PIP_CACHE_DIR

# We determine an install directory, in scratch for this
export PIP_INSTALL_DIR=/gpfs/alpine1/scratch/$USER/software/amd_gpu_standard_install
mkdir -pv $PIP_INSTALL_DIR

# We determine an install directory, in scratch for this
# We export all the ROCM related path and libraries
#export ROCM_PATH=/curc/sw/install/rocm/5.2.3
#export HIP_PATH=$ROCM_PATH
#export HIP_ROOT_DIR=$ROCM_PATH
#export ROCM_LIBRARIES=$ROCM_PATH/lib
export PYTHONPATH=$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 change the pip cache dir so that it does not fill the ~

ROCM standard to install packages

```
# We point the pip cache dir to /scratch as well
echo "********************************** Exporting PIP related variables  ********************************"
export PIP_CACHE_DIR=/gpfs/alpine1/scratch/$USER/cache_dir
mkdir $PIP_CACHE_DIR

# We determine an install directory, in scratch for this
export PIP_INSTALL_DIR=/gpfs/alpine1/scratch/$USER/software/amd_gpu_standard_install
mkdir -pv $PIP_INSTALL_DIR

# We determine an install directory, in scratch for this
# We export all the ROCM related path and libraries
#export ROCM_PATH=/curc/sw/install/rocm/5.2.3
#export HIP_PATH=$ROCM_PATH
#export HIP_ROOT_DIR=$ROCM_PATH
#export ROCM_LIBRARIES=$ROCM_PATH/lib
export PYTHONPATH=$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 need to specify a different install directory otherwise it will try to install into /curc

ROCM standard to install packages

```
# We point the pip cache dir to /scratch as well
echo "********************************** Exporting PIP related variables  ********************************"
export PIP_CACHE_DIR=/gpfs/alpine1/scratch/$USER/cache_dir
mkdir $PIP_CACHE_DIR

# We determine an install directory, in scratch for this
export PIP_INSTALL_DIR=/gpfs/alpine1/scratch/$USER/software/amd_gpu_standard_install
mkdir -pv $PIP_INSTALL_DIR

# We determine an install directory, in scratch for this
# We export all the ROCM related path and libraries
#export ROCM_PATH=/curc/sw/install/rocm/5.2.3
#export HIP_PATH=$ROCM_PATH
#export HIP_ROOT_DIR=$ROCM_PATH
#export ROCM_LIBRARIES=$ROCM_PATH/
export PYTHONPATH=$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
```

PYTHON PATH so that we add the new install path

ROCM standard to install packages

```
# We point the pip cache dir to /scratch as well
echo "********************************** Exporting PIP related variables  **********************************"
export PIP_CACHE_DIR=/gpfs/alpine1/scratch/$USER/cache_dir
mkdir $PIP_CACHE_DIR

# We determine an install directory, in scratch for this
export PIP_INSTALL_DIR=/gpfs/alpine1/scratch/$USER/software/amd_gpu_standard_install
mkdir -pv $PIP_INSTALL_DIR

# We determine an install directory, in scratch for this
# We export all the ROCM related path and libraries
#export ROCM_PATH=/curc/sw/install/rocm/5.2.3
#export HIP_PATH=$ROCM_PATH
#export HIP_ROOT_DIR=$ROCM_PATH
#export ROCM_LIBRARIES=$ROCM_PATH/lib
export PYTHONPATH=$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
```

Extra variables that can be useful at time

ROCM standard to install packages

```
echo "********************************** Installing complementary packages  ********************************"
# We may now install the additional pytorch AMD compatible packages:
pip install --target=$PIP_INSTALL_DIR pandas
pip install --target=$PIP_INSTALL_DIR transformers
pip install --target=$PIP_INSTALL_DIR scikit-learn
pip install --target=$PIP_INSTALL_DIR tqdm
#pip install --target=$PIP_INSTALL_DIR cmake
pip install --target=$PIP_INSTALL_DIR  mxnet #mxnet is a deeplearning framework for efficiency and flexibility : https://pypi.org/project
export PATH=$PATH:$PIP_INSTALL_DIR/bin
```

Now we can install the associated packages in the install directory

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

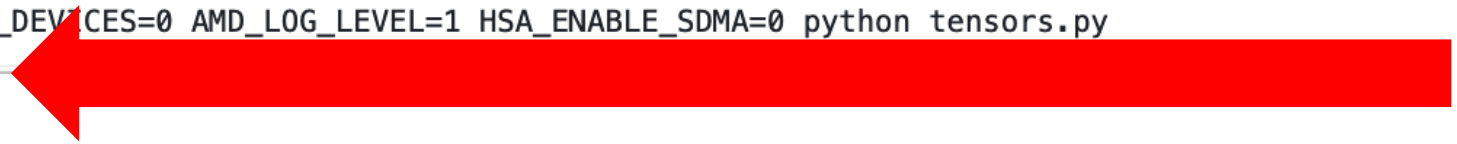
Running our first model

```
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

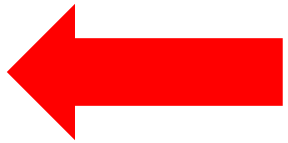
Now we run the job with LOG_LEVEL 4 or 5

- 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

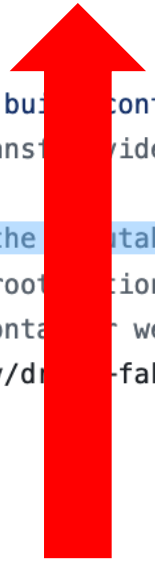
- |

```
#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 trans  ideo 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/d  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/
```

Note:

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

Note:

- 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.

Note:

- 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.

- We will work on it so that it can be supported as well soon.

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.

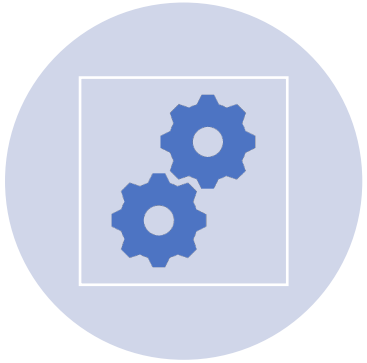
What is Horovod? (2)

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

Context (1) – Tensorflow distributed



There are other tools to achieve distributed ML parallelism.

Source: https://www.tensorflow.org/guide/distributed_training

Context (1) – Tensorflow distributed

There are other tools to achieve distributed ML parallelism.

Tensorflow distributed (tf.distribute.Strategy).

Specific to Tensorflow only.

Supports parallelization on GPUs and TPUs.

More used on the cloud and less on HPC.

Source: https://www.tensorflow.org/guide/distributed_training

Context (2) – Pytorch distributed

- **Pytorch Distributed framework**
- Developed by the Facebook AI team
- Lots of tutorial on how to use it on HPC
- Creates replicas of the model with each working on a batch

Pytorch distributed parallel example

```
import torch
import torch.nn as nn
import torch.nn.parallel as par
import torch.optim as optim

# initialize torch.distributed properly
# with init_process_group

# setup model and optimizer
net = nn.Linear(10, 10)
net = par.DistributedDataParallel(net)
opt = optim.SGD(net.parameters(), lr=0.01)

# run forward pass
inp = torch.randn(20, 10)
exp = torch.randn(20, 10)
out = net(inp)

# run backward pass
nn.MSELoss()(out, exp).backward()

# update parameters
opt.step()
```

Import torch and torch DDP modules

Source: "Pytorch Distrited: Experiences on Accelerating Data Parallel Training"

Pytorch distributed parallel example

```
import torch
import torch.nn as nn
import torch.nn.parallel as par
import torch.optim as optim

# initialize torch.distributed properly
# with init_process_group

# setup model and optimizer
net = nn.Linear(10, 10)
net = par.DistributedDataParallel(net)
opt = optim.SGD(net.parameters(), lr=0.01)

# run forward pass
inp = torch.randn(20, 10)
exp = torch.randn(20, 10)
out = net(inp)

# run backward pass
nn.MSELoss()(out, exp).backward()

# update parameters
opt.step()
```

Local model is created

Source: "Pytorch Distrited: Experiences on Accelerating Data Parallel Training"

Pytorch distributed parallel example

```
import torch
import torch.nn as nn
import torch.nn.parallel as par
import torch.optim as optim

# initialize torch.distributed properly
# with init_process_group

# setup model and optimizer
net = nn.Linear(10, 10)
net = par.DistributedDataParallel(net)
opt = optim.SGD(net.parameters(), lr=0.01)

# run forward pass
inp = torch.randn(20, 10)
exp = torch.randn(20, 10)
out = net(inp)

# run backward pass
nn.MSELoss()(out, exp).backward()

# update parameters
opt.step()
```

Converted into distributed training model

Source: "Pytorch Distrited: Experiences on Accelerating Data Parallel Training"

Pytorch distributed parallel example

```
import torch
import torch.nn as nn
import torch.nn.parallel as par
import torch.optim as optim

# initialize torch.distributed properly
# with init_process_group

# setup model and optimizer
net = nn.Linear(10, 10)
net = par.DistributedDataParallel(net)
opt = optim.SGD(net.parameters(), lr=0.01)

# run forward pass
inp = torch.randn(20, 10)
exp = torch.randn(20, 10)
out = net(inp)

# run backward pass
nn.MSELoss()(out, exp).backward()

# update parameters
opt.step()
```

Optimizer

Source: "Pytorch Distrited: Experiences on Accelerating Data Parallel Training"

Pytorch distributed parallel example

```
import torch
import torch.nn as nn
import torch.nn.parallel as par
import torch.optim as optim

# initialize torch.distributed properly
# with init_process_group

# setup model and optimizer
net = nn.Linear(10, 10)
net = par.DistributedDataParallel(net)
opt = optim.SGD(net.parameters(), lr=0.01)

# run forward pass
inp = torch.randn(20, 10)
exp = torch.randn(20, 10)
out = net(inp)

# run backward pass
nn.MSELoss()(out, exp).backward()

# update parameters
opt.step()
```

Backward pass of the model

Source: "Pytorch Distrited: Experiences on Accelerating Data Parallel Training"

Pytorch distributed parallel example

```
import torch
import torch.nn as nn
import torch.nn.parallel as par
import torch.optim as optim

# initialize torch.distributed properly
# with init_process_group

# setup model and optimizer
net = nn.Linear(10, 10)
net = par.DistributedDataParallel(net)
opt = optim.SGD(net.parameters(), lr=0.01)

# run forward pass
inp = torch.randn(20, 10)
exp = torch.randn(20, 10)
out = net(inp)

# run backward pass
nn.MSELoss()(out, exp).backward()

# update parameters
opt.step()
```

Relatively non intrusive overall!!!

Source: "Pytorch Distrited: Experiences on Accelerating Data Parallel Training"

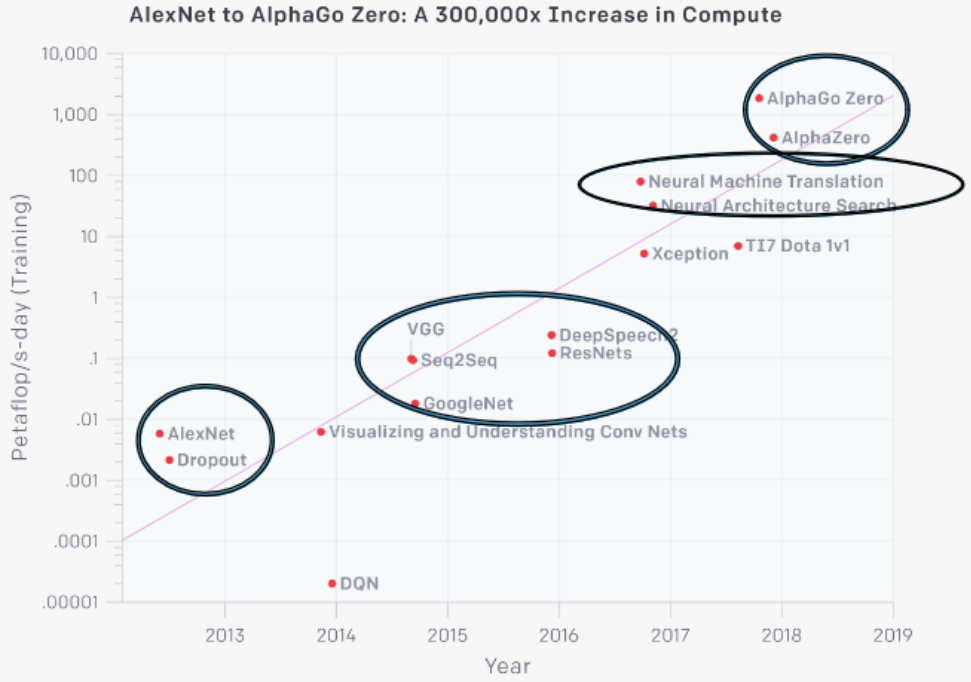
Context (3) - LBANN

- Livermore Big Artificial Neural Network toolkit (LBANN)
- Can be installed with spack
- Provides a Pytorch frontend
- Targets HPC systems with homogeneous compute nodes
- GPU accelerators.
- Relies only on MPI for communication.

Need for ML distributed training(1)

Need for distributed (parallel) training on HPC

"Since 2012, the amount of compute used in the largest AI training runs has been increasing exponentially with a 3.5 month doubling time (by comparison, Moore's Law had an 18 month doubling period)." https://openai.com/blog/ai-and-compute/



Eras:

- Before 2012 …
- 2012 – 2014: single to couple GPUs
- 2014 – 2016: 10 – 100 GPUs
- 2016 – 2017: large batch size training, architecture search, special hardware (etc, TPU)

Source: ATPESC 2022, https://extremecomputingtraining.anl.gov/wp-content/uploads/sites/96/2022/11/ATPESC-2022-Track-8-Talk-4-Velusamy-DataParallel-Updated.pdf

Need for ML distributed training(2)

Distributed deep learning for ResNet-50

TRAINING TIME AND TOP-1 VALIDATION ACCURACY WITH RESNET-50 ON IMAGENET

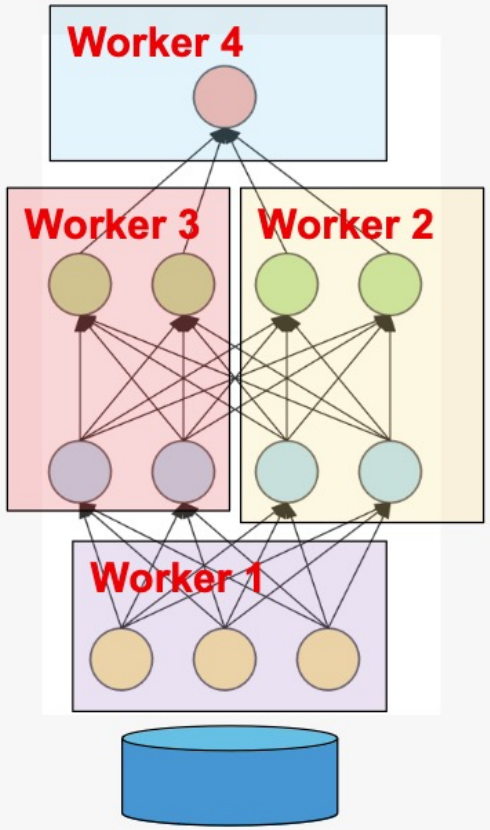
		Batch Size	Processor	DL Library	Time	Accuracy
He et al. [1]	2016	256	Tesla P100 × 8	Caffe	29 hours	75.3 %
Goyal et al. [2]		8,192	Tesla P100 × 256	Caffe2	1 hour	76.3 %
Smith et al. [3]		8,192 → 16,384	full TPU Pod	TensorFlow	30 mins	76.1 %
Akiba et al. [4]		32,768	Tesla P100 × 1,024	Chainer	15 mins	74.9 %
Jia et al. [5]		65,536	Tesla P40 × 2,048	TensorFlow	6.6 mins	75.8 %
Ying et al. [6]		65,536	TPU v3 × 1,024	TensorFlow	1.8 mins	75.2 %
Mikami et al. [7]		55,296	Tesla V100 × 3,456	NNL	2.0 mins	75.29 %
Yamazaki et al	2019	**81,920**	**Tesla V100 × 2,048**	**MXNet**	**1.2 mins**	**75.08%**

Quoted from Masafumi Yamazaki, arXiv:1903.12650

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 (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'))
```

```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()
```

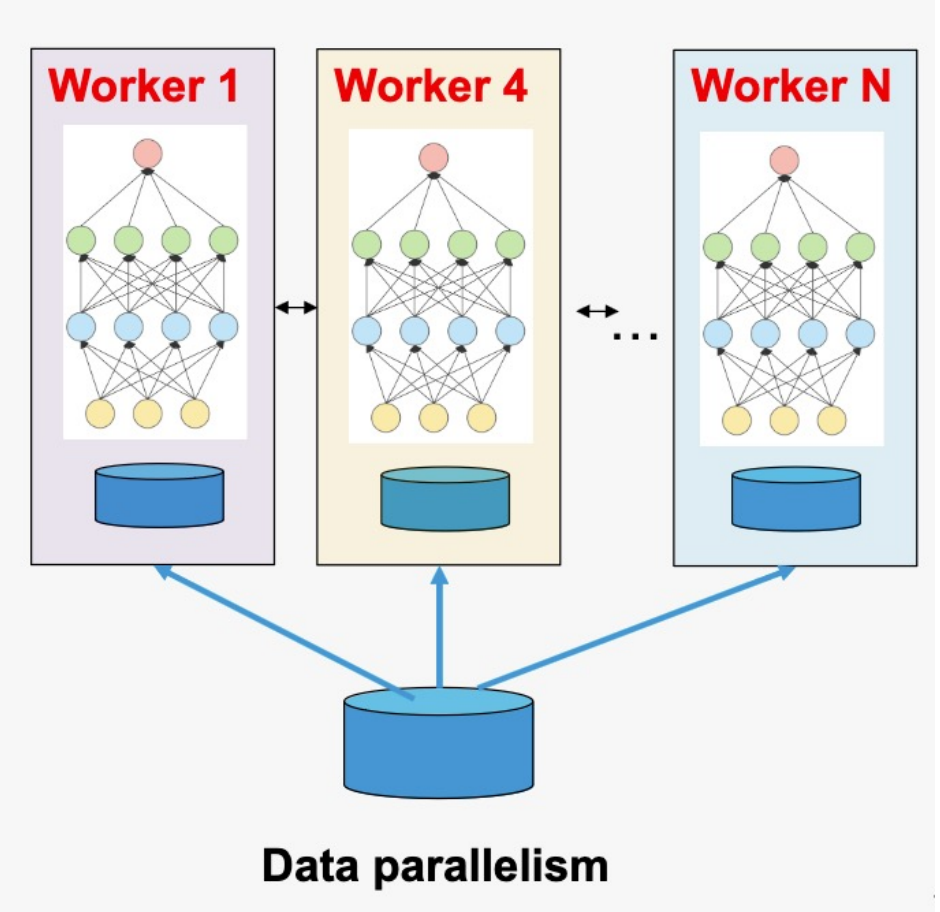
PyTorch multiple GPU model parallelism within a node

Worker 1

Worker 2

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

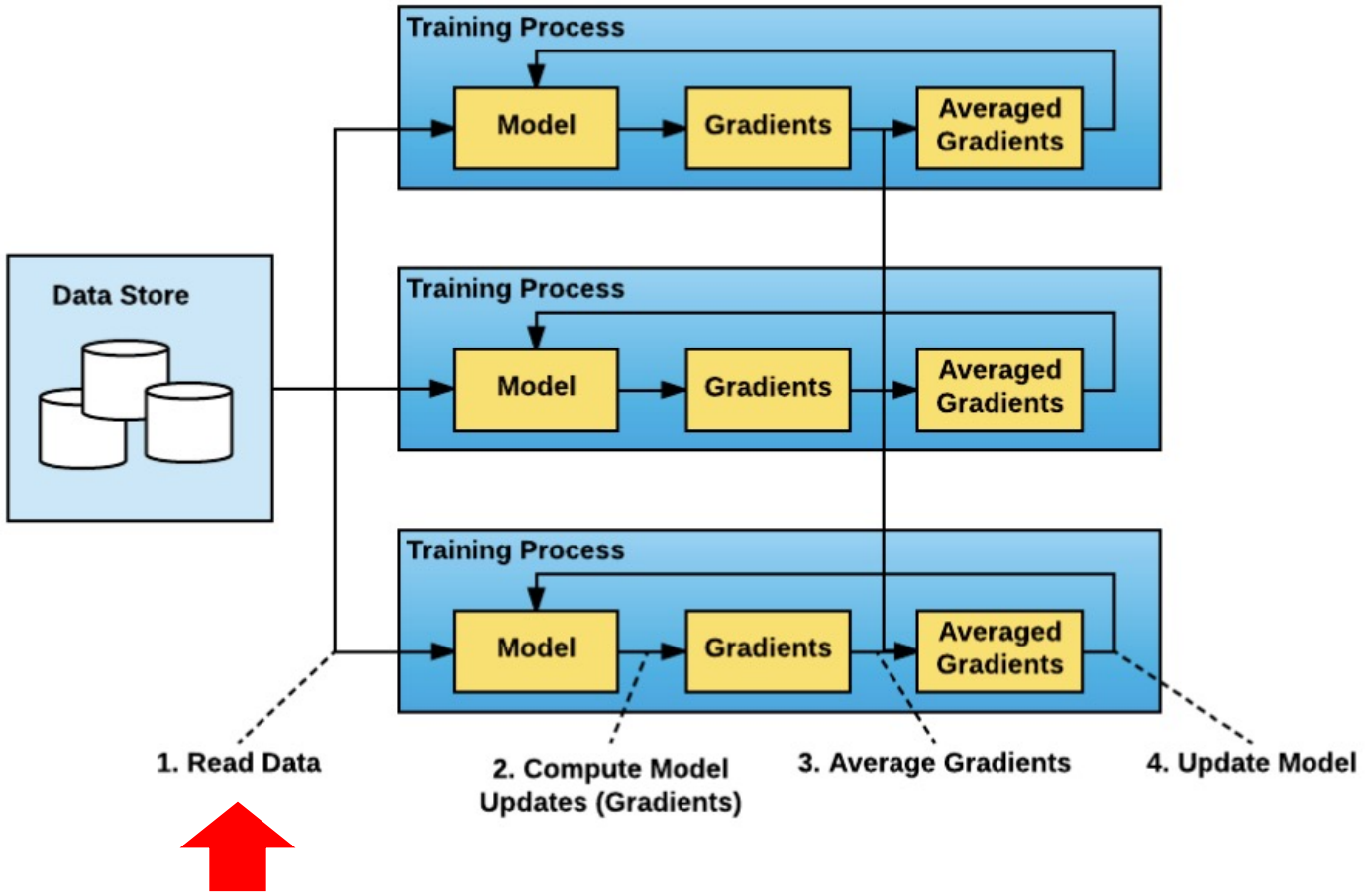
Parallelization schemes (2)



- 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

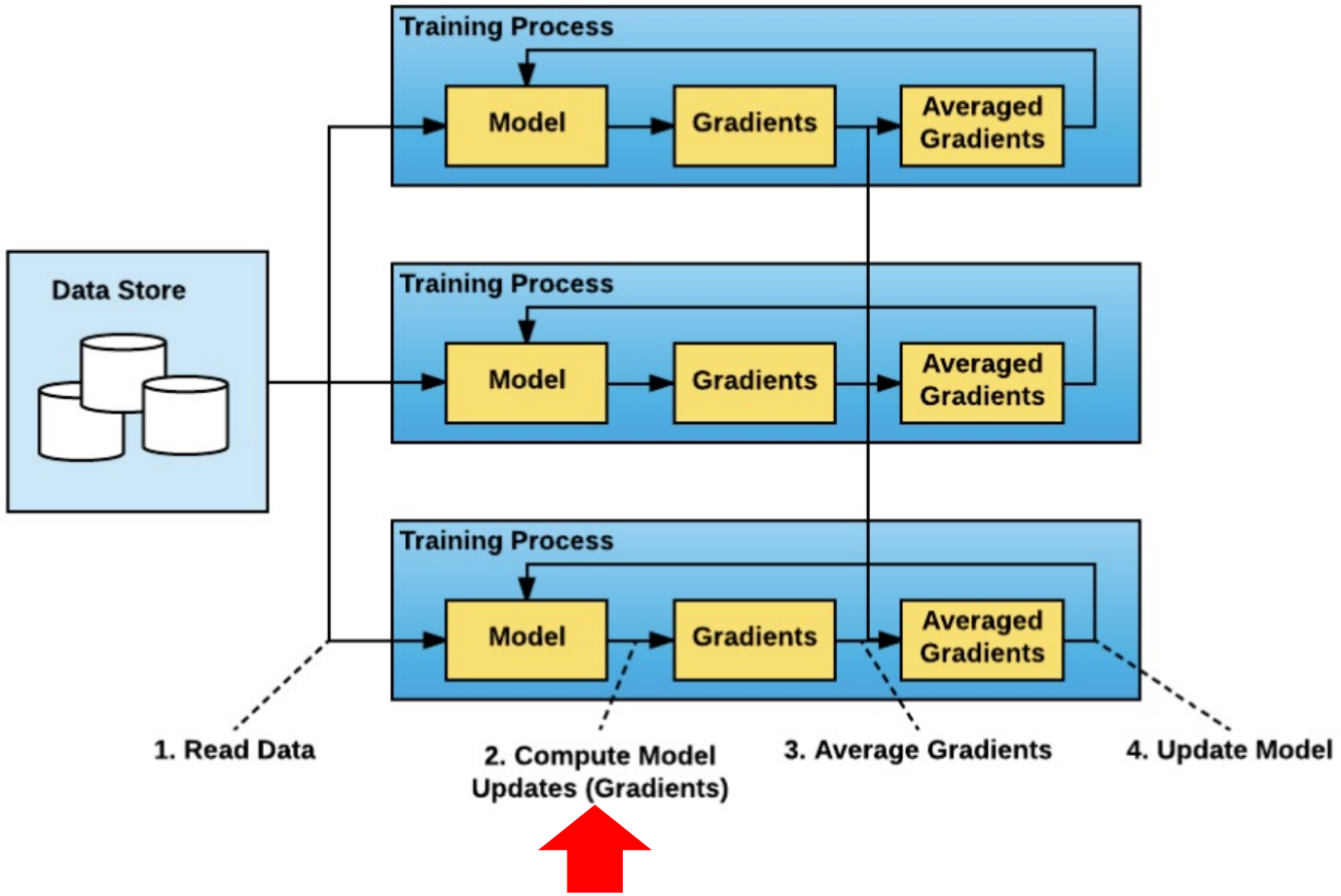
Overview of the Horovod training model



Reads chunk of data (batch)

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

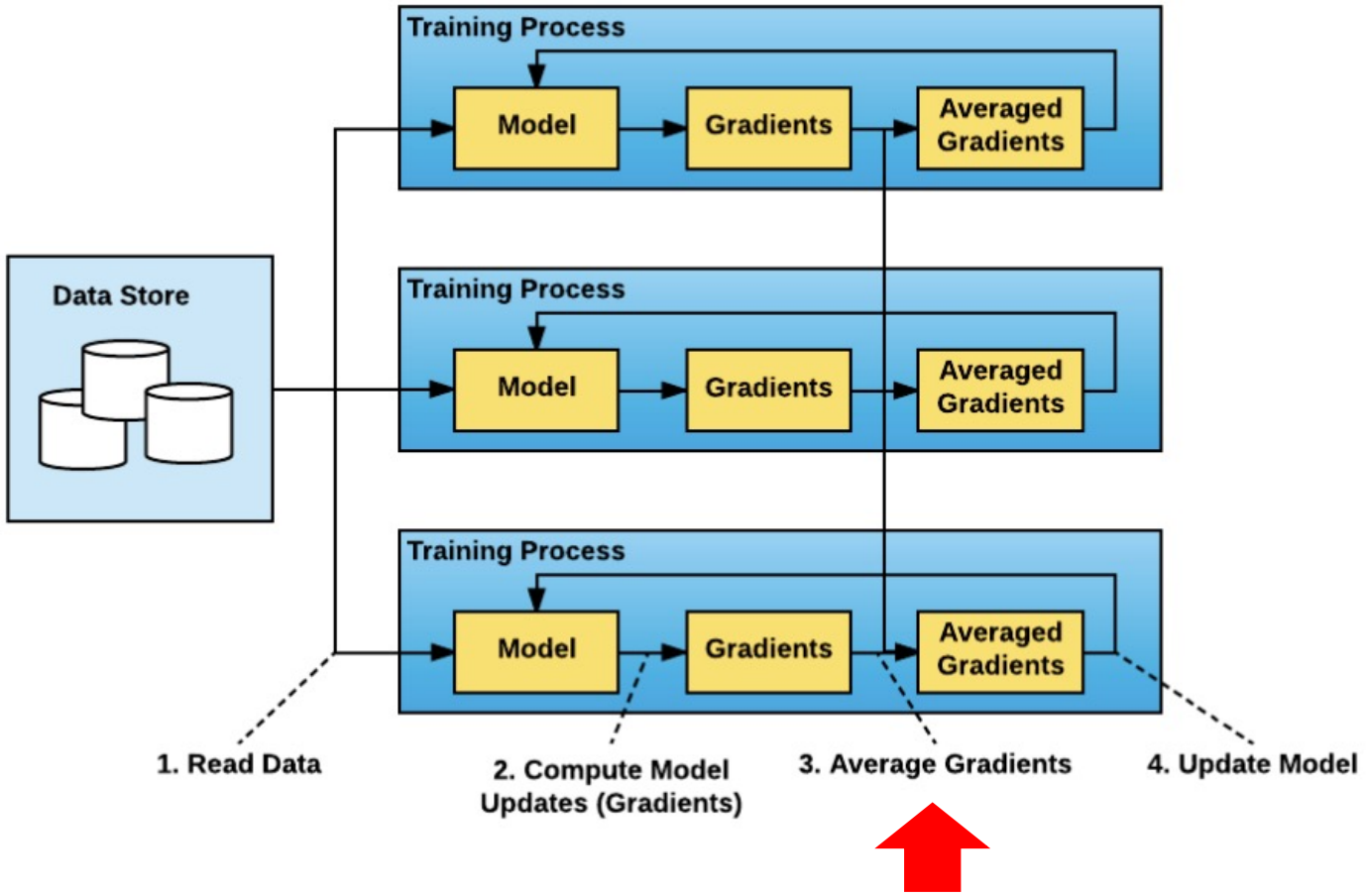
Overview of the Horovod training model (1)



Gradients of different batches calculated separately

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

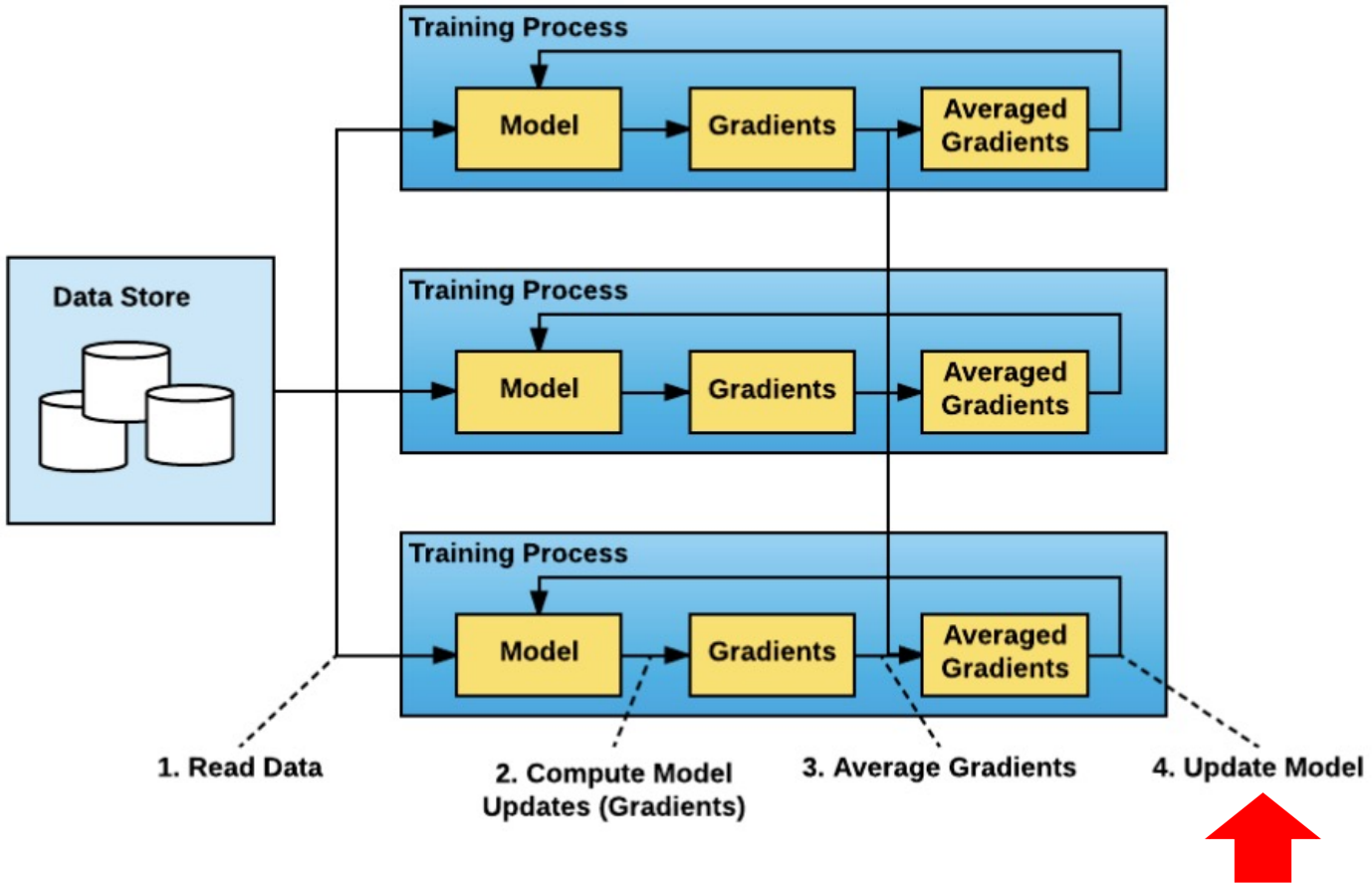
Overview of the Horovod training model(1)



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)



Update the model and repeat

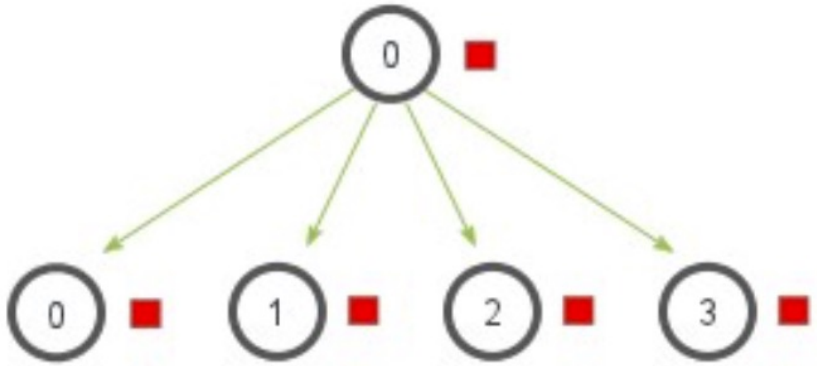
Source: "Horovod: fast and easy distributed deep learning in Tensorflow" by Sergeev and Balso

Distributed computing basics (1)

- A rank is an integer that serves as an identifier of a node from 0 to N-1

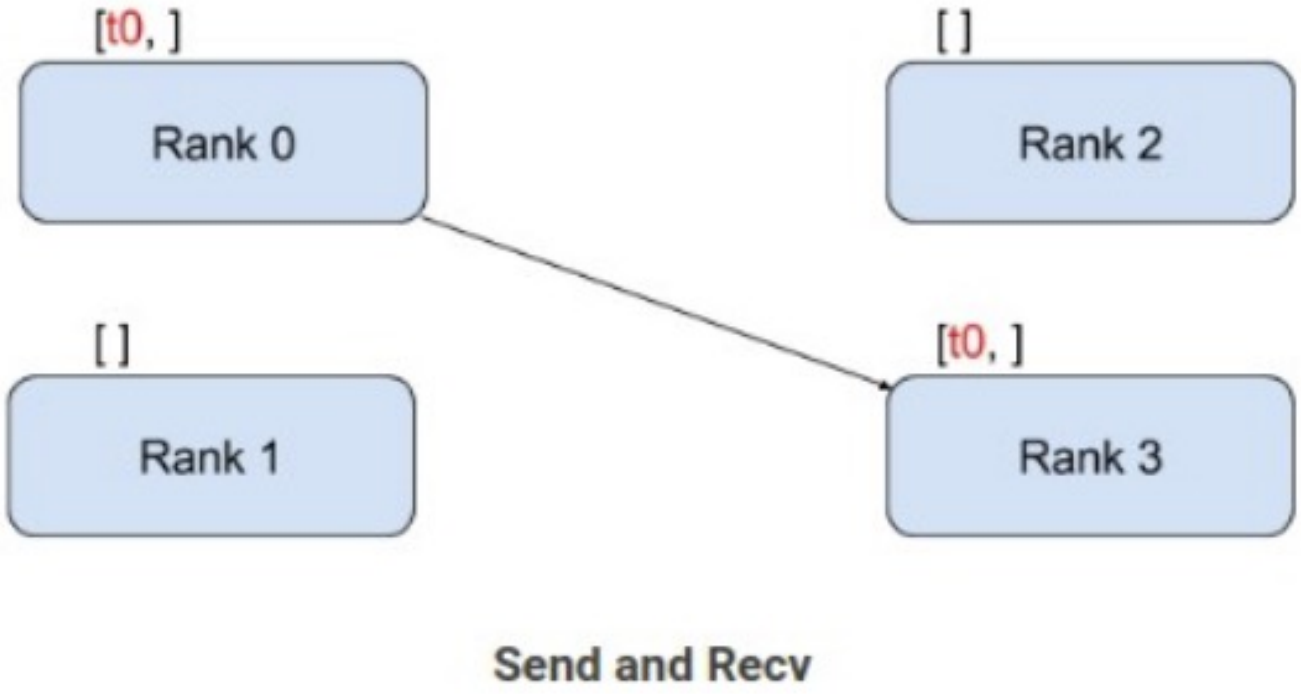
- A rank of 0 identifies the “Master Node” and any other ranks are the workers.

- Basic operation of distributed computing.



Point-to-point communication

- Transfer data from one rank to another



Source: Pytorch tutorial WRITING DISTRIBUTED APPLICATIONS WITH PYTORCH

Point-to-point communication

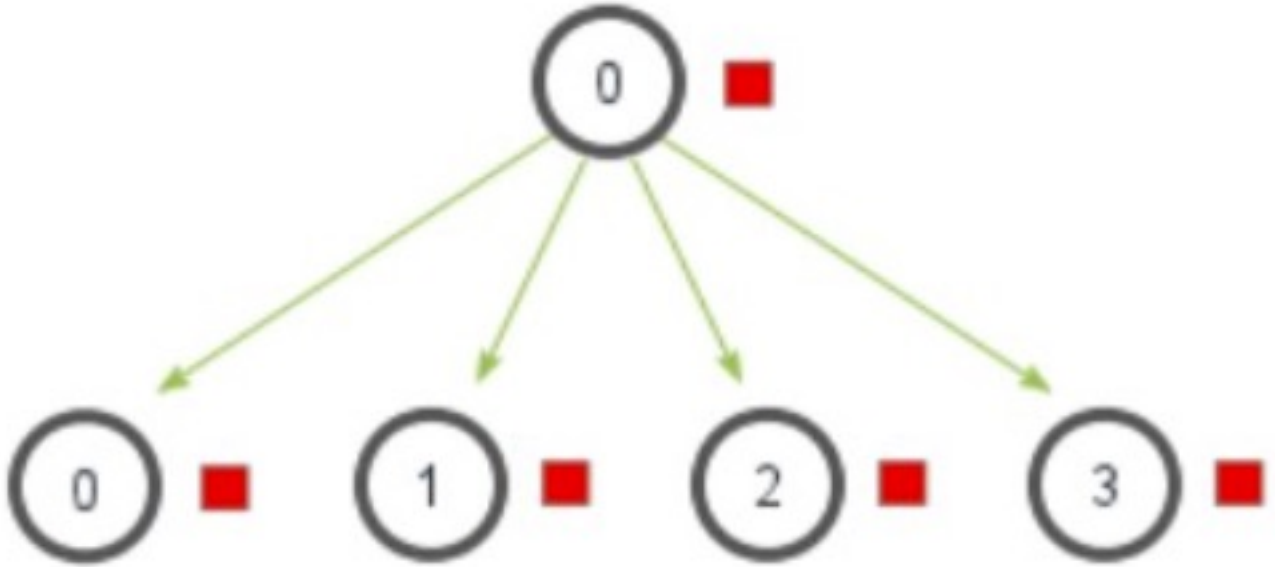
```
from mpi4py import MPI
import numpy

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

if rank == 0:
    data = {'a': 7, 'b': 3.14}
    comm.send(data, dest=1)
elif rank == 1:
    data = comm.recv(source=0)
    print('On process 1, data is ',data)
```

Rank 0 is sending data to rank 1

Source: https://mpi4py.readthedocs.io/en/stable/tutorial.html

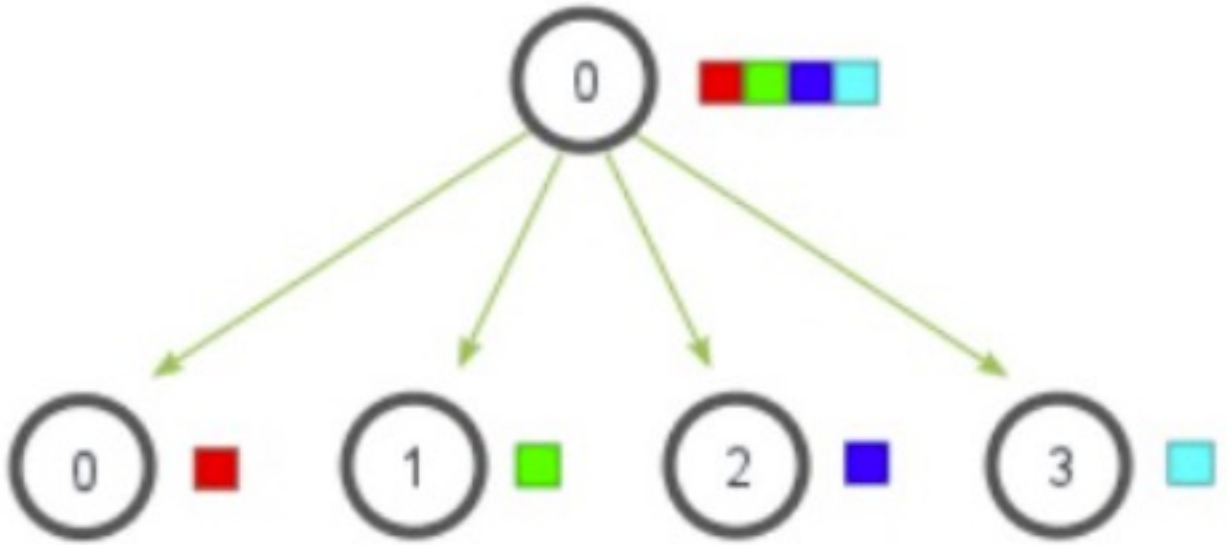
Recap- Collective communication (1)

- Broadcasting: Takes 1 chunk of variable and send an exact copy to all processes



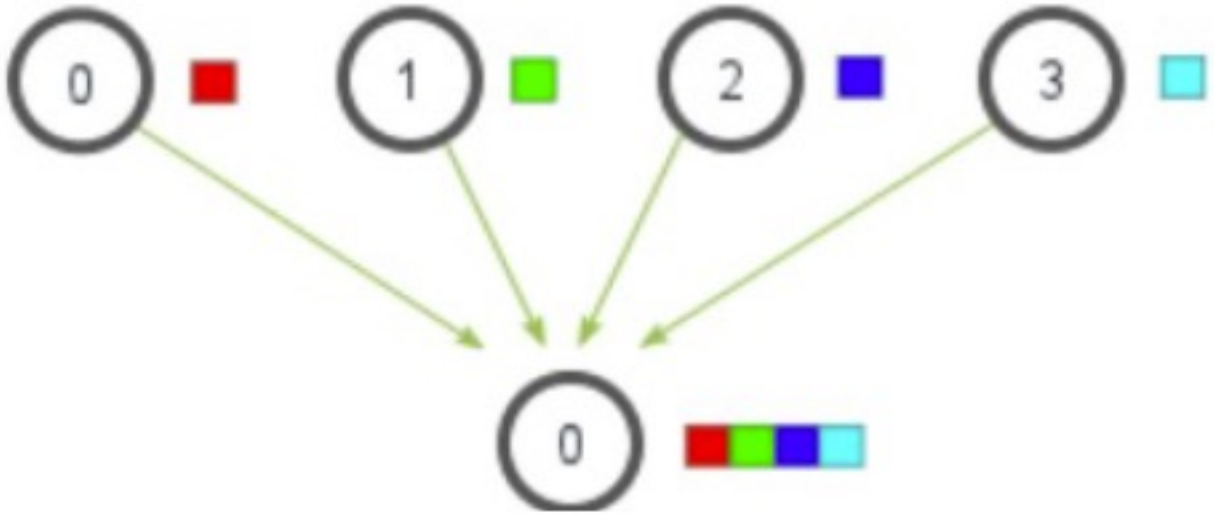
Collective communication (2)

- Scattering: Takes an array and distributes section across ranks



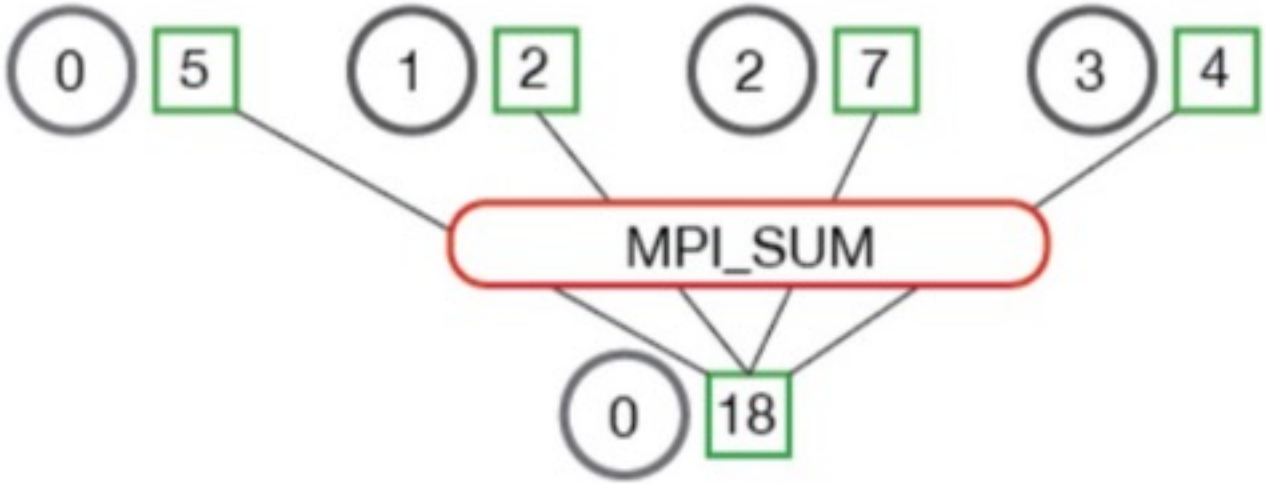
Collective communication (3)

- Gathering: Is the reverse of scattering



Collective communication (4)

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



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

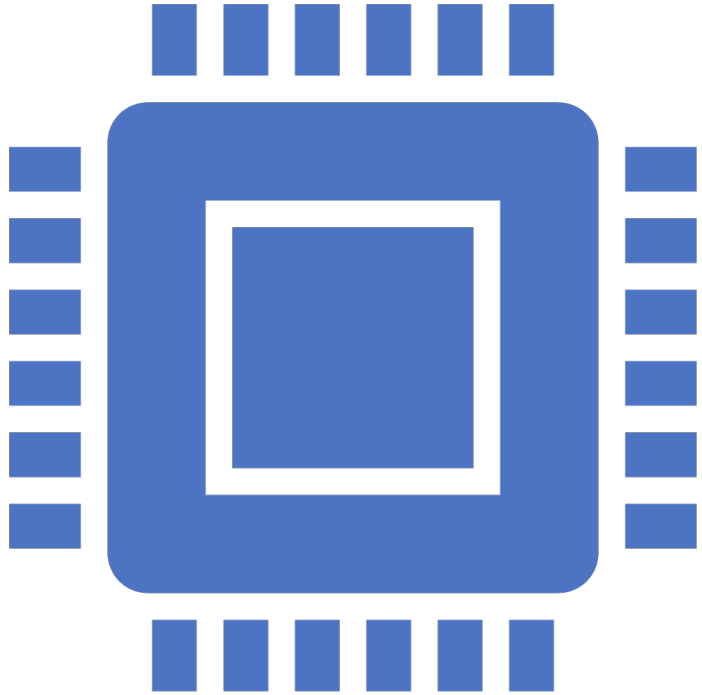
Collective communication with Horovod

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

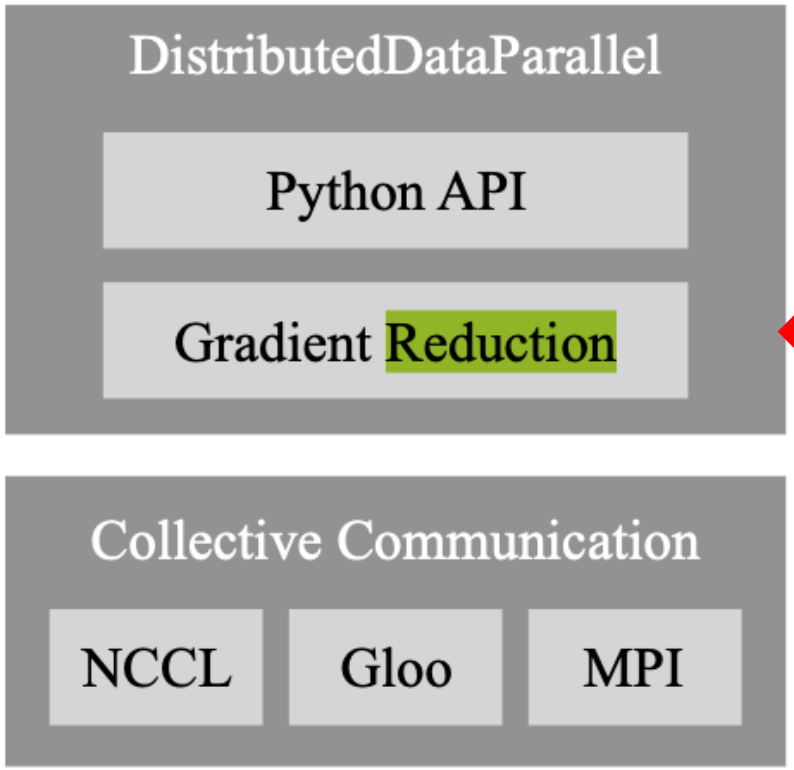
- NVIDIA Collective Communication Library (NCCL) with multi-node & multi GPU.

- GLOO developed by Facebook AI

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



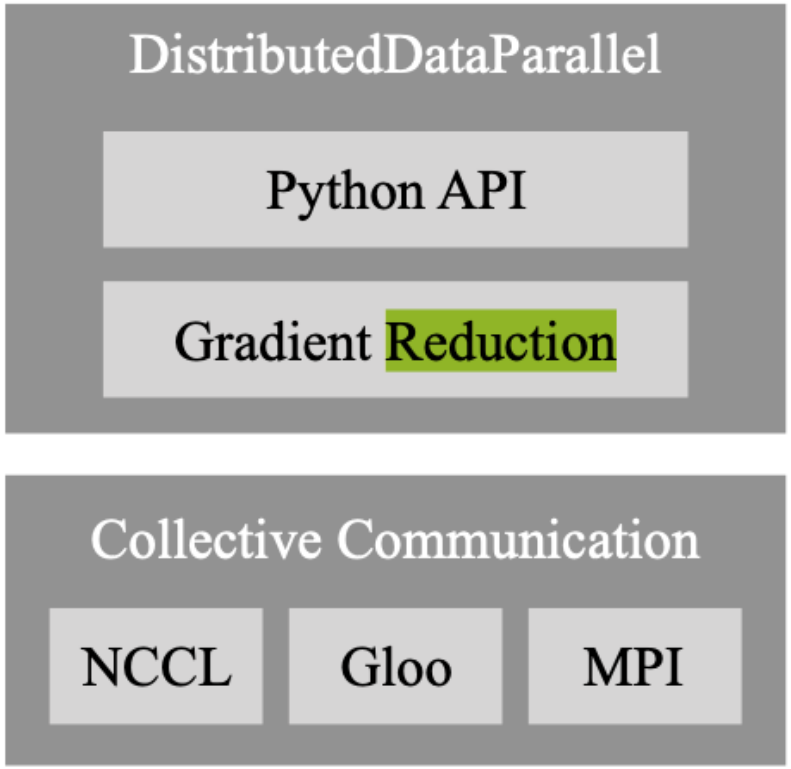
Collective communication with ibuted



AllReduce to calculate average gradients on each parameter across all processes -> gradient tensor

Source: "Pytorch Distributed: Experiences on Accelerating Data Parallel Training

Collective communication with ibuted



Communication backends

Source: "Pytorch Distributed: Experiences on Accelerating Data Parallel Training

Install Horovod on Alpine

- Install the Deep learning Framework (Tensorflow, Pytorch)
- Load Openmpi
- Install `gxx_linux-64`
- Use spack to install nccl (`cuda_arch=80`) or rccl for AMD.
- RCCL is already installed on Alpine
- Follow this guide: https://horovod.readthedocs.io/en/latest/gpus.html

Install Horovod on Alpine

Please open script **part9-install_horovod_amd_gpu.sh**

```
# We export all the ROCM related path and libraries
export ROCM_PATH=/curc/sw/install/rocm/5.2.3
export HIP_PATH=$ROCM_PATH
export HIP_ROOT_DIR=$ROCM_PATH
export ROCM_LIBRARIES=$ROCM_PATH/lib
export PYTHONPATH=$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
```

Install Horovod on Alpine

Please open script **part9-install_horovod_amd_gpu.sh**

```
# 1) Installing GXX and exporting the path:
module load mambaforge
mamba create --name gxx_horovod_amd -c anaconda gxx_linux-64 -y

echo "*************Activating environment*************"
mamba activate gxx_horovod_amd

echo "*************Environment activated **************"
mamba install -c anaconda gxx_linux-64 -y
mamba deactivate

echo "********** Environment deactivated *************"
module unload mambaforge

export GXX_PATH=/projects/$USER/software/anaconda/envs/gxx_horovod_amd
export LD_LIBRARY_PATH=$LIBRARY_PATH:$GXX_PATH/lib
export PATH=$PATH:$GXX_PATH/bin:$GXX_PATH/include

# 2) Loading the appropriate module dependencies for Horovod:
module load gcc openmpi cuda cmake
```

Install Horovod on Alpine

Please open script **part9-install_horovod_amd_gpu.sh**

```
#export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/projects/keto9343/software/spack/opt/spack/linux-rhel8-zen/gcc-8.4.1/nccl-2.18.3-1-2pmuetqe

export RCCL_INCLUDE_DIRS=/curc/sw/install/rocm/5.2.3/include:/curc/sw/install/rocm/5.2.3/rccl/include
export LD_LIBRARY_PATH=/curc/sw/install/rocm/5.2.3/lib:/curc/sw/install/rocm/5.2.3/rccl/lib:$LD_LIBRARY_PATH
export PATH=/curc/sw/install/rocm/5.2.3/rccl:/curc/sw/install/rocm/5.2.3/include:/curc/sw/install/rocm/5.2.3/rccl/include:$PATH

export HOROVOD_RCCL_INCLUDE=/curc/sw/install/rocm/5.2.3/include:/curc/sw/install/rocm/5.2.3/rccl/include
export HOROVOD_RCCL_LIB=/curc/sw/install/rocm/5.2.3/lib:/curc/sw/install/rocm/5.2.3/rccl/lib
export HOROVOD_RCCL_HOME=/curc/sw/install/rocm/5.2.3/rccl
# 4) Installing Horovod
export HOROVOD_INSTALL_DIR=$PIP_INSTALL_DIR          #/projects/$USER/software/horovod_amd_install
export HOROVOD_CMAKE_INSTALL_PREFIX=$HOROVOD_INSTALL_DIR/bin


echo "*************Installing Horovod ******************"
```

Install Horovod on Alpine

Please open script **part9-install_horovod_amd_gpu.sh**

```
export HSA_OVERRIDE_GFX_VERSION=9.0.8
export SDL_GFX_LIBRARIES=/curc/sw/install/rocm/5.2.3/lib/rocblas/library/Kernels.so-000-gfx908-xnack-.hsaco
export SDL_GFX_LIBRARIES=$SDL_GFX_LIBRARIES:/curc/sw/install/rocm/5.2.3/rdc/lib/hsaco/gfx908/gpuReadWrite_kernels.hsaco:/curc/sw/install/
export SDL_GFX_LIBRARIES=$SDL_GFX_LIBRARIES:/curc/sw/install/rocm/5.2.3/llvm/lib-debug/libomptarget-new-amdgpu-gfx908.bc:/curc/sw/install
export SDL_GFX_LIBRARIES=$SDL_GFX_LIBRARIES:/curc/sw/install/rocm/5.2.3/lib/rocblas/library/TensileLibrary_gfx908.co
export NVTX_INCLUDE_DIR=/curc/sw/cuda/12.1.1/include


HSA_OVERRIDE_GFX_VERSION=9.0.8 CMAKE_HIP_ARCHITECTURES="gfx908"  CMAKE_HIP_PLATFORM="gfx908" ROCM_VERSION=5.2.3 \
__HIP_PLATFORM_AMD__=1  HOROVOD_CPU_OPERATIONS=MPI PYTORCH_ROCM_ARCH="gfx908" \
AMDGPU_TARGET="gfx908" \
AMDGPU_TARGETS="gfx908" GFX_ARCH="gfx908" HAVE_GPU=1  HOROVOD_WITH_PYTORCH=1 HAVE_ROCM=1 Pytorch_ROCM=1 \
HOROVOD_GPU_ALLREDUCE=NCCL  HOROVOD_GPU=ROCM HOROVOD_ROCM_HOME=$ROCM_PATH HOROVOD_WITH_MPI=1 \
pip install --no-cache-dir  --target=$PIP_INSTALL_DIR  -v git+https://github.com/horovod/horovod.git >& make_horovod_update_V15.log &

#  --target=$PIP_INSTALL_DIR  -v git+https://github.com/horovod/horovod.git@refs/pull/3588/head --upgrade

#Exporting the install directory
export PATH=$PATH:$HOROVOD_CMAKE_INSTALL_PREFIX/bin

# 5) Finally checking the build:
echo "**************Checking Horovod build ********************"
horovodrun --check-build
```

Important point(1)

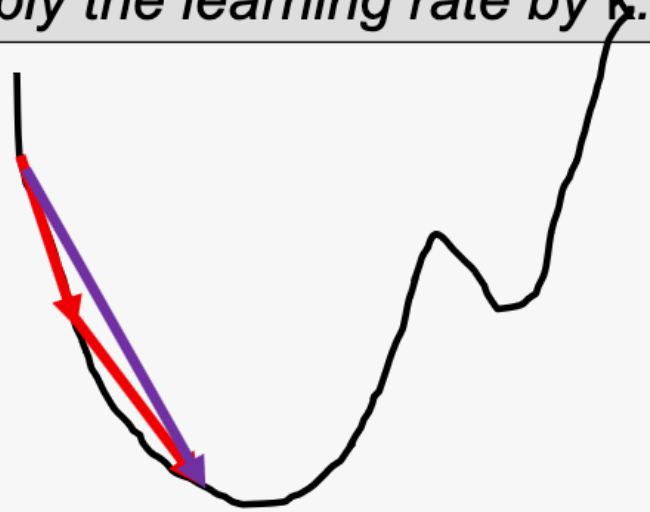
Linear rate scaling rule

When the minibatch size is multiplied by k, *multiply the learning rate by* k.

Mini-batch stochastic Gradient Descent

$$w_{t+1} = w_t - \eta \frac{1}{n} \sum_{x \in \mathcal{B}} \nabla l(x, w_t)$$

Learning rate, lr Mini-batch



lr_{scale} = lr*nprocs

Typical practice / suggestion:

- keep local batch size per worker, i.e., increase the global batch size linearly
- Increase the learning rate proportionally

Actuality: need to adjust the batch size and learning rate at different scale

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

Important point (1)

Local batch size represents the batch size on each rank

Global batch size is the aggregation of all the ranks batch size

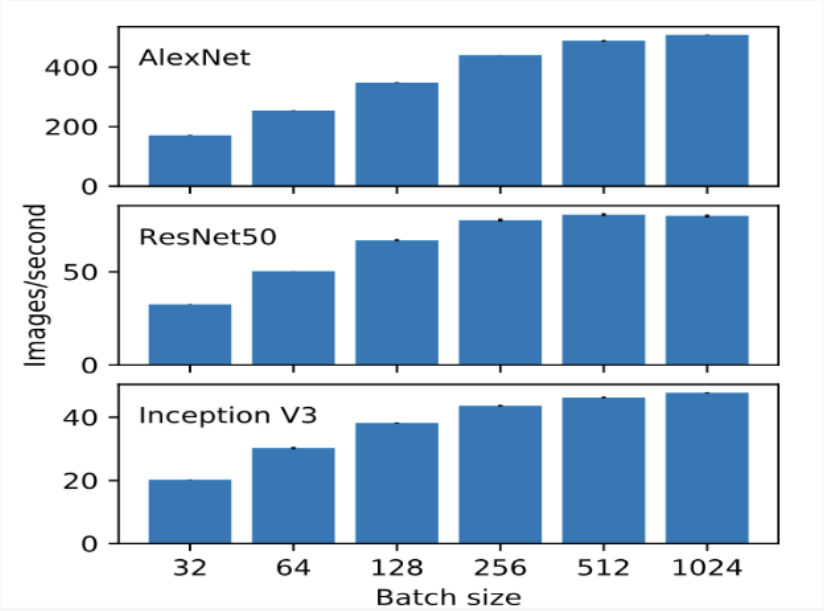
Important point(2)

Large minibatch training

$$w_{t+1} = w_t - \eta \frac{1}{n} \sum_{x \in \mathcal{B}} \nabla l(x, w_t)$$

Minibatch

- Option 1. Keeping the same global minibatch size with each worker processing B/N batch (strong scaling)
- Option 2. Increasing the global minibatch size by N times, so that each worker processes batches of size B (week scaling)



Per node throughput of different local batch size

H. Zheng, https://www.alcf.anl.gov/files/Zheng_SDL_ML_Frameworks_1.pdf

Source: ATPESC 2022, https://extremecomputingtraining.anl.gov/wp-content/uploads/sites/96/2022/11/ATPESC-2022-Track-8-Talk-4-Velusamy-DataParallel-Updated.pdf

Horovod code snipet Pytorch

```
#…
import torch.nn as nn
import horovod.torch as hvd
hvd.init()
train_dataset = datasets.MNIST('data-%d' % hvd.rank(), train=True, download=True,
                    transform=transforms.Compose([
                        transforms.ToTensor(),
                        transforms.Normalize((0.1307,), (0.3081,))
                    ]))
train_sampler = torch.utils.data.distributed.DistributedSampler(
    train_dataset, num_replicas=hvd.size(), rank=hvd.rank())
train_loader = torch.utils.data.DataLoader(
    train_dataset, batch_size=args.batch_size, sampler=train_sampler, **kwargs)
# Horovod: broadcast parameters.
hvd.broadcast_parameters(model.state_dict(), root_rank=0)
# Horovod: scale learning rate by the number of GPUs.
optimizer = optim.SGD(model.parameters(), lr=args.lr * hvd.size(),
                      momentum=args.momentum)
# Horovod: wrap optimizer with DistributedOptimizer.
optimizer = hvd.DistributedOptimizer(
    optimizer, named_parameters=model.named_parameters())
```

Transforms to preprocess data

Source: ATPESC 2022, https://extremecomputingtraining.anl.gov/wp-content/uploads/sites/96/2022/11/ATPESC-2022-Track-8-Talk-4-Velusamy-DataParallel-Updated.pdf

Horvod code snipet Pytorch

```
#…
import torch.nn as nn
import horovod.torch as hvd
hvd.init()
train_dataset = datasets.MNIST('data-%d' % hvd.rank(), train=True, download=True,
                    transform=transforms.Compose([
                        transforms.ToTensor(),
                        transforms.Normalize((0.1307,), (0.3081,))
                    ]))
train_sampler = torch.utils.data.distributed.DistributedSampler(
    train_dataset, num_replicas=hvd.size(), rank=hvd.rank())
train_loader = torch.utils.data.DataLoader(
    train_dataset, batch_size=args.batch_size, sampler=train_sampler, **kwargs)
# Horovod: broadcast parameters.
hvd.broadcast_parameters(model.state_dict(), root_rank=0)
# Horovod: scale learning rate by the number of GPUs.
optimizer = optim.SGD(model.parameters(), lr=args.lr * hvd.size(),
                      momentum=args.momentum)
# Horovod: wrap optimizer with DistributedOptimizer.
optimizer = hvd.DistributedOptimizer(
    optimizer, named_parameters=model.named_parameters())
```

Loading the data in a distributed fashion

Source: ATPESC 2022, https://extremecomputingtraining.anl.gov/wp-content/uploads/sites/96/2022/11/ATPESC-2022-Track-8-Talk-4-Velusamy-DataParallel-Updated.pdf

Horvod code snipet Pytorch

```
#…
import torch.nn as nn
import horovod.torch as hvd
hvd.init()
train_dataset = datasets.MNIST('data-%d' % hvd.rank(), train=True, download=True,
                    transform=transforms.Compose([
                        transforms.ToTensor(),
                        transforms.Normalize((0.1307,), (0.3081,))
                    ]))
train_sampler = torch.utils.data.distributed.DistributedSampler(
    train_dataset, num_replicas=hvd.size(), rank=hvd.rank())
train_loader = torch.utils.data.DataLoader(
    train_dataset, batch_size=args.batch_size, sampler=train_sampler, **kwargs)
# Horovod: broadcast parameters.
hvd.broadcast_parameters(model.state_dict(), root_rank=0)
# Horovod: scale learning rate by the number of GPUs.
optimizer = optim.SGD(model.parameters(), lr=args.lr * hvd.size(),
                      momentum=args.momentum)
# Horovod: wrap optimizer with DistributedOptimizer.
optimizer = hvd.DistributedOptimizer(
    optimizer, named_parameters=model.named_parameters())
```

Broadcast params

Source: ATPESC 2022, https://extremecomputingtraining.anl.gov/wp-content/uploads/sites/96/2022/11/ATPESC-2022-Track-8-Talk-4-Velusamy-DataParallel-Updated.pdf

Horvod code snipet Pytorch

```
#…
import torch.nn as nn
import horovod.torch as hvd
hvd.init()
train_dataset = datasets.MNIST('data-%d' % hvd.rank(), train=True, download=True,
                    transform=transforms.Compose([
                        transforms.ToTensor(),
                        transforms.Normalize((0.1307,), (0.3081,))
                    ]))
train_sampler = torch.utils.data.distributed.DistributedSampler(
    train_dataset, num_replicas=hvd.size(), rank=hvd.rank())
train_loader = torch.utils.data.DataLoader(
    train_dataset, batch_size=args.batch_size, sampler=train_sampler, **kwargs)
# Horovod: broadcast parameters.
hvd.broadcast_parameters(model.state_dict(), root_rank=0)
# Horovod: scale learning rate by the number of GPUs.
optimizer = optim.SGD(model.parameters(), lr=args.lr * hvd.size(),
                      momentum=args.momentum)
# Horovod: wrap optimizer with DistributedOptimizer.
optimizer = hvd.DistributedOptimizer(
    optimizer, named_parameters=model.named_parameters())
```

Wrap optimizer in hvd optimizer

Source: ATPESC 2022, https://extremecomputingtraining.anl.gov/wp-content/uploads/sites/96/2022/11/ATPESC-2022-Track-8-Talk-4-Velusamy-DataParallel-Updated.pdf

Special Thanks

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

Questions?