Center for Understandable, Performant Exascale Communication

University of New Mexico
University of Alabama
University of Tennessee at Chattanooga

Nicholas Bacon: Evaluating the Viability of LogGP for Modeling MPI Performance with Non-contiguous Datatypes on Modern Architectures

Evelyn Namugwanya (UTC): Collective-Optimized FFTs

Carson Woods (UTC): Modeling and Benchmarking Irregular MPI Communication

Center for Understandable Performant Exascale Communication Systems

THE UNIVERSITY OF NEW MEXICO

Evaluating the Viability of LogGP for Modeling MPI Performance with Non-contiguous Datatypes on Modern Architectures

Nicholas Bacon and Patrick Bridges | Department of Computer Science

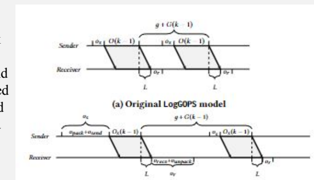
Introduction

Modern architectures and communication systems software include complex hardware, communication abstractions, and optimizations that make their performance difficult to measure, model, and understand. The communication abstractions such as MPI's derived datatypes are a core component of modern high-performance computing (HPC) communication systems. These abstractions are designed to ease programmability and allow the communication system to efficiently send, receive, and compute on (e.g., reduce) complex data structures. Unfortunately, even highly-optimized versions of these abstractions have wildly-varying performances when using realistic application data structures on modern GPU-based systems. In our initial tests, even highly-optimized datatype engines such as MPICH/Yaksa and TEMPI often performed significantly (5%-50%) worse than simple application data packing kernels when working on realistic application data layouts. Importantly, we have not found any case where datatypes outperformed simple application packing kernels when doing GPU to GPU communication.

We modified versions of the existing Netgauge communication performance measurement tool and LogGOPS performance model to accurately characterize the communication behavior of modern hardware, MPI abstractions, and implementations. This includes analyzing their ability to model both GPU-aware communication in different MPI implementations and quantifying the performance characteristics of different approaches to non-contiguous data communication on modern GPU systems. We apply these techniques to quantify the performance of different implementations and optimization approaches to non-contiguous data communication on a variety of systems, demonstrating that modern communication system design approaches can result in widely varying and difficult-to-predict performance variation, even within the same hardware/communication software combination.

Modling changes

Figure 1a shows a simple example of sending two back-to-back *k*- byte messages between a Sender and Receiver. In networks that allow communication-computation overlap, the network and the CPU can progress independently. The *G* and *g* terms are used to determine the network time required for a send and the *O* and *os* terms are used to determine the processor time required for a send. The time required to complete a send operation is the maximum of the network time and the processor time (i.e., the point at which both the network and the processor have completed the work necessary for a send).

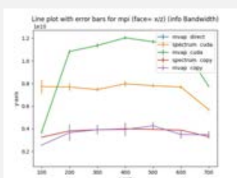


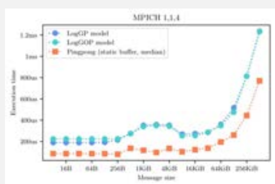
(a) Original LogGOPS model

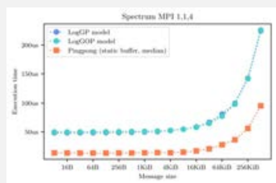
(b) LogGOPS model w/ packing & unpacking message data

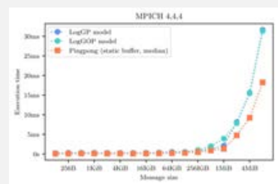
Figure 1b shows a simple example of sending two back-to-back *k*-byte messages using our simple extension of the LogGOPS model. The principal difference between this model and the original LogGOPS model is that, unlike the original model, we explicitly account for the costs associated with moving data between host and device memory and assembling non-contiguous data into Evaluating the Viability of LogGP for Modeling MPI Performance with contiguous message buffers. To capture the impact of these costs, we model the per-message overheads (*os* and *or*) and per-byte overhead (*Os* and *Or*) to include: (i) the time required for sending messages to (*osend*), and receiving messages from (*orecv*), the network; and (ii) the costs associated with preparing non-contiguous data for transmission (*opack*) and the costs associated with processing non-contiguous data after reception (*ounpack*). These costs include datatype packing or unpacking (including launching kernels to pack or unpack data directly in device memory), copying data between host and device memory, creating scatter-gather lists, or other similar per-message or per-byte costs associated with every send

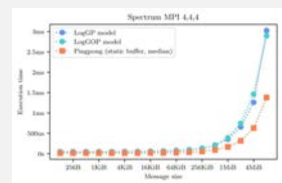
Results











In the original work by Keira Haskins, they explored the time differences between different versions of mpi and a simple hand-packing loop on a 4d data structure. Their results showed (first graph) that it was almost never worth letting mpi handle the packing and unpacking in a real-world application. We used our version of netgauge to extend this work and try to answer the following questions.

1. How effectively do the LogGPS and LogGOPS models quantify communication performance of MPI implementations on modern GPU systems when using simple primitive datatypes?
2. How effectively do the LogGPS and LogGOPS models quantify the performance of communication using MPI derived datatypes?
3. How do the LogGPS and LogGOPS parameters for different MPI implementations change across a range of datatypes and message sizes?

1. The models measured using Netgauge capture some key features of MPI performance, particularly for mid-sized messages. However, they also tend to consistently over-predict ping-pong communication times, particularly for very large and very small messages.
2. In general, this data shows that LogGPS and LogGOPS modeling is more accurate when datatype packing and unpacking costs are high compared to network communication costs. As a result, we conclude that our modified Netgauge-measured LogGPS parameters appear to: (1) accurately model packing and unpacking costs; and (2) continue to systematically overestimate network communication costs similar to the original Netgauge.
3. In the graph above we can see the trends and time change based on the data layout of the MPI vector. In the mvapich case, we can see that going from a space matrix to a continuous vector does not affect timing drastically, but Spectrum has three orders of magnitude slow-down when going to sparse data.

Center for Understandable Performant Exascale Communication Systems

THE UNIVERSITY OF TENNESSEE CHATTANOOGA

Collective-Optimized FFTs

Evelyn Namugwanya[1], Amanda Bienz[2], Derek Schafer[2], Anthony Skjellum[1] | [1]UTC, [2]UNM

Introduction

- HeFFTe is a new FFT library designed for Exascale, dominated by MPI_Alltoallv communication
- Key goal: make MPI_Alltoallv faster so HeFFTe is faster
- Beatnik is a benchmark for global communication based on Pandya and Shkoller's 3D fluid interface "Z-Model" in the Cabana/Cajita mesh framework.
- Beatnik bottlenecked by HeFFTe; it's a good driver app.
- MPI Advance is a collection of MPI extension libraries showcasing new APIs or optimizations of MPI APIs.
- MPI Advance includes faster MPI_Alltoallv variants

Methodology

- We used Tau and Caliper to profile Beatnik, with a specific focus on MPI_Alltoallv.
- We modified the HeFFTe library and replaced the OpenMPI Alltoallv with MPI Advance's Alltoallv.
- We tested six different setups of collective communication:
  - **Non-blocking Alltoallv**: sends all Isends and Irecvs messages and waits for all to complete.
  - **Alltoallv pairwise:** pairwise exchange.
  - **Multi-pair blocking exchange :** combines Non-blocking Alltoallv and Pairwise Alltoallv, uses Waitall.
  - **Multi-pair nonblocking exchange :** Uses Waitany.
  - **Multi-pair test exchange :** Uses Testany.
  - **Alltoallv:** the OpenMPI Alltoallv
- Our goal is to see which setup is fastest in various situations and vs. baseline performance.

Algorithm 1: Pairwise Exchange

Input: p {process id}
n {number of processes}
args {arguments passed to MPI_Alltoallv}

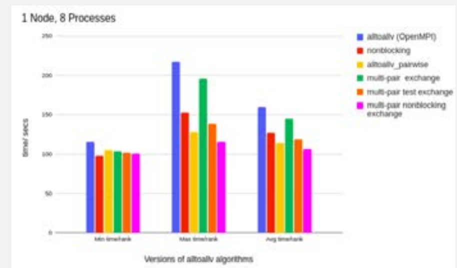
for $i \leftarrow 0$ **to** n **do**
  $p_{send} = p + i \mod n$
  $p_{recv} = p + n - i \mod n$
  Send message to p_{send} and receive message from p_{recv}

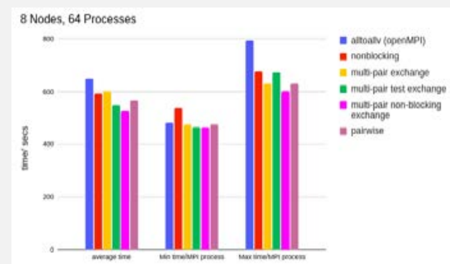
Algorithm 2: Non-Blocking

Input: p {process id}
n {number of processes}
args {arguments passed to MPI_Alltoallv}

for $i \leftarrow 0$ **to** n **do**
  $p_{send} = p + i \mod n$
  $p_{recv} = p + n - i \mod n$
  Initialize non-blocking send to p_{send}
  Initialize non-blocking receive from p_{recv}
Wait for all sends and receives to complete

Results





Conclusions

- While profiling Beatnik, we performed a couple of tests on one through eight nodes, varying the number processes with various versions of MPI Advance's Alltoallv.
- We had seven sets of tests, testing MPI_Alltoallv from standard MPI, Non-blocking Alltoallv, Pairwise Alltoallv and multi-pair blocking exchange, multi-pair non-blocking exchange, multi-pair test exchange from MPI Advance.
- In six of the seven sets performed, MPI Advance's algorithms performed better than the OpenMPI's Alltoallv.
- We also observed that multi-pair non-blocking exchange's performance stands out as compared to other MPI Advance algorithms.

References

1. https://github.com/mpi-advance
2. https://github.com/CUP-ECS/beatnik/
3. https://icl.utk.edu/files/publications/2022/icl-utk-1558-2022.pdf
4. https://hpc.llnl.gov/software/development-environment-software/tau-tuning-and-analysis-utilities
5. https://software.llnl.gov/Caliper/

Center for Understandable Performant Exascale Communication Systems

THE UNIVERSITY OF TENNESSEE CHATTANOOGA

Modeling and Benchmarking Irregular MPI Communication

Carson Woods[1], Derek Schafer[2], Patrick Bridges[2], Anthony Skjellum[1]

1. UTC
2. UNM

Background

- Many applications rely on irregular MPI communication.
- Data irregularity stems from the exchanged data evolving throughout application runtime.
- The exact communication patterns are undetermined at compile time.
- The behavior and performance varies across applications.
- These factors make it challenging to characterize and improve the performance of irregular communication patterns.

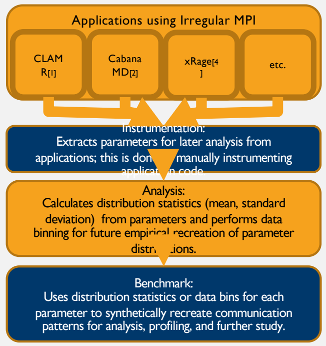


Diagram of general process from instrumenting applications to replicating communication patterns in the synthetic benchmark.

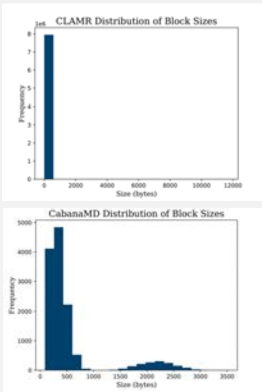
Methods

- Developed a synthetic benchmark that replicates communication patterns of real-world scientific applications.
- The benchmark utilizes parameterized communication data without computational overhead.
- Enables examination and understanding of communication performance in a consistent context.

Extracted Parameters	Meaning
N-owned	Amount of data "owned" by an individual process (in bytes).
N-remote	Amount of data to be sent from one process to another.
Block-Size	The size of the messages to be sent when communicating between processes.
Stride	The number of bytes between blocks.
Communication-Partners	The number of processes that a single process will exchange data with.

Results

- Currently, we are continuing to gather parameterized data by instrumenting various real-world applications.
- The applications being instrumented include CLAMR[1], xRage[4], and Cabana-based proxy-apps like CabanaMD[2].
  - We intend on including more applications in the future.
- Through this process, we extracted and replicated the communication patterns and behavior of these parameters in our benchmark.
- We utilized both empirical and Gaussian distributions to recreate these patterns.
- Initial analysis revealed significant variations in distributions across applications and parameters.
- Our benchmark proved effective at reproducing communication patterns.

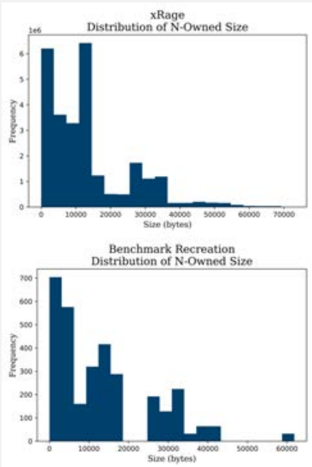


Differing block size parameter communication patterns between CabanaMD and CLAMR mini-apps on Quartz[3]

Conclusions

After implementing an empirical distribution method for our benchmark, we can consistently recreate the communication patterns of an application within our benchmark. Now we can begin to examine the impact that certain communication characteristics have on communication performance.

This work is discussed in greater detail in a paper that was submitted to EuroMPI 2023[5]. It is titled "Quantifying and Modeling Irregular MPI Communication." It is currently pending review.



Comparison of parameter distribution in xRage[4] run vs the benchmark recreation.

References

1. D. Nicholaeff, N. Davis, D. Trujillo, & R. W. Robey (2012). Cell-Based Adaptive Mesh Refinement Implemented with General Purpose Graphics Processing Units.
2. Mniszewski SM, Belak J, Fattebert J-L, et al. Enabling particle applications for exascale computing platforms. The International Journal of High Performance Computing Applications. 2021;35(6):572-597. doi:10.1177/10943420211022829
3. "Quartz." *HPC @ LLNL.* https://hpc.llnl.gov/hardware/compute-platforms/quartz. (Jan. 2023)
4. Grove, John W. 2019. The xRage Hydrodynamic Solver. (7 2019). https: //doi.org/10.2172/1532686
5. Woods et al. Quantifying and Modeling Irregular MPI Communication. Manuscript submitted for publication at EuroMPI 2023

Acknowledgements

This work was performed with partial support from the National Science Foundation under Grants Nos. CCF-2151020, CCF-1918987, CCF-1562306, CCF-1822191, CCF-1821431, OAC-1923980, OAC-1549812, OAC-1925603, and OAC-2201497 and the U.S. Department of Energy's National Nuclear Security Administration (NNSA) under the Predictive Science Academic Alliance Program (PSAAP-III), Award DE-NA0003966.