CSCI-4320/6360 - Assignment 4: Hybrid Parallel *Reduction* Using CUDA and MPI

Christopher D. Carothers
Department of Computer Science
Rensselaer Polytechnic Institute
110 8th Street
Troy, New York U.S.A. 12180-3590

March 13, 2023

DUE DATE: 11:59 a.m., Friday, March 24th, 2022

1 Overview

For Assignment 4, you are to extend the CUDA Reduction implementation to running across multiple GPUs and compute nodes using MPI. You will run your hybrid parallel CUDA/MPI C-program on the *AiMOS* supercomputer at the CCI in parallel using at most 8 compute nodes for a total of 48 GPUs and compare the results of using up to 8 compute nodes, CPU only `MPI_Reduce` configured with up to 256 MPI ranks (32 MPI ranks per compute node like before in Assignment 3).

Like Assignment 3, there will be a large distributed array except here is the array is of size 1,610,612,736 "double" elements or $48x32^5$ "double" elements. This way, the array will divide evenly across all MPI rank/CUDA device configurations. The array is initialized the same way as in Assignment 3 except the values will be double precision floating point. The answer it is easy to compute via the equation $N * N - 1/2$ which is $1.2970367e+18$. Note that $2^{63} - 1$ is $9.22337233e+18$ and so the sum result will fit in 64 bit double precision number.

Here, there will be two ways in which the sum reduction will be computed. First is *local CPU reduce* which is where each MPI rank will compute their own local sum reduction of the big distributed array. Next, each MPI rank will call `MPI_Reduce()` to compute the full array sum value (again double precision). The second approach is *local GPU reduce* where each MPI rank will invoke the `reduce7()` kernel on a separate CUDA/GPU device and then perform a `MPI_Reduce` across all the local sum values to arrive at the full array sum value. A couple of notes. First, the CUDA implementation does rely on the CPU doing a sum across the blocks. For your implementation it is fine if you only call the `reduce()` CUDA function once and do the remaining part of the sum on the CPU. Second, there are 6 CUDA/GPU devices per compute-node on AiMOS for the standard `el8` partition.

1.1 Review of CUDA Reduction

Recall, that the CUDA reduction code is part of the NVIDIA "samples" and is located in: `/usr/local/cuda-11.2/samples/6_Advanced/reduction/`. **Note, you still need to load the CUDA module using `module load`.** Using that code as a starting point, pull the `reduce7()` templated function into your hybrid CUDA/MPI C-program's CUDA code specific file `cuda-reduce.cu`.

Below is an example from `nvprof` as well as a parallel run on 4 GPUs.

```
(base) [SPNRcaro@dcs223 MPI-CUDA-Reduction]$ export OMPI_COMM_WORLD_RANK=1
(base) [SPNRcaro@dcs223 MPI-CUDA-Reduction]$ nvprof ./reduce-exe
--------------------------------------------------------------------------
WARNING: There was an error initializing an OpenFabrics device.

  Local host:   dcs223
  Local device: mlx5_1
--------------------------------------------------------------------------
==544866== NVPROF is profiling process 544866, command: ./reduce-exe
Mapping Rank 0 to CUDA Device 0
MPI Rank 0, completed CUDA init
CUDA Reduce starting ...threads 1024, blocks 1572864, size 1610612736
CUDA Reduce completed ...Summing 1572864 elements on CPU
MPI Rank 0: Global Sum is 1297036691877396480.000000 in 3.574205 secs
==544866== Profiling application: ./reduce-exe
==544866== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:  100.00%  3.56638s         1  3.56638s  3.56638s  3.56638s  void reduce7<double,
                                                                                unsigned int=1024,
                                                                                bool=0>
                                                                                (double const *,
                                                                                 double*,
                                                                                 unsigned int)
      API calls:   93.97%  3.56640s         1  3.56640s  3.56640s  3.56640s  cudaDeviceSynchronize
                    5.87%  222.61ms         2  111.30ms  57.404us  222.55ms  cudaMallocManaged
                    0.10%  3.7617ms         4  940.42us  838.64us  1.2355ms  cuDeviceTotalMem
                    0.06%  2.1537ms       404  5.3310us     193ns  241.10us  cuDeviceGetAttribute
                    0.01%  198.29us         4  49.572us  46.791us  54.781us  cuDeviceGetName
                    0.00%  99.418us         1  99.418us  99.418us  99.418us  cudaLaunchKernel
                    0.00%  20.670us         6  3.4450us     512ns  17.255us  cuPointerGetAttributes
                    0.00%  8.7850us         4  2.1960us  1.1130us  3.8670us  cuDeviceGetPCIBusId
                    0.00%  4.8590us         1  4.8590us  4.8590us  4.8590us  cudaSetDevice
                    0.00%  3.6350us         8     454ns     246ns  1.4750us  cuDeviceGet
                    0.00%  1.3980us         3     466ns     352ns     653ns  cuDeviceGetCount
                    0.00%  1.3750us         4     343ns     313ns     412ns  cuDeviceGetUuid
                    0.00%  1.1600us         1  1.1600us  1.1600us  1.1600us  cudaGetDeviceCount

==544866== Unified Memory profiling result:
Device "Tesla V100-SXM2-16GB (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
   41541  303.20KB  64.000KB  960.00KB  12.01172GB  283.6298ms  Host To Device
      60  204.80KB  64.000KB  960.00KB  12.00000MB  312.7660us  Device To Host
   18366         -         -         -           -   3.540507s  Gpu page fault groups
Total CPU Page faults: 73764
```

Example parallel output for a 4 GPU case. *Note, you don't have a 4 GPU case for performance evaluation. This is only an example.*

```
(base) [SPNRcaro@dcs223 MPI-CUDA-Reduction]$ mpirun -np 4 ./reduce-exe
Mapping Rank 3 to CUDA Device 3
Mapping Rank 1 to CUDA Device 1
Mapping Rank 2 to CUDA Device 2
Mapping Rank 0 to CUDA Device 0
```

```
MPI Rank 3, completed CUDA init
CUDA Reduce starting ...threads 1024, blocks 1572864, size 402653184
MPI Rank 1, completed CUDA init
CUDA Reduce starting ...threads 1024, blocks 1572864, size 402653184
MPI Rank 2, completed CUDA init
CUDA Reduce starting ...threads 1024, blocks 1572864, size 402653184
MPI Rank 0, completed CUDA init
CUDA Reduce starting ...threads 1024, blocks 1572864, size 402653184
CUDA Reduce completed ...Summing 393216 elements on CPU
CUDA Reduce completed ...Summing 393216 elements on CPU
CUDA Reduce completed ...Summing 393216 elements on CPU
CUDA Reduce completed ...Summing 393216 elements on CPU
MPI Rank 0: Global Sum is 1297036691877396480.000000 in 0.751331 secs
```

2 Implementation

To initialize MPI in your `main` function (in `mpi-reduce.c`) do:

```
 // MPI init stuff
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  MPI_Comm_size(MPI_COMM_WORLD, &numranks);
```

To initialize CUDA (after MPI has been initialized in `main`) function, do:

```
  if( (cE = cudaGetDeviceCount( &cudaDeviceCount)) != cudaSuccess )
    {
      printf(" Unable to determine cuda device count, error is %d, count is %d\n",
             cE, cudaDeviceCount );
      exit(-1);
    }

  if( (cE = cudaSetDevice( myrank % cudaDeviceCount )) != cudaSuccess )
    {
      printf(" Unable to have rank %d set to cuda device %d, error is %d \n",
             myrank, (myrank % cudaDeviceCount), cE);
      exit(-1);
    }
```

Note, `myrank` is this MPI rank value.

Also, CUDA initialization and CUDA memory allocations needs to be performed by a function that resides in the `cuda-reduce.cu` file. While all the MPI functionality needs to be performed by functions on the `mpi-reduce.c` file. Care must be taken to break up the functionality across these files correctly.

3 What to time?

Using the `clock_read()` function from Assignment 3, time the LOCAL reduction using either CPU or GPU AND the `MPI_Reduce()` function. Recall this is a system clock with a resolution of 512,000,000 ticks per second.

4 Running on AiMOS

4.1 Building Hybrid MPI-CUDA Programs

REMEMBER TO LOAD THE CORRECT IBMXLC, SPECTRUM-MPI and CUDA MODULES!.

Next, you will need to break apart your code into two files. The `reduce-mpi.c` file contains all the MPI C code. The `reduce-cuda.cu` contains all the CUDA specific code. You'll need to make sure cross routines are correctly "extern". Because `nvcc` is a C++ like compiler, you'll need to turn off name mangling for any C functions exported from the CUDA side to the MPI side via `extern ''C'' { function dec }`. Specifically, these are functions called from code in `reduce-mpi.c` and defined in `reduce-cuda.cu`. Next, create your own `Makefile` with the following:

```
all: reduce-mpi.c reduce-cuda.cu
<tab>   mpixlc -O3 reduce-mpi.c -c -o reduce-mpi.o
<tab>   nvcc -O3 reduce-cuda.cu -c -o reduce-cuda.o
<tab>   mpixlc -O3 reduce-mpi.o reduce-cuda.o -o reduce-exe \
          -L/usr/local/cuda-11.2/lib64/ -lcudadevrt -lcudart -lstdc++
```

To debug your code replace the `-O3` compile options with `-g` for C code and `-G` for CUDA code. *Note for nvcc/CUDA debugging, you should turn on both* `-g -G`.

4.2 SLURM Submission Script

The create your own `slurmSpectrum.sh` batch run script (or re-use the one from Assignment 3) with the following:

```
module load spectrum-mpi cuda/11.2

################################################################################################
#
# Launch N tasks per compute node allocated. Per below this launches 6 MPI rank per compute
# taskset insures that hyperthreaded cores are skipped.
################################################################################################
#

taskset -c 0-159:4 mpirun -N 6 /gpfs/u/home/SPNR/SPNRcaro/scratch/MPI-CUDA-Reduction/reduce
```

5 Parallel Performance Analysis and Report

For your report, describe your implementation and how your broke up your code across the CUDA and MPI specific code files and describe how the call graph flow works between CUDA and MPI.

Next, you will compare the performance of the *local CPU reduce* vs. *local GPU reduce* by executing the following cases.

- Local CPU Reduce: 1 compute-node with 32 MPI ranks total with and without DEBUGGing enabled.
- Local CPU Reduce: 2 compute-node with 64 MPI ranks total with and without DEBUGGing enabled.
- Local CPU Reduce: 4 compute-node with 128 MPI ranks total with and without DEBUGGing enabled.
- Local CPU Reduce: 8 compute-node with 256 MPI ranks total with and without DEBUGGing enabled.
- Local GPU Reduce: 1 compute-node with 6 MPI ranks total using 6 GPU devices with and without DEBUGGing enabled.
- Local GPU Reduce: 2 compute-node with 12 MPI ranks total using 12 GPU devices with and without DEBUGGing enabled.
- Local GPU Reduce: 4 compute-node with 24 MPI ranks total using 24 GPU devices with and without DEBUGGing enabled.
- Local GPU Reduce: 8 compute-node with 48 MPI ranks total using 48 GPU devices with and without DEBUGGing enabled.

When debugging is disable, you should enable `-O3` compiler optimization levels for both CUDA/nvcc and C/mpixlc

Also, notice in the above configurations, the CPU-only cases have more MPI ranks and so the big performance question becomes can the 48 GPUs perform faster than the 256 CPU cores. Other key performance questions includes:

- Determine how much disabling DEBUGGing and OPTIMIZING the code improves performance (e.g., reduces execution time) for both the CPU and GPU cases. This can be expressed as a range of improvement OR average across the CPU and GPU configurations
- For the OPTIMIZED code cases, determine your maximum speedup across all cases relative to using a single compute-node using 32 MPI ranks in total.
- Did GPUs always outperform the CPU cases. Why or why not for your code?
- Finally, explain why you think FASTEST configuration was faster than others.

6 HAND-IN and GRADING INSTRUCTIONS

Please submit your C-code and PDF report with performance data/table to the `submitty.cs.rpi.edu` grading system. All grading will be done manually because Submitty currently does not support GPU programs. Your program should output the reduction sum total (rank 0 only) for correctness. Also, please make sure you document the code you write for this assignment. That is, say what you are doing and why.