Evaluating MPI Performance and Scalability in Supercomputing Environments

Morgan Olivia Newton
High-Performance Computing, Code 606.2
NASA Center for Climate Simulation
NASA Goddard Space Flight Center
Summer 2024

Table of contents

01 Introduction

GMAO | GEOS

- The Global Modeling and Assimilation Office (GMAO) conducts modeling and assimilation activities to support NASA's Earth Observation missions.
- The GMAO utilizes data from past missions while aiding in planning for future missions.
- The Goddard Earth Observing System (GEOS) model is key to these activities and consists of multiple components designed for various Earth Science applications.
- The GEOS Earth System Model spans a range of dynamical, physical, chemical, and biological processes.
- This study aims to **investigate MPI performance anomalies** in high-performance computing (HPC) for climate and weather forecasting, with the goal of optimizing these systems for future use.

01 Introduction

GMAO | GEOS

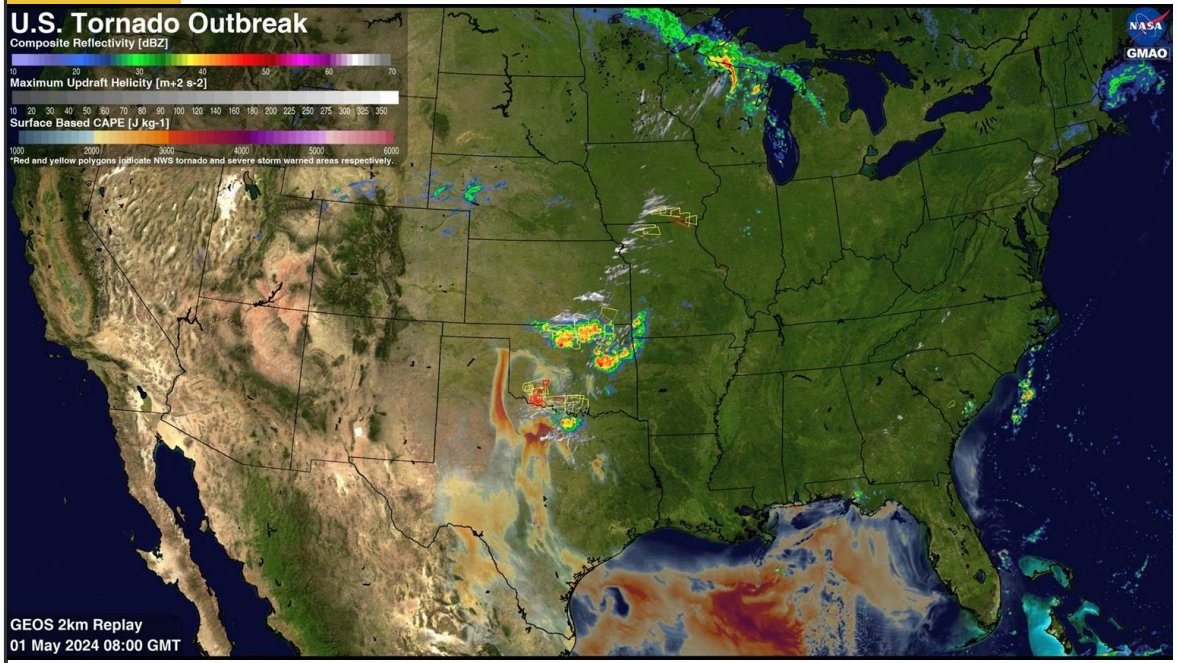


Figure 1: GEOS-Exp 2km Severe Weather Modeling Capabilities Highlighted In 2024 HWT Spring Forecast Experiment

01 Introduction

MPI Performance and Supercomputing Scalability

- MPI (Message Passing Interface) is a standard way for different parts of a program to communicate with each other in a supercomputing environment.
- When running GEOS models, a significant drop in MPI performance is observed once the number of tasks per node exceeds 64 on AMD Milan architecture.
- This performance drop includes significant increases in both latency and elapsed time. OSU Micro-benchmarks were conducted across three different systems to compare MPI performance, specifically focusing on these metrics.
- Scalability within supercomputing is crucial as it allows systems to handle larger tasks and more data efficiently, ensuring faster and more accurate simulations and models.
- By evaluating MPI performance, areas for improvement in communication efficiency can be identified, enhancing **scalability**, optimizing **resource utilization**, and improving overall **system performance**.

02 Systems and Architecture

NASA Center for Climate Simulation (NCCS)

The NASA Center for Climate Simulation (NCCS) provides global HPC resources to scientists and researchers.These resources include the Discover supercomputer, a crucial component of NCCS. Discover features several hardware architectures, most notably AMD Milan nodes, which deliver nearly **8.1 petaflops** of computational power.

NASA Advanced Supercomputing (NAS)

Sponsored by the High-End Computing Capability project, the NASA Advanced Supercomputing (NAS) facility at NASA's Ames Research Center features three supercomputers: Pleiades, Electra, and Aitken. Aitken, NASA's most powerful supercomputer (**10.76 petaflops**), is housed in the Modular Supercomputing Facility at Ames.

02 Systems and Architecture

Microsoft Azure and Amazon Web Services (AWS)

Science Managed Cloud Environment (SMCE) integrates Microsoft Azure and Amazon Web Services (AWS) to offer scalable cloud computing to users. Both HPC clusters and cloud computing provide access to powerful hardware on demand, but they differ in how users access and manage these resources.

02 Systems and Architecture

System Evaluation Overview

- A comparative analysis was performed to provide insights into the optimal use of these diverse computing environments for supercomputing tasks, particularly with GEOS climate code.
- OSU micro-benchmarks were conducted on Discover, AWS, and Aitken using AMD Milan architecture to compare the performance and scalability of all three systems.
- The comparison included examining the operating systems (OS) used in these environments: NCCS cloud systems Azure and AWS run on **Ubuntu**, NAS Aitken operates on **Red Hat Enterprise Linux (RHEL)**, and NCCS Discover uses **SUSE Linux Enterprise Server (SLES)**.
- This analysis aimed to identify any performance variances due to the underlying operating systems.

03 Benchmarking

OSU Micro-benchmarks

- The OSU (Ohio State University) Micro-benchmarks are a suite of tests used to evaluate the performance of MPI. These benchmarks measure latency, bandwidth, multiple-bandwidth, and message rates.
- They are essential tools for identifying performance bottlenecks and comparing different systems, interconnects, and MPI implementations.
- In our study, we used benchmarks **`osu_allgather`**, **`osu_allgatherv`**, **`osu_allreduce`**, **`osu_barrier`**, **`osu_bcast`**, **`osu_gather`**, **`osu_gatherv`**, **`osu_reduce`**, **`osu_reduce_scatter`**, **`osu_scatter`**, and **`osu_scatterv`**.
- MPI **`allgather`** and **`allgatherv`** operations face three scaling limits: small word injection limit (the rate of small message injection), injection bandwidth limit (the maximum data transfer rate), and network bisection limit (the maximum throughput the network can handle when evenly split).
- These limits cause congestion and decreased performance as messages and core counts increase.

03 Benchmarking

MPI Tuning

MPI tuning consists of several configurations designed to **optimize** communication performance by tailoring configuration settings to specific workloads.

I. No Tuning
- Uses default MPI settings without any added adjustments.

II. OFI Tuning
- Involves setting the environment variable I_MPI_FABRICS=ofi.
- Utilizes OpenFabrics Interfaces for efficient inter-node communication.

III. GMAO + OFI Tuning
- Uses I_MPI_FABRICS=ofi and incorporates additional environment variables to enhance performance developed by the GMAO.

IV. GMAO + SHM + OFI Tuning
- Uses I_MPI_FABRICS=shm:ofi to combine shared memory (shm) and OFI for communication.

Parameters

All systems used constant parameters with minor deviations based on version and architecture availability.

Three sets of runs were conducted and analyzed with varying message sizes (Note: AWS had a 96 tasks per node restriction):

- Tuning evaluations: 8192 message size for all systems
- Message size evaluations: 2048, 4096, 8192 message sizes
- Operating System evaluations: 8192 message size for all systems

Operating System	SLES, Ubuntu, Red Hat
Compiler	GCC 12.3
MPI	MPI 2021.13
Message Size	8192
nodes : tasks per node	50:2, 50:4, 50:8, 50:16, 50:32, 50:46, 50:64, 50:96, 64:96, 50:126, 64:126

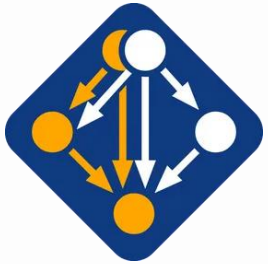
04 Setup and Build

Compilation Process

Compiling across each system involved accessing each environment via SSH, transferring and extracting necessary files, allocating resources, loading GCC and MPI modules, and configuring the build environment with appropriate compilers. Benchmarks were compiled using parallel jobs, and all steps were documented in Confluence to streamline future work.

Comparative System Analysis

All four systems utilized unique hardware and software configurations, leading to slight variations in MPI performance. While Discover and AWS use Slurm for workload management, NAS uses PBS, with Discover employing traditional software modules and Cloud systems relying on the Spack package manager for a more customizable approach.



04 Setup and Build

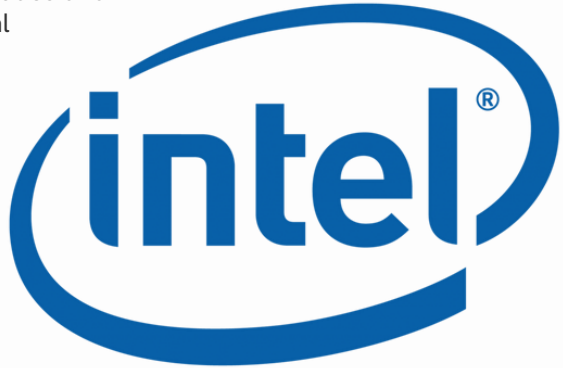
Challenges

NAS

On NAS, **PBS** presented significant challenges requiring conversion of the sbatch command into a qsub command. Several unique flags also needed to be added to correctly convert to the new system. PBS only provided an environment variable for tasks per node and lacks variables for nodes and total tasks. To account for this inconsistency, custom variables were built. These manual adjustments enabled build scripts to function accurately on the new system.

Azure

Azure required a customized Spack environment for the necessary compiler and MPI versions, but using **Intel MPI** caused a system-wide issue where jobs would hang without error information, necessitating escalation to senior engineers and Intel experts. This bug discovery is beneficial but has **paused progress with Azure**.



AWS

Navigating AWS posed challenges, including resource allocation issues and manual increases in usage limits, with tasks per node adjusted to 96 due to system architecture. Segmentation faults during benchmarks were traced to the **'-x' flag** associated with the benchmark. The '-x' flag is used to denote which iteration to start gathering data from, typically set to 1 to ensure that no warm-up iterations are skipped. The root cause of the problem remains unclear, but removing the '-x' flag from the script resolves the issue. This adjustment may affect performance measurement accuracy by omitting startup costs.

05 Results

Scalability Findings

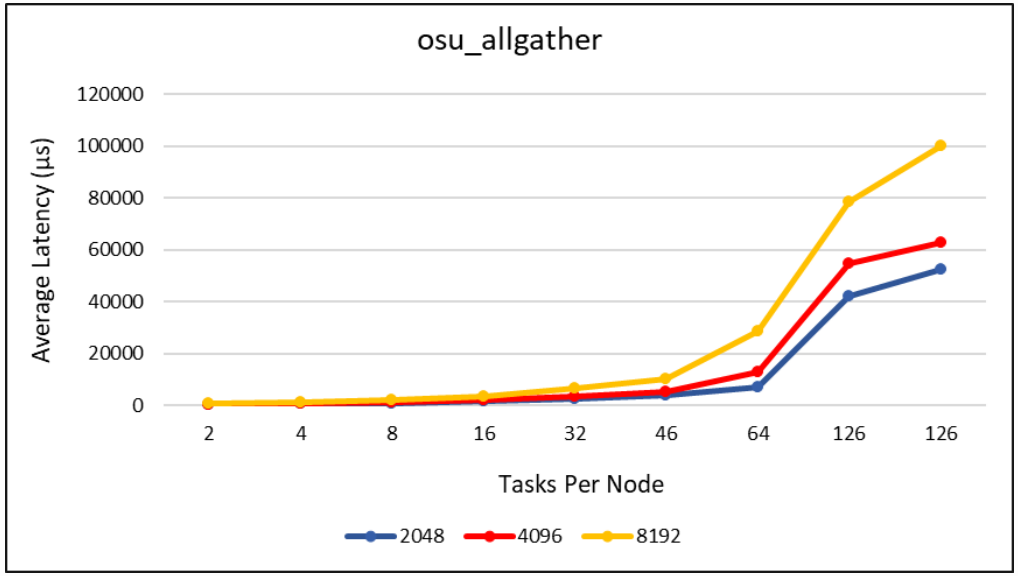


Figure 2: OSU Allgather Benchmark: Average latency increases significantly across all message sizes after 64 tasks per node on Discover Milan architecture for message sizes 2048, 4096, and 8192.

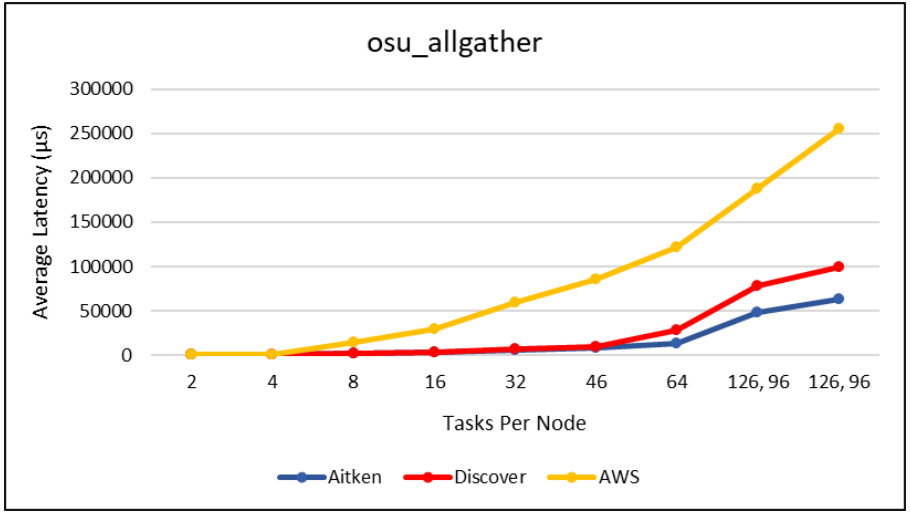
Curiously...

2048 and 4096 show potential leveling off in average latency at around 60,000 µs, suggesting that the impact on latency **may stabilize at higher ranks**.

06 Conclusions

Operating system evaluation of Discover (SLES), AWS (Ubuntu), and Aitken (Red Hat) showed **no significant performance differences** in increasing average latency.

A consistent average latency increase across systems with more than 64 tasks per node confirms issues are **not due to operating system anomalies**.

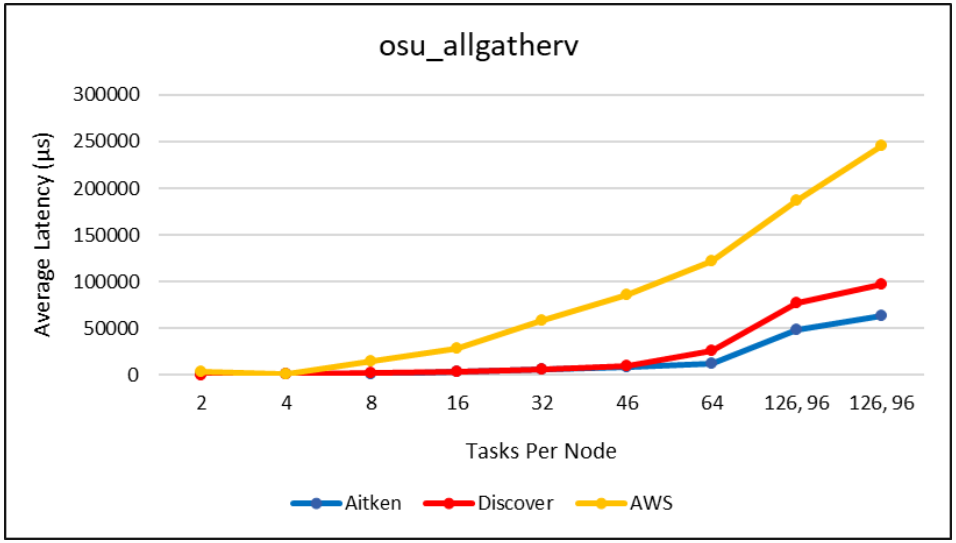


Despite latency increases, Aitken demonstrated better scalability and lower average latency in allgather and allgatherv benchmarks. Aitken performed **twice as well** as Discover with higher configurations (126 tasks per node).

06 Conclusions

AWS showed a concerning increase in average latency as tasks per node increased.

We expected better performance due to **omitted startup iterations** and **lower task limits** (96 tasks per node), yet the results indicate otherwise.



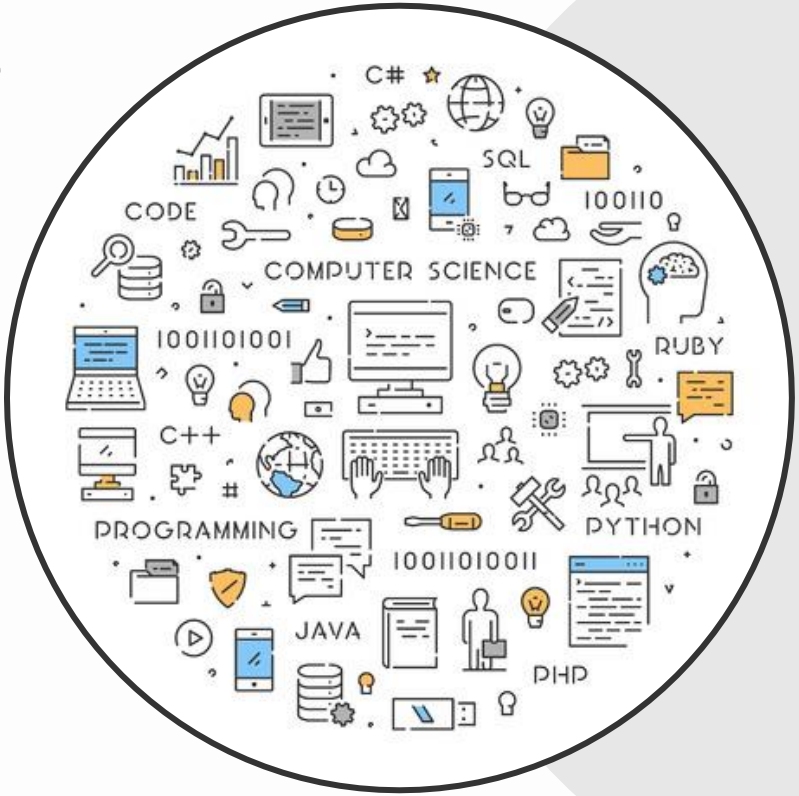
Important Note!

AWS's use of **Ethernet** may adversely affect MPI performance compared to **InfiniBand** on Discover and Aitken.

06 Conclusions

Why does this research matter?

- The implications of this work are crucial for supercomputing and directly benefit the **GMAO** and **NCCS**.
- By confirming that **operating system differences do not significantly impact performance**, the research eliminates one potential variable in optimizing supercomputing environments.
- This research also helps software developers refocus their efforts on their MPI approach.
- Additionally, the validated methodology for assessing the impact of operating systems can be applied to **future studies and evaluations**, fostering more efficient and effective research.



References

NASA Center for Climate Simulation (NCCS). Retrieved from https://www.nccs.nasa.gov.

NASA Scientific and Mission Computing Environment (SMCE). Retrieved from https://www.smce.nasa.gov.

NASA Advanced Supercomputing (NAS) Division. Retrieved from https://www.nas.nasa.gov.

Acknowledgments

Laura Carriere

Bruce Pfaff

William Woodford

Nicko Acks

Hoot Thompson

Matt Thompson

Let's connect!

If anyone has any questions, please contact me!

morgan.o.newton@nasa.gov
morganoliviaevans@gmail.com