Optimizing the GPU Allreduce using Multiple Processes per GPU

Michael Adams
mikethebos@unm.edu
University of New Mexico
Albuquerque, New Mexico, USA

Amanda Bienz
bienz@unm.edu
University of New Mexico
Albuquerque, New Mexico, USA

CCS Concepts

• **Computing methodologies** → **Parallel algorithms**; **Massively parallel algorithms**; **Distributed algorithms**; • **Computer systems organization** → **Parallel architectures**.

Keywords

Allreduce, heterogeneous, GPU, MPI, NCCL, RCCL, multi-lane, node-aware

ACM Reference Format:
Michael Adams and Amanda Bienz. 2025. Optimizing the GPU Allreduce using Multiple Processes per GPU. In . ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Emerging exascale systems are comprised of heterogeneous nodes, each typically containing $4-8$ GPUs and dozens to hundreds of CPU cores. Parallel applications achieve high performance through device utilization, accelerating computation on the available GPUs and communicating directly between devices with GPU-aware MPI. Applications typically use a single MPI process per GPU, utilizing one core optimally located near each GPU. While these architectures are equipped with dozens of CPU cores per node, the vast majority of these cores are unused by parallel applications.

The multi-lane all-reduce [1] improves the cost of large reductions through lane-awareness, with each process per node performing a separate portion of the inter-node all-reduce.

This poster presents novel optimizations to large GPU-aware all-reduce operations, extending lane-aware reductions to the GPUs, and notably using multiple CPU cores per GPU to accelerate these operations. These multi-CPU-accelerated GPU-aware lane all-reduces using an intermediate host buffer yield speedup of up to 2.45x for large MPI all-reduces across the NVIDIA A100 GPUs of NCSA's Delta supercomputer. Finally, the approach is extended to GPUDirect RDMA communication, yielding speedup of 1.17x for large all-reduces.

2 Methods

Assuming there are PPG processes per GPU, each process r on a node has a local rank $l_r \leftarrow r \mod \texttt{PPG}$. Each process with local rank $l_r = 0$ is assigned as a leader process. The leader process creates the full buffer to be reduced on its corresponding device. After the buffer is created, the leader extracts its Inter-Process Communication (IPC) memory handle with `(cuda/hip)GetIPCMemHandle`. This memory handle is then broadcast to all processes assigned to the given GPU. Each non-leader process gains access to the shared device pointer by opening the received memory handle, using `(cuda/hip)IpcOpenMemHandle`.

After the initial setup, each process performs an equal portion of the all-reduce asynchronously. On Delta, an intermediate host buffer is used by MPI for communication while GPUDirect RDMA communication with an intermediate device buffer is used on Tuolumne. In this algorithm, the total number of messages among all processes increases by the factor of the number of processes per GPU. However, each message is reduced in size by the same factor. These messages are communicated concurrently. This approach is extended to utilize multi-lane collectives [1] with an intra-node reduce-scatter to shrink buffer sizes during an inter-node all-reduce, leading to a total reduction in inter-node buffer size by the number of MPI processes per node. However, there is an increase in intra-node communication from the reduce-scatter and the subsequent all-gatherv operations.

The figure in the Methods section of the poster shows a ping-pong benchmark on Tuolumne where adding additional processes per GPU yields speedup.

3 Results

The impact of multi-lane and multi-process per GPU all-reduce algorithms is analyzed against OpenMPI. Cray MPICH is analyzed on LLNL's Tuolumne.

The poster's bottom left figure shows the multi-lane algorithm is further accelerated with increasing numbers of processes per GPU. Large all-reduce operations across all 4 GPUs on 8 nodes yield around a 1.39x speedup over the multi-lane algorithm with a single process per GPU.

The bottom middle and right figures compare the performance of all algorithms, including where the ring algorithm is used in the inter-node stage of the multi-lane all-reduce, comparing 1 versus 16 processes per GPU. The multi-lane algorithm with 16 processes per GPU achieves up to 2.4x speedup over the standard `MPI_Allreduce` for large data sizes.

The top figures show the performance of a standard all-reduce with increasing message size (left) and nodes (right). At 32 nodes, using 2 processes per GPU, we see speedups of up to 1.17x while 4 processes per GPU yields 1.09x speedup.

Conference'17, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

4 Conclusions

We propose to extensively evaluate the performance characteristics of our multi-process approaches and also extend them to collectives such as the neighborhood all-to-all used in sparse linear systems.

References

[1] Traff, J.L., Hunold, S.: Decomposing mpi collectives for exploiting multi-lane communication. 2020 IEEE International Conference on Cluster Computing (CLUSTER) (2020). https://doi.org/10.1109/cluster49012.2020.00037