Multi GPU on ROCm

ROCm Tutorial – Part 6
Multi-GPU Systems

- DNN training
- Large-scale physics simulation
- Bitcoin mining
Multi-GPU Computing Use Cases

- Deep Neural Training
- Weather Forecasting
- Bitcoin Mining
Introduction

1. Single GPU is no longer sufficient for many modern day workloads
   • Deep Neural Training, Weather Forecasting, Bitcoin Mining

2. Multi-GPU systems are becoming pervasive in computing centres

3. AMD’s Instinct series of GPUs provide:
   • 24.6 TFlops for FP16
   • 12.32 TFlops for FP32 data

4. ROCm seamlessly provides support for multi-GPU computing
   • Rich set of libraries
   • Driver and runtime support

5. In this tutorial, we will take a look at leveraging multi-GPU systems with ROCm
Prerequisites

1. This tutorial assumes you have a working ROCm installation

2. Ensure that:
   1. There is a working MPI installation
   2. RCCL library is installed( [https://github.com/ROCmSoftwarePlatform/rccl](https://github.com/ROCmSoftwarePlatform/rccl) )

3. Basic understanding of:
   - HIP APIs
   - Compiling HIP applications
   - NCCL/RCCL concepts
   - MPI concepts

4. System with multiple-GPUs is required
Online Guides

1. All online guides for ROCm can be found here ([https://rocmdocs.amd.com/en/latest/](https://rocmdocs.amd.com/en/latest/))


4. MPI refresher ([https://mpitutorial.com](https://mpitutorial.com))
Goals

- To understand ROCm support for Multi-GPU systems
- Writing a multi-GPU application using the RCCL library
- Writing a multi-GPU application using MPI
Multi-GPU Programming

Multi-GPU programs in ROCm rely on two widely used communication libraries:

- Message Passing Interface (MPI)
- RCCL (ROCm Collective Communications library)

These two libraries are widely used in today’s HPC centers for accelerating scientific applications and Machine Learning workloads.

They provide an easy to use interface to communicate between multiple nodes that host multiple GPUs.

We will be looking at examples for each of these libraries in the hands-on tutorial for this section.
An Example: Data Parallelism DNN Training
An Example: Data Parallelism DNN Training
An Example: Data Parallelism DNN Training

- Model
- Model
- Model
- Model
An Example: Data Parallelism DNN Training

Training Set | Model
---|---
Training Set | Model
Training Set | Model
Training Set | Model
An Example: Data Parallelism DNN Training
An Example: Data Parallelism DNN Training

Model

Model

Model

Model

Gradient

Gradient

Gradient

Gradient
An Example: Data Parallelism DNN Training

Model → Gradient
Model → Gradient
Model → Gradient
Model → Gradient
An Example: Data Parallelism DNN Training

Model

Gradient

Gradient

Gradient

Gradient

Gradient

Model
An Example: Data Parallelism DNN Training
Multi-GPU RCCL with ROCm
RCCL

1. RCCL is a widely used communications primitive library
   - Used widely in Deep Neural Network training

2. Support a host of widely used multi-GPU fabrics

3. Automatic topology detection for high BW paths

4. RCCL uses the same C API as NCCL
   - NCCL APIs do not need to be converted
Operations Supported by RCCL

- All Reduce
- Broadcast
- Reduce
- All Gather
- Reduce Scatter

RCCL implements the NCCL API standard

- Code written with the NCCL library does not need to be rewritten
ncclResult_t ncclBroadcast(
    const void* sendbuff,
    void* recvbuff, size_t count,
    ncclDataType_t datatype,
    int root,
    ncclComm_t comm,
    cudaStream_t stream)
Reduce

ncclResult_t ncclReduce(
    const void* sendbuff,
    void* recvbuff,
    size_t count,
    ncclDataType_t datatype,
    ncclRedOp_t op,
    int root,
    ncclComm_t comm,
    cudaStream_t stream)

out[i] = sum(inX[i])
All Reduce

\[
\text{out}[i] = \text{sum}(\text{inX}[i])
\]

\[
\text{ncclResult_t ncclAllReduce}(
\quad \text{const void* sendbuff,}
\quad \text{void* recvbuff,}
\quad \text{size_t count,}
\quad \text{ncclDataType_t datatype,}
\quad \text{ncclRedOp_t op,}
\quad \text{ncclComm_t comm,}
\quad \text{cudaStream_t stream})
\]
ncclResult_t ncclAllGather(
    const void* sendbuff,
    void* recvbuff,
    size_t sendcount,
    ncclDataType_ datatype,
    ncclComm_t comm,
    cudaStream_t stream)
Reduce Scatter

ncclResult_t ncclReduceScatter(
    const void* sendbuff,
    void* recvbuff,
    size_t recvcount,
    ncclDataType_t datatype,
    ncclRedOp_t op,
    ncclComm_t comm,
    cudaStream_t stream)
Example Application

1. In this tutorial, we are going to look at a simple example to use RCCL on ROCm.

2. The example application performs an AllReduce Sum operation in a Multi-GPU system consisting of 2 devices.

3. AllReduce is one of the most common operations in DNN training.

4. It is important to note that RCCL uses the same named APIs as NCCL.
   - If you are porting NCCL application to HIP, you do not have to translate NCCL APIs to RCCL.

![Diagram of the example application]

\[
\text{out}[i] = \text{sum}(\text{inX}[i])
\]
Demo: Demonstration of using RCCL to calculate the element-wise sum.
Multi-GPU MPI with ROCm
MPI with ROCm

1. ROCm supports multi-GPU computing with the popular Message Passing Interface (MPI)

2. MPI is widely used to scale to multiple nodes in HPC applications

3. Most popular HPC applications rely on multi-GPU MPI programming models to scale their workloads

4. Porting from CUDA+MPI to HIP+MPI is very easy
   - Done using HIP translator tools covered in module 4
MPI scatter basically takes a group of elements (such as an array), splits and distributes it to other processes as shown below.
MPI gather takes elements from each processes and gathers them to the root process as shown below.
Example Application: Scatter, Gather, and Reduce

- The example application for this tutorial follows a typical scatter, gather and reduce approach, common in MPI applications.
- The random numbers are first generated and then divided into chunks based on the number of processes specified.
- The root node scatters these elements to the multiple GPUs.
- Each GPU runs a kernel that squares this numbers.
- The root node then gathers these values from the GPUs.
- Finally, the root node does a local reduction to produce the result.
Demo: Demonstration of Multi-GPU MPI with ROCm
MPI vs RCCL

1. The choice of MPI vs RCCL is usually dependent on the problem at hand

2. Typically HPC applications are written in MPI
   • Helps maintain legacy behavior of old HPC codes

3. Deep Neural Network training is done using RCCL

4. However one can be used in place of the other

5. Popular DNN frameworks support both MPI and RCCL in their backend
Conclusion

In this module, we looked at the support provided by ROCm for Multi-GPU systems.

We looked at writing, compiling and running multi-GPU applications using libraries such as MPI and RCCL.

Multi-GPU scaling is still a big challenge and provides interesting research opportunities for both hardware and software designers.

AMD continues to innovate in this space with state-of-the-art software and hardware support for multi-GPU systems.