Assignment 4

Karan Bhanot, Dhyanjyoti Nath, Abolaji Adesoji, Aditya Joshi

Setup

The files for CUDA and non-CUDA are separate. This means we have a total of 7 files:
Non-CUDA files (In folder: non-cuda):

- **parallelio.c:** The main C file that includes all the read and write operations. It measures time in ticks and then outputs the execution time in seconds
- **Makefile:** Enables compilation of the C code
- **slurmSpectrum.sh:** The script file that runs across MPI ranks

CUDA files (In folder: cuda):

- **parallelio.c:** The main C file that includes all the read and write operations. It measures time in ticks and then outputs the execution time in seconds. It interacts with the CUDA file
- **cuda.cu:** This file has a method to allocate space for reading and writing buffer.
- **Makefile:** Enables compilation of the C & CUDA code and creates an executable for script
- **slurmSpectrum.sh:** The script file that runs across MPI ranks

The **Makefile** creates the object file. Then, we run the shell script for the various configurations of MPI ranks, processor block size and GPUs.

The *slurmSpectrum.sh* file is where we define the arguments that are needed by the object file which requires the size of the processor block as the argument. The file is saved by the name *datafile.txt*, and is then used for reading data in. After each configuration, the data file was removed manually.

For *cudaMallocManaged*, we have to determine the CUDA device id using *mpi_rank* modulo 6. This value is passed when we run the shell script by setting number of GPUs as 6.

Execution

The goal of this assignment was to measure the performance of an MPI parallel I/O C program on a single file on the RPI AiMOS Supercomputer. There was also a comparison shown when CUDA is used in allocating the read and write buffers. Each MPI rank writes/reads 64 different blocks. For each configuration, the bandwidth is also calculated by dividing the total file size by the time and then by 1,000,000,000 to convert it to GB/s.

Table 1 includes the write and read times as well as bandwidth, in GB/sec, for various configurations of MPI ranks and block size, where buffer memory is allocated in regular system memory. Table 2 includes the write and read times as well as bandwidth, in GB/sec, for various configurations of MPI ranks and block size, where buffer memory is allocated by CUDA. Note that total file size is defined by the product of processor block size, number of MPI ranks and 64.

So, if we have a block size of 128000 with 2 MPI ranks, then the file size is 128000*2*64 = 16384000.

Table 1: Write/read time of different MPI ranks and block size (without CUDA)

Total MPI Ranks	Block size	Write time (sec)	Read time (sec)	Write bandwidth (GB/sec)	Read bandwidth (GB/sec)
2	128000	0.005	0.064	3.277	0.256
2	256000	0.007	0.047	4.681	0.697
2	512000	0.01	0.078	6.554	0.840
2	1000000	0.014	0.098	9.143	1.306
2	2000000	0.022	0.125	11.636	2.048
2	4000000	0.179	0.063	2.860	8.127
2	8000000	1.506	0.584	0.680	1.753
2	16000000	2.423	0.425	0.845	4.819
4	128000	0.021	0.117	1.560	0.280
4	256000	0.01	0.089	6.554	0.736
4	512000	0.18	0.009	0.728	14.564
4	1000000	0.181	0.017	1.414	15.059
4	2000000	0.97	0.282	0.528	1.816
4	4000000	1.298	0.416	0.789	2.462
4	8000000	2.823	0.508	0.725	4.031
4	16000000	3.823	0.515	1.071	7.953
8	128000	0.019	0.154	3.449	0.426
8	256000	0.473	0.006	0.277	21.845
8	512000	0.312	0.325	0.840	0.807
8	1000000	4.786	0.132	0.107	3.879
8	2000000	3.499	0.319	0.293	3.210
8	4000000	3.707	0.807	0.552	2.538
8	8000000	3.795	0.797	1.079	5.139
8	16000000	8.259	0.828	0.992	9.894
16	128000	0.535	0.009	0.245	14.564
16	256000	0.423	0.014	0.620	18.725
16	512000	17.213	0.736	0.030	0.712
16	1000000	16.741	0.287	0.061	3.568
16	2000000	17.589	0.409	0.116	5.007
16	4000000	10.787	0.869	0.380	4.713
16	8000000	10.349	1.482	0.792	5.528
16	16000000	13.606	1.701	1.204	9.632

32	128000	0.24	2.594	1.092	0.101
32	256000	12.503	0.428	0.042	1.225
32	512000	31.518	0.955	0.033	1.098
32	1000000	26.135	0.843	0.078	2.429
32	2000000	23.579	1.378	0.174	2.972
32	4000000	21.005	2.031	0.390	4.033
32	8000000	16.678	3.429	0.982	4.778
32	16000000	18.375	4.431	1.783	7.395
64	128000	2.668	0.872	0.197	0.601
64	256000	10.918	0.455	0.096	2.305
64	512000	28.968	0.443	0.072	4.734
64	1000000	31.087	1.202	0.132	3.408
64	2000000	34.422	2.242	0.238	3.654
64	4000000	23.651	2.393	0.693	6.847
64	8000000	19.936	5.22	1.644	6.277
64	16000000	26.133	8.597	2.508	7.623

Table2: Write/read time of different MPI ranks and block size (with CUDA)

Total MPI Ranks	**Block size**	**Write time (sec)**	**Read time (sec)**	**Write bandwidth (GB/sec)**	**Read bandwidth (GB/sec)**
2	128000	0.007	0.045	2.341	0.364
2	256000	0.025	0.1	1.311	0.328
2	512000	0.01	0.377	6.554	0.174
2	1000000	0.018	0.113	7.111	1.133
2	2000000	0.025	0.256	10.240	1.000
2	4000000	1.352	0.025	0.379	20.480
2	8000000	1.239	0.093	0.826	11.011
2	16000000	3.967	0.611	0.516	3.352
4	128000	0.03	0.183	1.092	0.179
4	256000	0.132	0.034	0.496	1.928
4	512000	0.04	0.256	3.277	0.512
4	1000000	0.025	0.332	10.240	0.771
4	2000000	0.865	0.354	0.592	1.446
4	4000000	1.515	0.315	0.676	3.251
4	8000000	4.548	0.377	0.450	5.432
4	16000000	4.041	0.772	1.014	5.306
8	128000	0.045	0.079	1.456	0.830
8	256000	0.04	0.153	3.277	0.857

8	512000	0.077	0.247	3.404	1.061
8	1000000	3.28	0.303	0.156	1.690
8	2000000	5.229	0.467	0.196	2.193
8	4000000	5.602	0.72	0.366	2.844
8	8000000	5.819	0.654	0.704	6.263
8	16000000	13.513	1.589	0.606	5.155
16	128000	0.152	0.703	0.862	0.186
16	256000	1.004	0.488	0.261	0.537
16	512000	31.233	0.431	0.017	1.216
16	1000000	35.668	1.344	0.029	0.762
16	2000000	26.662	1.294	0.077	1.583
16	4000000	15.349	1.432	0.267	2.860
16	8000000	10.764	1.45	0.761	5.650
16	16000000	16.192	3.075	1.012	5.328
32	128000	0.266	3.043	0.986	0.086
32	256000	14.348	0.396	0.037	1.324
32	512000	56.838	1.825	0.018	0.575
32	1000000	80.344	3.382	0.025	0.606
32	2000000	34.87	1.607	0.117	2.549
32	4000000	25.499	2.945	0.321	2.782
32	8000000	18.533	2.574	0.884	6.365
32	16000000	20.222	5.543	1.620	5.912
64	128000	15.752	0.134	0.033	3.913
64	256000	9.452	0.617	0.111	1.699
64	512000	76.98	3.42	0.027	0.613
64	1000000	48.438	2.438	0.085	1.680
64	2000000	47.173	3.215	0.174	2.548
64	4000000	37.215	4.237	0.440	3.867
64	8000000	20.201	4.828	1.622	6.787
64	16000000	31.206	8.948	2.100	7.324

Analysis

To start with the analysis, we plot 4 different plots of data. The four plots are as follows:

1. Write bandwidth for *malloc* as a function of MPI ranks
2. Read bandwidth for *malloc* as a function of MPI ranks
3. Write bandwidth for *cudaMallocManaged* as a function of MPI ranks
4. Read bandwidth for *cudaMallocManaged* as a function of MPI ranks

This implies that Figures 1 and 2 are results without use of CUDA while Figures 3 and 4 are based on CUDA usage. Figure 1 plots the write bandwidth (without cuda) for various block sizes across several MPI ranks ranging from 2 to 64. Similarly, Figure 2 plots the read bandwidth (without cuda) for various block sizes across several MPI ranks. Figure 3 is similar to Figure 1 but here the memory is allocated from CUDA. So, it is basically the plot of write bandwidth (with cuda) for various block sizes across multiple MPI rank configurations. Figure 4 plots the read bandwidth (with cuda) for various block sizes and MPI ranks. All bandwidth values are in GB/sec.

Each plot is a line plot with the x-axis as the number of MPI ranks and y-axis as the bandwidth in GB/sec. Each plot includes 8 lines corresponding to block size of: 128K, 256K, 512K, 1M, 2M, 4M, 8M, and 16M.

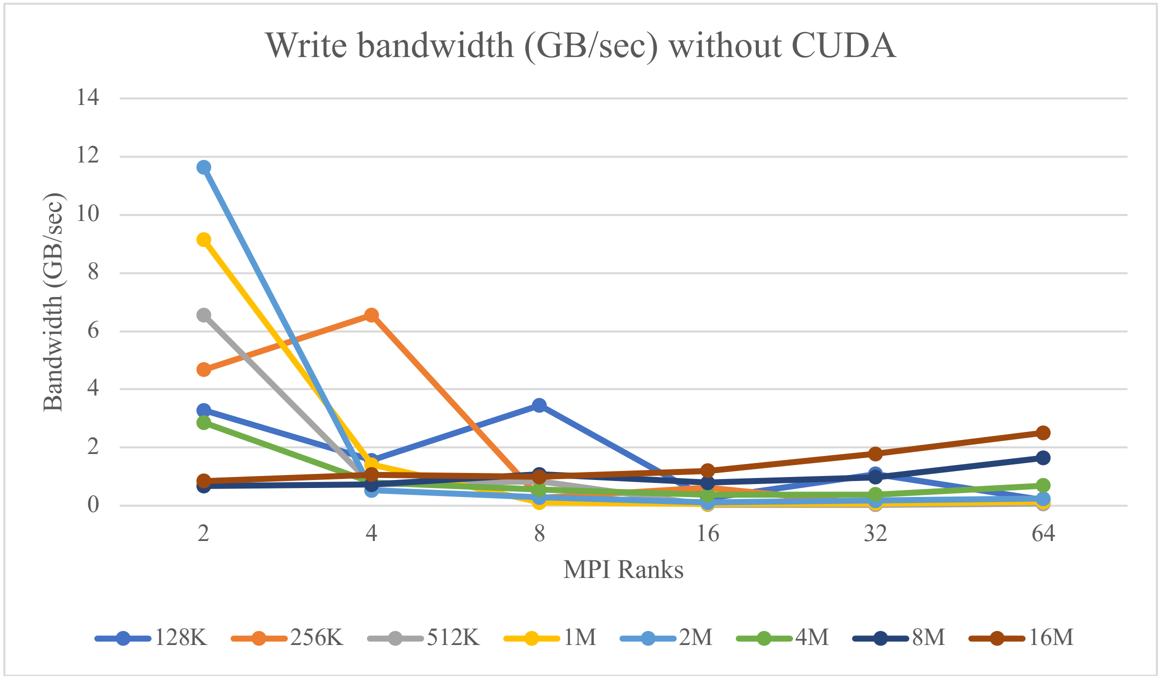

Figure 1: Write bandwidth (GB/sec) without CUDA

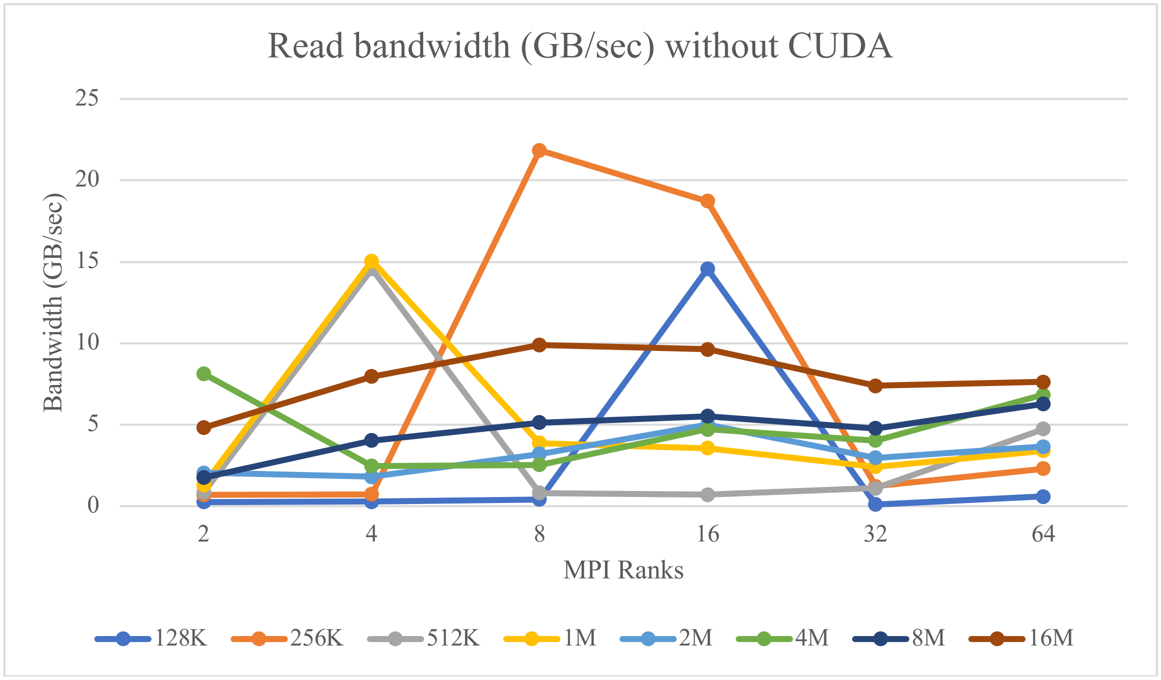

Figure 2: Read bandwidth (GB/sec) without CUDA

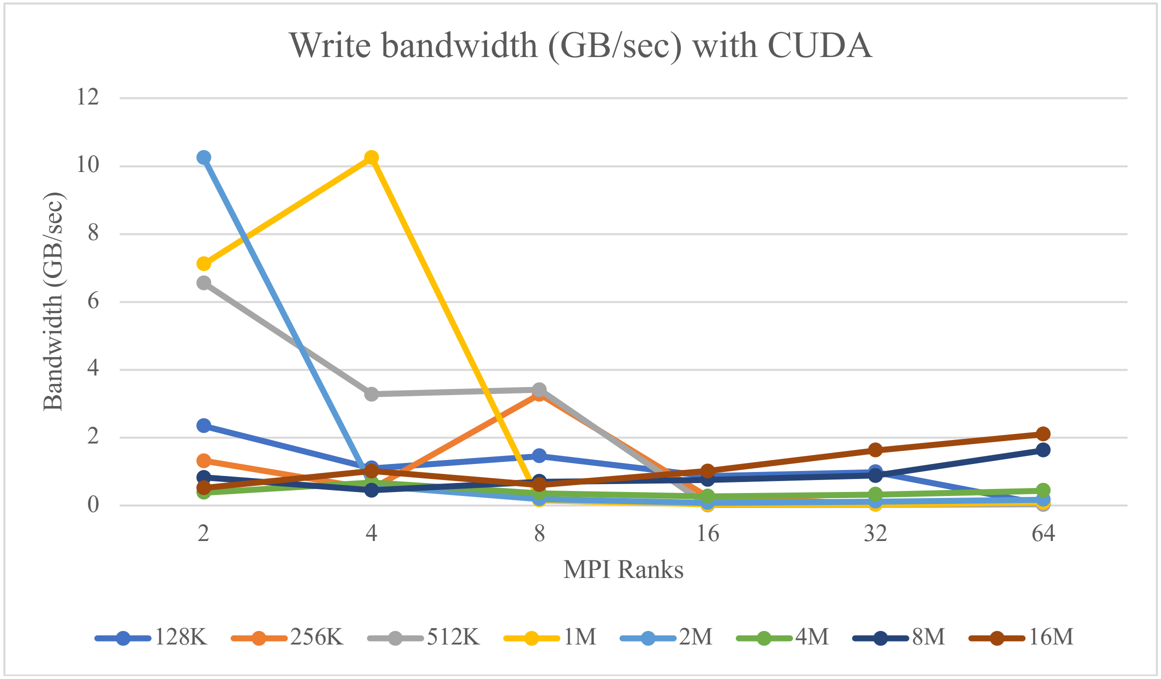

Figure 3: Write bandwidth (GB/sec) with CUDA

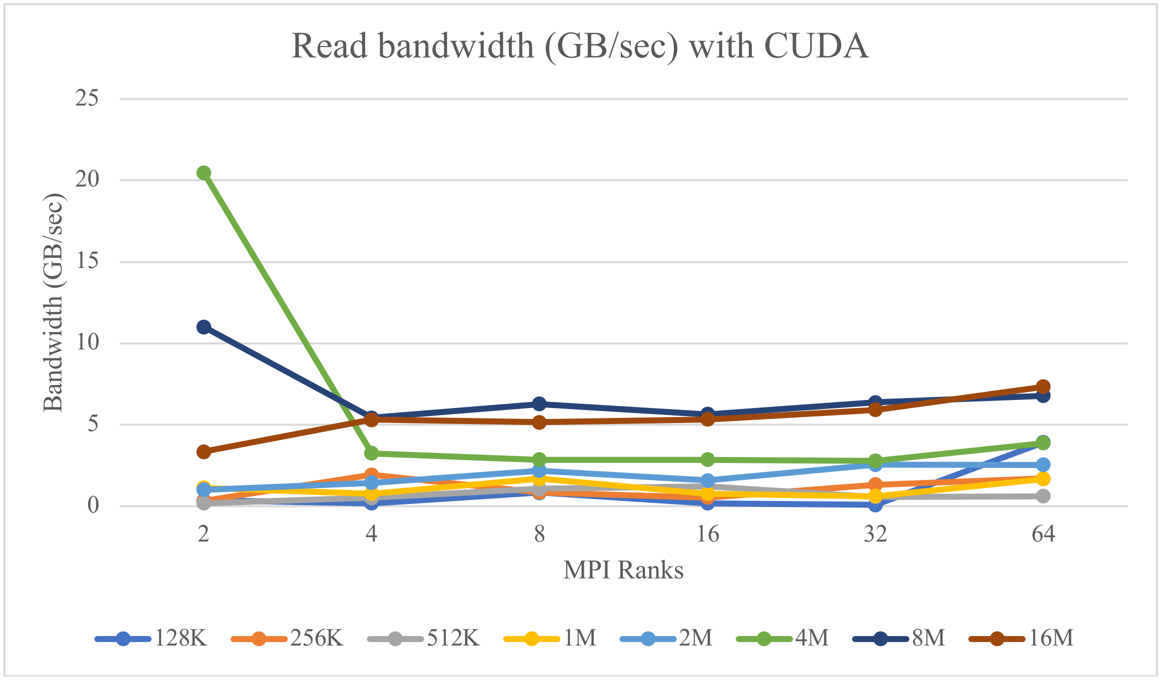

Figure 4: Read bandwidth (GB/sec) with CUDA

From the plots above, we conclude that the bandwidth is highest for almost all block sizes when MPI ranks are set to 2. As we increase the MPI ranks, the bandwidth tends to decrease. This is probably because as we increase MPI ranks, the total size of data read/written also increases, increasing the ticks for the whole process. We also observe that the read bandwidth increases significantly for MPI ranks 8 and 16 when we do not use CUDA. Note that all these values are in GB/sec which means they are quite fast read and write speeds in general. A drop in bandwidth is observed as we increase the number of MPI ranks probably because the values have to be written at sparsely spaced locations as each MPI rank tries to write its own block. The move from one place to the next adds to the time and thus, decreases the overall bandwidth when doing parallel IO with just one file.

Write bandwidth remains almost consistent if memory is allocated from CUDA or not when the MPI ranks is 2. As we increase the MPI ranks, the CUDA results stay higher than regular memory results up till 8 MPI ranks but then both seem to have almost consistent bandwidth. This can be attributed to the fact that once the MPI ranks start increasing, each CUDA device has to manage more MPI ranks than just one and thus the decreased bandwidth. However, in contrast, read bandwidth was higher for regular memory than CUDA memory for ranks 8 and 16.

The results of the experiment reveal that the read and write bandwidths differ significantly from one other. This however is based on the configuration of the number of MPI ranks and the block size. For CUDA, the maximum read bandwidth is 20.5 GB/sec for 4M block size while for non-cuda it is 21.85 GB/sec for 256K block size. For CUDA, the maximum write bandwidth is 10 GB/sec for 2M block size while for non-cuda it is 11.6 GB/sec for 2M block size. If we compare jus the maximum values, we can see that the read bandwidth is twice the write bandwidth. If we compare based on block sizes, the ratio of read to write can vary a lot and can be even 3-4 times faster, the fastest in our case being 21x (8 MPI ranks, 256K block size, non-cuda). There could be two possible reasons why such a behavior happens. Usually disks have faster reads and slower writes and thus, this would most certainly translate when we do our reads and writes, giving the result we observe here. Secondly, writing does not involve any caching, however, reading involves caching. Thus, getting values from cache memory is much faster. As we do a read after a write, some data might still be in cache increasing the speed of reading in the data.

Finally, in most cases (with a few exceptions), the read and write bandwidth tends to decrease as the ranks increase which may be due to increased requirement of coordination and synchronization. However, as we keep increasing the ranks, the bandwidths ever so slightly increase as more and more data is now being written/read but the coordination overhead is not increasing significantly.

We can also take a look the bandwidth per MPI rank as we increase the number of MPI ranks. This could help us also analyze if with the increase in MPI ranks has an effect. Let's take the write bandwidth per MPI rank for CUDA as an example.

From figure 5, we observe that the write speed is indeed higher when MPI ranks are less, meaning each MPI rank is having a higher bandwidth. This decreases as we increase the block size as well as the number of MPI ranks in total. It becomes almost consistent from MPI ranks 16 to 64. This could be possible because as we increase the number of MPI ranks, the coordination overhead also increases amongst them.

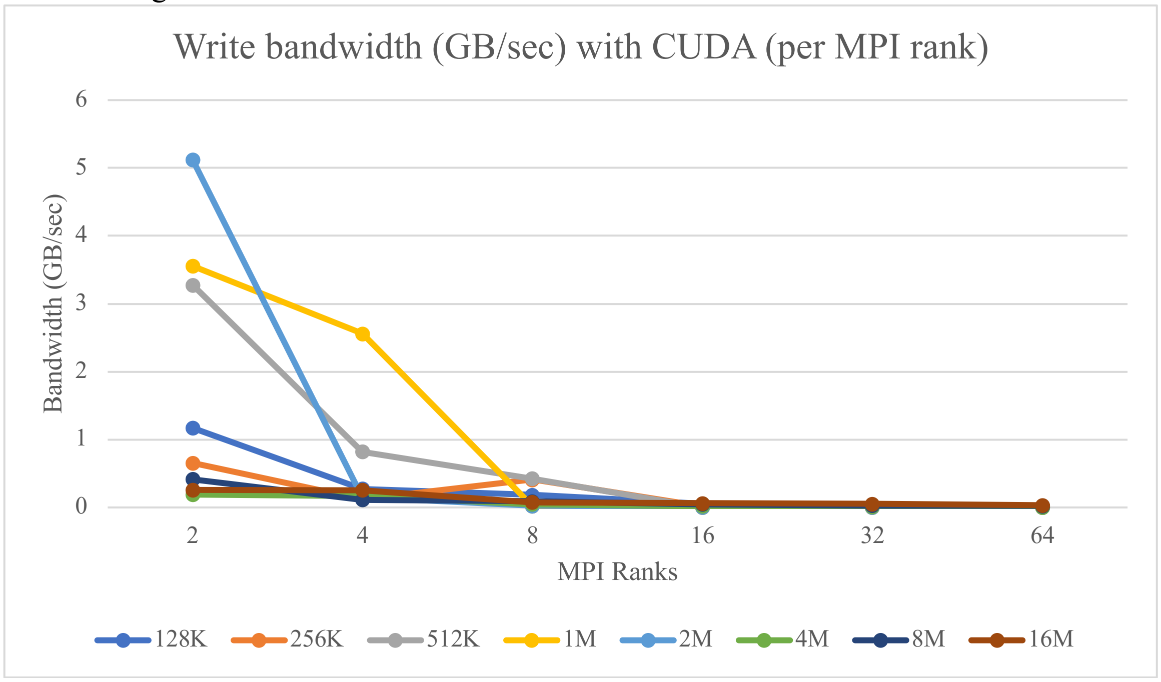

Figure 5: Write bandwidth (GB/sec) with CUDA (per MPI rank)

To ensure the correct number of values are written and read, we check the count value by using MPI_Get_count(). We also tried to remove this code and check for any performance improvement but there was no noticeable effect, so we kept it in.