Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

ARTICLE TYPE

Evaluating Integration and Performance of Containerized Climate Applications on a HPE Cray System†

Subil Abraham[1] | Ryan Prout[1] | Thomas Robinson*[2] | Chris Blanton[2] | Luis Sal-Bey[2] | Matthew Davis[1]

[1]Oak Ridge National Laboratory, Tennessee, USA
[2]SAIC/NOAA/OAR/Geophysical Fluid Dynamics Laboratory, New Jersey, USA

Correspondence
*Thomas Robinson Email: thomas.robinson@noaa.gov

Present Address
1 Bethel Valley Rd, Oak Ridge, TN 37830

Summary

Containers have taken over large swaths of cloud computing as the most convenient way of packaging and deploying applications. The features that containers offer for packaging and deploying applications translate to High Performance Computing (HPC) as well. At The National Oceanic and Atmospheric Administration (NOAA), containers provide an easy way to build and distribute complex HPC applications, allowing faster collaboration, portability, and experiment computer environment reproducibility amongst the scientific community. The challenge arises when applications rely on Message Passing Interface (MPI). This necessitates investigation into how to properly run these applications with their own unique requirements and produce performance on par with native runs. We investigate the MPI performance for benchmarks and containerized climate models for various containers covering selection of compiler and MPI library combinations from the Cray provided Programming Environments on the Cray XC supercomputer GAEA. Performance from the benchmarks and the climate models shows that for the most part containerized applications perform on par with the natively built applications when the system optimized Cray MPICH libraries are bound into the container, and the hybrid model containers have poor performance in comparison. We also describe several challenges and our solutions in running these containers, particularly challenges with heterogeneous jobs for the containerized model runs.

KEYWORDS:
containers, climate, benchmark, cray

1 | INTRODUCTION

With the advent of containers in the form of LXC[1], they have been continuously considered and evaluated[2] for use in High Performance Computing (HPC) environments. The creation of Docker brought containers into the mainstream and further increased its demand in various applications. Containers found great use in cloud deployments as an easy way to package applications and create reproducible builds in a clean and easily distributable form. The advantages provided by the ease of packaging and reproducible builds cannot be understated, especially in HPC. Projects like Spack[3] were created to address this gap in easily building and packaging applications of libraries in HPC and have found great success, seeing wide use in many HPC centers. However, ease of distribution and reproducibility was still a standing problem that needs to be addressed, especially for providing standalone environments for individual users. The use of container technologies like Singularity aim to fill that gap in HPC, by offering

†This manuscript has been authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The publisher acknowledges the US government license to provide public access under the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

the ability to create a single Singularity Image Format (SIF) file that can be passed around from developer to user in a center, or whose build recipe can be adapted for other centers with different systems.

The National Oceanic and Atmospheric Administration (NOAA) develops and runs many environmental computer models that run for various environmental systems at many different time scales. NOAA's Geophysical Fluid Dynamics Laboratory (GFDL) develops weather and climate models which require a tremendous amount of computing resources and must adhere to strict scientific reproducibility guidelines. NOAA's supercomputer GAEA [4], a HPE Cray XC40 system was purpose built to develop and run these models at scale to serve the needs of climate prediction. However, these models built natively rely on the specific software stack available on the system itself, so reproducing these models effectively at other centers can become a challenge, especially when reproducibility depends on old versions of compilers and dependencies such as netCDF. Containers solve this problem by being able to package the software environment needed for the model in a reproducible artifact that can then be distributed to researchers and users who do not have access to the GAEA system, and potentially be distributed to users on other systems. Users can build and run their own containers without the struggle of trying to exactly match the bespoke software environment that the models rely on.

However, using these containerized environments to run these models can be a challenge as these models as well as many other HPC applications rely on MPI for parallel computation. Evaluating the compatibility of MPI between the container and the host, as well as performance of containers with MPI under different models becomes a necessity. It is also important to look at how well the stated Application Binary Interface (ABI) compatibility between different MPI vendors work in practice for our containerized applications when we bind the host MPI libraries into the container where the application was built with a different MPI library. The fully coupled climate models rely on a resource manager's ability to run heterogeneous jobs or heterogeneous job steps with a shared MPI_COMM_WORLD and have as such been tailored to work with GAEA's Slurm resource manager's ability to run heterogeneous jobs. In Slurm for example, a heterogenous job (or heterogenous job steps) effectively starts two separate jobs with their own job IDs at the same time (or two separate job steps with separate job step IDs within the same job). These two separate entities are separate enough that they can run their own separate codes but share the MPI_COMM_WORLD to allow any communication that is needed between those codes. Some atmospheric models like the Earth System Model 4 (ESM4) take advantage of this "separate but connected" feature. The use of heterogeneous jobs is not something that has been evaluated in previous work on studying containers for HPC. We will be looking at the issues that arise when trying to run such containerized models in heterogeneous jobs alongside studying the MPI compatibility and performance of the models with containers for heterogeneous jobs. Understanding and evaluating these issues would provide great benefit to the GAEA user base who can take prebuilt containerized models with the confidence of knowing the level of performance they will be getting out of these container images.

In Section 2 we cover some background on using MPI with containers and the different models available for that. In Section 3 we look at past work that examine this area of container performance under different conditions on HPC systems and identify the gap in evaluating containerized climate models at scale and containerized heterogeneous jobs. In Section 4 we describe the climate models we use in this study. In Section 5 we describe the containers and host environments, the experiments we performed with benchmarks and climate models, and describe the results. In Section 6 we discuss the challenges in getting to native like performance for the containers and discoveries made for the getting heterogeneous jobs working for the containerized climate models. In Section 7 we discuss potential future paths we could explore as we move forward in standardizing the use and distribution containers for GAEA and other systems.

2 | BACKGROUND

Containers are a lightweight virtualization method with the core idea of creating a clean operating system environment without the need for an independent kernel. The earliest implementation was LXC [1]. Today, projects looking to use containers have access to several different container platforms. Utilizing containers for HPC comes with limitations in terms of finding suitable container platforms that serve the specific niche of HPC [5].

Our goal is to evaluate the portability and performance of containers when used to package certain climate model applications from NOAA. Building and managing these applications, especially when they are needed and heavily used by multiple parties in different locations, is a tricky problem. The idea of having an upstream repository of containers that package these applications that can simply run on a wide array of systems is a very appealing prospect. The issue is the diversity of HPC systems across different centers and under different vendors and the dependency on certain libraries provided by the targeted HPC system, like those underlying MPI. We analyze how to build and run containerized MPI applications while looking to achieve performance and portability.

Container builds follow one of two models to fulfill the MPI library dependency; either applying the hybrid or bind model for including MPI [6]. The hybrid model refers to the method of building and running the MPI-enabled container in a way that the MPI inside the container will work with the MPI outside the container. Doing this successfully requires a couple of things. First, it is required that the MPI inside the container is compatible with the MPI on the targeted system and can communicate across the container border using the Process Management Interface (PMI). Second, if performance is critical, it is up to the container developer to build the optimal MPI implementation for the hardware being targeted

if the application is supposed to be used with the hybrid model. The bind model revolves around the idea of integrating the host provided MPI directly into the container. There is just the single MPI implementation used: the host's implementation. The application developer has to make sure that this host's MPI libraries are made available during the container build process so that the application being containerized will make use of the host optimized libraries. Alternatively, the developer can build the container elsewhere using an independent MPI installation in the container, but making sure that the MPI the developer builds with is ABI compatible with the eventual host's MPI. We use this second method of building containers for our tests for the bind model. The bind model gives you a clearer path to performance, since you can use the optimized host MPI, but it makes portability more difficult. In our case, we are testing Cray-MPICH and Intel-MPI from the host, and GNU-MPICH and Intel-MPI in the containers. This means we are able to take advantage of some work from the MPICH ABI compatibility initiative [7]. We can predetermine what is ABI compatible, build the container remotely, and enable the ability to grab the system MPI at runtime. In the rest of this paper we look at the results of doing this in practice, with real applications from NOAA, on a production Cray system.

3 | PRIOR WORK

Many prior work have evaluated the overhead i.e. the additional computing power used by containers on top of the application it runs, produced by containers on different platforms, including on HPC systems. Morabito [8] identified that they have negligible overhead even in extremely resource constrained systems like Raspberry Pis, making containers suitable for a wide variety of large and small systems. Full virtualization is unsuitable for HPC because of its incredible overhead, but containers almost entirely eliminate that overhead while still providing a consistent repeatable environment that can be passed around to others. Xavier et al. [9] and Beserra et al. [2] compare the performance of earlier container runtimes like LXC [1] and hypervisor based virtualization on HPC systems to identify that containers have negligible overhead and much better performance for IO than virtualization. Similarly, Torrez et al. [10] have performed a thorough study at HPC scale with 512 nodes to see how more modern HPC focused container solutions like Charliecloud, Shifter, and Singularity produce almost zero performance overhead on the system and had very little performance difference between them, giving freedom to HPC centers to provide whichever container framework is most suitable for its users. This provides a consensus to the findings of several others like Rudyy et al. [11] who studied the performance overhead and scaling of a containerized multiphysics biological simulation of a pulsatile artery with Docker, Shifter, and Singularity and demonstrated Shifter and Singularity's comparable superiority in HPC environments; with Younge et al. [5] who conduct a performance study of Singularity on a Cray XC system at Sandia and find little performance difference from native for MPI benchmarks when using the native Cray MPI libraries and compared performance with a deployment on Amazon EC2; and Le et al. [12] who did MPI performance comparisons using NEURON, OSU, and Intel MPI benchmarks between native and Singularity and found little difference in performance at the San Diego Supercomputing Center. Younge et al. [5] also test the dynamic linking of vendor MPI libraries into containers and its effect on performance, which we also investigate as part of our work. Ruiz et al. [13] discovered potential bottlenecks for containers used with network bound applications due to the default network settings, which could have implications for MPI applications in HPC. Abraham et al. [14] studied the impact of container usage on the Lustre storage system, and the impact of the storage system on the container application performance, and found that Singularity due to its single file structure of its image has the least impact on the MDSs and the OSSs, which further motivates our use of Singularity on GAEA which relies on Lustre.

4 | CLIMATE MODELS

Two of GFDL's models have been selected for use in this study. The first is the Aquaplanet model based on the physics of the GFDL Atmosphere Model 4 (AM4) [15 16]. The Aquaplanet is an idealized full physics atmosphere model that runs without any land or ocean variation. Instead, the surface is an "ice model" that is smooth and has a constant surface forcing. This model is simpler than the full AM4, runs faster, and ensures that we are only testing the atmosphere model so that other model components are not contributing to any load imbalance. This model can be used for studying large scale atmospheric phenomenon such as the Walker Circulation [17] and global tropical cyclone frequency [18].

The second model is the fully coupled GFDL Earth System Model 4.1 (ESM4) [19]. This model runs with an updated AM4 atmosphere model with full chemistry and the GFDL LM4P model coupled with the Modular Ocean Model (MOM6) and the Sea Ice Simulator version 2 (SIS2) ice model. The AM4 and LM4P run on one set of MPI ranks while the MOM6 and SIS2 runs on a different set of MPI ranks. Two separate communicators are set up for each of the sets of ranks referred to in the model as a "pelist". A coupler is used so that the ocean/ice and the atmosphere/land pelists can run concurrently. The atmosphere/land runs three time steps for every one ocean/ice time step. Information is exchanged about fluxes, wind velocity, and other environmental factors that influence the scientific calculations between the different model components using the MPI_COMM_WORLD communicator at the end of the slower time step. This concurrent running of model components utilizing separate communicators that the ESM4 runs on Gaea is being referred to as a heterogeneous run in this paper.

Table 1 Container images and the package versions used

Environment	Packages
GNU MPICH container (hpcme_gnu_rhel8.sif)	RHEL 8.4 GCC 8.4.1 MPICH 3.3.2 Netcdf 4.8.0
Intel MPI container (ubuntu-intel_2022.1.1.sif)	Ubuntu 18.04 Intel 2021.5 Intel MPI 2019.10 GCC 7.5.0 Netcdf 4.8.0
Intel 2021 OneAPI container (intel2021.2_netcdfc4.7.4_ubuntu.sif)	Ubuntu 18.04 Intel 2021.2 Intel MPI 2021.2 GCC 7.5.0 Netcdf 4.7.4
Host Environment	Intel 19.0.5 (PrgEnv-intel) GCC 8.3.0 (PrgEnv-gnu) Cray MPICH 7.7.11 Intel MPI 2019.5

Both models are run for a month of simulation time from January 1-31. The month run test is used by GFDL to calculate the simulated years per day (SYPD): the number of simulated years that the model can run in an actual day on the computer. The times reported are the main loop times and ignores the initialization and finalization. The main loop is where the actual simulation occurs in the model, and the one month benchmark has been successfully used in the past to calculate the SYPD.

5 | EVALUATION AND RESULTS

We ran our experiments, both small and large scale, on Gaea: a HPE Cray XC 40 system located in the Oak Ridge National Laboratory and operated for NOAA by the Department of Energy. For production, Gaea's C4 cluster consists of 2656 nodes, with each node running 2 18-core Intel Xeon Broadwell CPUs and 64 GB of memory, with a peak performance of 3.52 petaflops, and makes use of Cray's Aries/Dragonfly interconnect. For our small scale runs we used T4, a 20 node test and development system, with each node running 2 16-core Intel Xeon Haswell CPUs and also with 64 GB of memory. Both clusters have access to the center-wide DDN Lustre scratch file system with 38 petabytes of space.

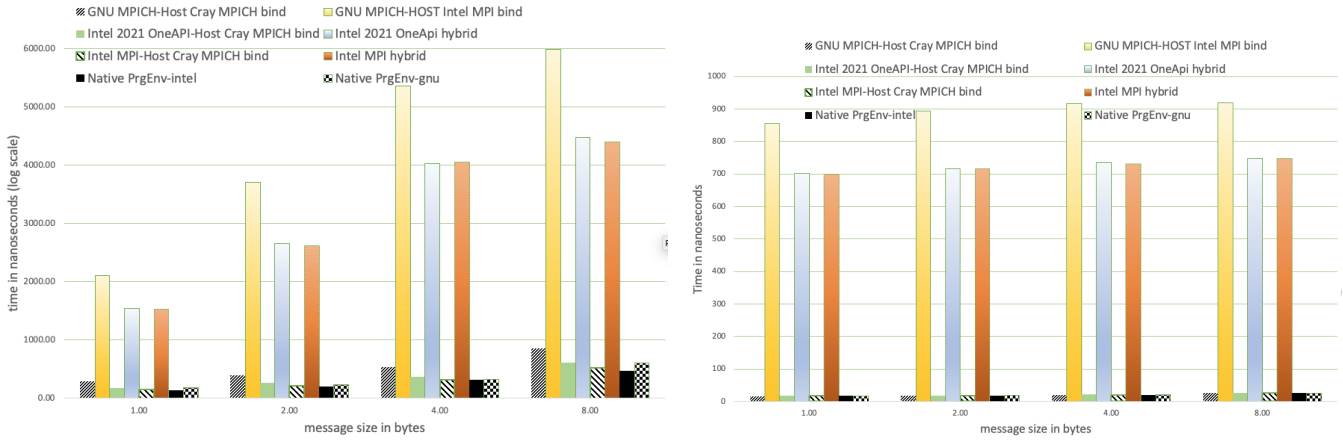
For our evaluations of the MPI based programs, we executed runs on bare metal built with the Intel and the GCC compilers targeting the Cray MPICH installation on Gaea, as well as using three container images with different software environments. The programs compiled for bare metal only target the Cray MPICH installation as that is the default MPI variant users on Gaea compile their code for, and this provides a baseline against which we can compare the containers. Separate runs were performed for hybrid and bind MPI models for each container to get the full scope of performance. Table 1 describes the relevant software packages on the host and in the container images, for the benchmark and climate model applications we used. We note that the climate model experiments primarily used the Intel compiler and are tested with Intel MPI and Cray MPICH, and the ESM4 model and Aquaplanet model with GNU compilers did not work with the hybrid model as it was incompatible with GNU MPICH (these models were tested and developed with Intel MPI and Cray MPICH, and fixing incompatibilities with GNU MPICH was out of scope for our work). We also note that the programs compiled in a container necessarily need to target MPI libraries that are freely available to use like GNU MPICH or Intel MPI, as Cray MPICH is proprietary and not available outside of official installations like the one on Gaea. This is why it is essential to evaluate the performance these containers with hybrid and bind model runs to see if they are compatible with the Cray MPICH on the host system, and also compare the performance of hybrid and bind models with the bare metal runs that directly use Cray MPICH.

To avoid a long description each time when referring to a container and identifying whether it is running under the hybrid or bind model and identifying which host MPI it is using in case of the bind model, we will describe some short names we will use in the rest of the paper. Containers run with the bind model will be named in the form "ContainerName-Host HostMPIName bind" where ContainerName is one of the containers described in Table 1 and HostMPIName will refer to either Cray MPICH or Intel MPI which are installed on the host also described in Table 1 e.g.

GNU MPICH-Host Cray MPICH bind. Containers run with the hybrid model will be named in the form "ContainerName hybrid" e.g. Intel 2021 OneAPI hybrid.

5.1 | OSU Benchmark runs

As an initial performance evaluation to see how well the containers performed under heavy MPI communication workloads, we ran the osu_allgather benchmark from the OSU Micro Benchmark suite [20]. We ran separately compiled benchmarks for the Host Environment as well as separately compiled in each container, so that we are not mixing binaries built for different environments, host or container. For each container, we ran the benchmark using both the hybrid and the bind MPI model. For the GNU MPICH container, we ran the bind model tests by binding the Cray MPICH (ABI compatible) libraries built for GCC, as well as binding the non-Cray Intel MPI libraries just to see how far the ABI compatibility could stretch and see how their performance differed. For the Intel containers, we only bind mounted the Cray MPICH libraries built for Intel for the bind model runs. We ran into errors we couldn't resolve when trying to bind mount the non-Cray Intel MPI libraries into the two Intel containers. So we chose to focus on bind mounting the Cray MPICH libraries with the Intel containers. We ran each of these on the small scale on the T4 cluster, with each run on 16 nodes running 32 ranks per node. The results of the small scale runs were taken as the median of 5 repetitions. The large scale runs on C4 used 450 nodes with 32 ranks per node but due to time and resource limits on the Gaea supercomputer (experiments at this size disrupt the primary climate modelling work being performed on Gaea) we were only able to do 1 repetition.



(a) Large scale - 450 nodes, 32 tasks per node

(b) Small scale - 16 nodes, 32 tasks per node

Figure 1 Performance of osu_allgather for message sizes up to 8 bytes

With the small scale runs and also in the large scale runs shown in Figure 1a and 1b, we can already see the clear difference between hybrid model runs and bind model runs that use the Host Cray MPICH in their time to complete the osu_allgather benchmark. There is an order of magnitude difference between the Host Cray MPICH bind containers and the other containers (the hybrid containers and the GNU MPICH-Host Intel MPI bind container) on allgather performance for message sizes up to 8 bytes. The Host Cray MPICH bind containers are close to or on par with the native runs in performance for the small message sizes, with the GNU MPICH-Host Cray MPICH bind container being the slowest among them. Figure 1a and Figure 1b shows this difference for the small and large scale runs, without the GNU MPICH hybrid container. The GNU MPICH hybrid container in the large scale run performs the worst and jumps several orders of magnitude worse starting at 8 bytes. Figure 2 shows the comparison for large scale runs up to message size of 8 bytes on log scale in order to display the GNU MPICH hybrid performance (without log scale, the GNU MPICH hybrid performance bar would be the only one visible in the figure). Further analyses of these benchmarks will not consider the GNU MPICH hybrid container. So from Figures 1a, 1b, and 2 we can see that for small message sizes up to 8 bytes, all the hybrid model containers as well as the GNU MPICH-Host Intel MPI Bind container show very poor performance compared to the Cray MPICH bind containers.

Another interesting thing to note about the large scale runs is that the GNU MPICH container-Host Intel MPI bind run, the Intel 2021 OneAPI container hybrid run, and the Intel MPI hybrid run display these spikes in their time to complete at the 16 and 64 byte message sizes during the large scale run. Figure 3 shows the graph where the spikes are visible in comparison with the other runs. This could be a quirk of the GAEA networking or the PMI communication or something to do with how the MPI parameters on GAEA are tuned when using the Intel MPI libraries (either in hybrid or bind mode). These spikes are not observed for the Cray MPICH bind containers which use the Cray MPICH libraries from the host.

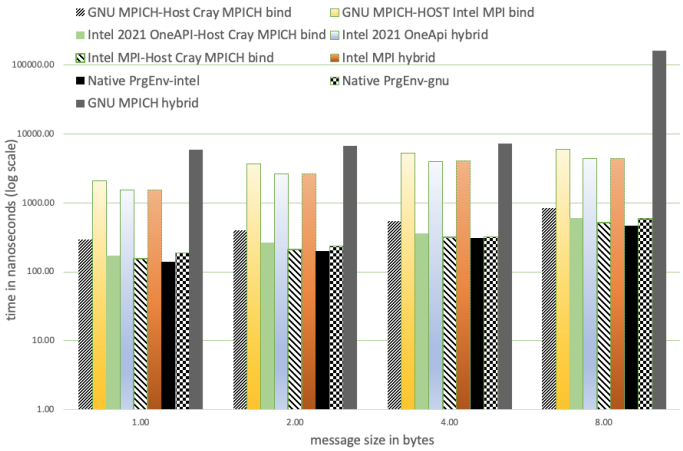


Figure 2 Performance of large scale run of osu_allgather (450 nodes, 32 tasks per node) for small message sizes in log scale

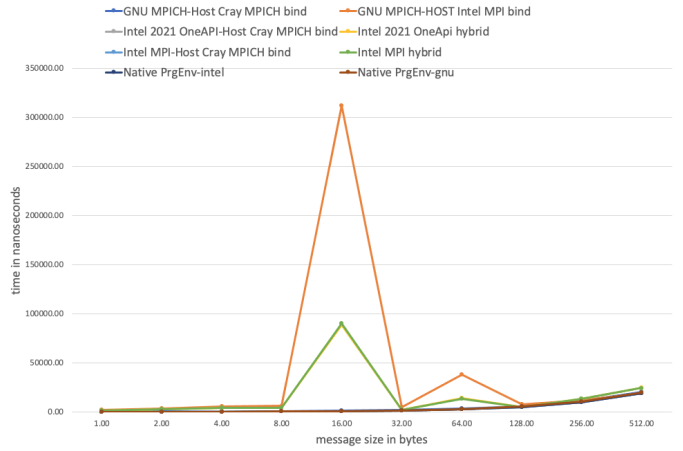


Figure 3 Performance of large scale runs up to a message size of 512 bytes. Spikes observed at 16 and 64 byte message size

And these spikes occurred when the experiment was repeated to verify this behavior (these repetitions aren't represented in the figures) so we could rule out transient issues on the system itself. Further investigation is required to make sure that actual containerized applications that require those message sizes aren't affected, determine the cause of the spikes, and how best to tune the parameters to eliminate those spikes. We choose to defer this further investigation to future work to focus on evaluating the containers with the default parameters as that is more informative to the majority of Gaea users running these models that don't tune their parameters.

For the large scale runs, the GNU MPICH hybrid run and the GNU MPICH-Host Intel MPI bind run could only complete up to 512 bytes message size before the job timed out. There seems to be some setup time overhead for the GNU MPICH-Intel MPI bind run that is not being measured in the benchmark itself because the actual measured performance gets closer to the other bind model runs at 256 and 512 byte message size. Table 2 shows the the performance from 1 to 512 byte message sizes comparing the GNU MPICH-Host Intel MPI bind container and the Intel 2021 OneAPI-Host Cray MPICH bind container. The strangely high time to complete for the GNU MPICH-Host Intel MPI bind container at 16 and 64 bytes you see in the table are the spikes we discussed earlier in Figure 3.

In the large scale runs, there are a couple of instances at 8192 and 16584 bytes message size where the Intel container hybrid runs seem to perform better than the native or the Intel container bind runs. Figure 4 shows this. This behavior reoccurred when the experiment was repeated to verify this behavior (these repetitions aren't represented in the figures). Further investigation could net useful information on why this might be happening. But from an overall perspective, it is clear that the Intel containers with the Cray MPICH bind model are the closest to native performance, and indicates to us that we should target that set up for containerizing applications to run on GAEA.

Table 2 GNU MPICH-HOST Intel MPI bind vs Intel 2021 OneAPI-Host Cray MPICH bind upto 512 byte message size for osu_allgather for the large scale run (450 nodes, 32 tasks per node)

Message Size	GNU MPICH-Host Intel MPI bind (μs)	Intel 2021 OneAPI-Host Cray MPICH bind (μs)
1.00	2106.26	172.13
2.00	3705.07	266.33
4.00	5357.16	363.31
8.00	5982.68	612.30
16.00	311939.36	1022.08
32.00	5132.17	1663.66
64.00	38318.42	2983.23
128.00	7422.58	5457.56
256.00	11731.34	10497.98
512.00	18970.68	19832.21

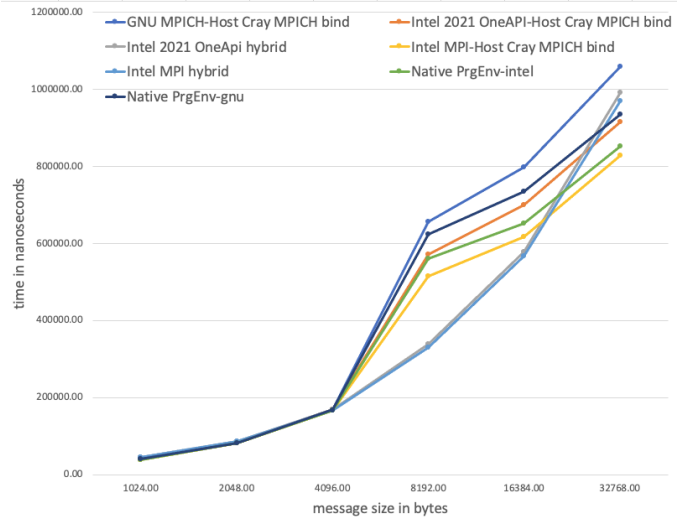


Figure 4 Performance of large scale runs for large message sizes. Hybrid runs of the Intel containers seem to perform better in a couple of instances.

5.2 | Aquaplanet model runs

The Aquaplanet model was scaled using an increasing number of MPI ranks and run on bare metal, as well as run with the hybrid and bind models with Singularity. The container and executable for both hybrid and bind are identical. The Aquaplanet models were compiled using the Intel OneAPI 2021.2 compilers and Gfortran/GCC 8.4 for the Intel and GNU container runs, respectively. The container builds use a newer version of NetCDF than bare metal.

The timings for the month runs of the Aquaplanet with Intel are found in Table 3. The hybrid model runs much slower than the bare metal runs. As the number of cores increases, the percent difference increases from 12% for 216 cores to 102% for 1728 cores. For smaller and short runs, such as for development purposes, the hybrid model would be a viable choice, but for production runs at scale, the hybrid method is not a reasonable methodology.

The container bind model timings are similar to the bare metal runs. The difference between the bare metal and container MPI bind model timings 1% - 7%) is within the difference in timings experienced on a run-to-run basis by the bare metal runs on the machine (up to 14%). Compared to multiple bare metal runs (not shown), the bind model timings are faster in some cases. While there is a steeper learning curve and a great deal of system work to be done in order to figure out what needs to be bound into the container to get the bind model to work, the bind model is a viable use case for production runs.

Table 3 Number of cores (column 1) with main loop timings (in seconds) from one month runs of the aquaplanet model using bare metal, and the Singularity MPI-Hybrid and Bind configurations in the intel2021.2_netcdfc4.7.4_ubuntu.sif container.

Total Cores	Bare Metal (s)	Hybrid (s)	Bind (s)
216	1533.07	1745.50	1587.56
384	1194.56	1467.48	1217.80
432	847.57	1131.60	850.40
576	632.96	946.75	635.71
864	467.62	722.46	464.47
1152	378.64	775.05	392.15
1728	283.91	572.44	302.46

Table 4 Number of cores (column 1) with main loop timings (in seconds) from a one month run of the aquaplanet model using bare metal, and the Singularity MPI-Bind configurations in the hpcme_gnu_rhel8.sif container.

Cores	Bare Metal (s)	Bind (s)
216	2026.35	1951.29
384	1597.43	1532.81
432	1108.32	1103.46
864	877.39	601.98

Similar results are observed in Table 4 when the Aquaplanet was run with the bind model using the GNU container. The timings are all within the natural variation of run-to-run differences, except for the run with 864 cores where the container was about 30% faster. The bare metal run appears to be losing its scaling, but the container seems to maintain the scalability. This could be because of the time to search and load library files is faster for the container run than the bare metal run. The bare metal run might have to search and load libraries from disparate locations on the Lustre file system, whereas for the container all the data including the library files are in squashed into one Singularity Image Format (SIF) file that the container runs from. So there is no need to go far to get the library files which could provide speed up at larger scales. This result would become important for a higher-resolution model that might be run on many thousands of cores, such as a weather model. It may be possible to get better scaling performance using a container based on this result.

5.3 | ESM4 runs

The ESM4 model programs were compiled in the same manner as the Aquaplanet model but with just the Intel 2021 OneAPI container (and with the PrgEnv-Intel module for the bare metal run). Then the standard one-month timing test suite was run on both the host and container environments. Table 5 shows the test experiment configurations; the coupled ESM4 model runs the atmosphere/land and ocean/ice on separate sets of MPI ranks, and OpenMP threads are used for the atmosphere and not the ocean model component. These configurations are chosen so that the atmosphere and ocean components run in a similar amount of time.

Challenges were overcome in order to run the coupled ESM4 model within the container environment, including binding needed library directories, setting specific environment variables, and including the submitting Gaea job environment in the batch environments. Additionally, only the bind model was used, as the container's Intel MPI which we would need to use for the hybrid model, is currently incompatible with heterogeneous jobs. These are described more in the Discussion section.

At least two runs each of the bare metal and the Intel 2021 OneAPI container were done for the timing experiments, and the results are shown in Table 5. The bind model run times were between 0% and 4% different from the bare metal runs. The bind model run times are consistent with the variation observed with the Aquaplanet experiments and are within the run to run variability of running on Gaea.

While there is more work to understand and optimize which system resources and directories need to be bound in the container, it is encouraging that when using the bind model with containers, the performance impact of containerization on real-world MPI applications such as the ESM4 model is low.

Table 5 ESM4 experiment configurations (nodes, atmosphere MPI ranks and OpenMP threads, and ocean MPI ranks) with main loop timings (in seconds) from a one month run of using Native and MPI-Bind configurations in the intel2021.2_netcdfc4.7.4_ubuntu.sif container. (Timings are an average of 2-3 runs. Ocean model does not use OpenMP.)

Nodes	Atmos ranks x Threads	Ocean ranks	Native (s)	Container (s)
30	216 x 2	648	2898.5	2899.6
36	216 x 2	842	2904.5	2971.4
51	432 x 4	951	1675.0	1732.4
54	432 x 4	1057	1720.7	1734.5
60	432 x 4	1296	1685.3	1732.9
61	432 x 4	1300	1656.1	1714.9
69	576 x 4	1300	1366.0	1417.9
85	864 x 4	1300	1057.8	1047.0
89	864 x 4	1441	1018.8	1016.0
93	864 x 4	1608	975.8	1010.6
94	1152 x 4	1057	1072.5	1065.0
105	864 x 4	2044	971.5	1010.7

6 | DISCUSSION

A useful feature of containerization is the ability to simply package up the application and the required dependencies, including any MPI libraries, in a single container image to pass on to other users. However, from our evaluation there is a clear benefit in being able to bind the optimized host Cray MPICH libraries into the container, provided the host MPI libraries are ABI compatible with the container's MPI libraries. This is useful to know for any given supercomputer so that the appropriate paths can be mounted into the container. However, the process of binding these paths is manual where the appropriate MPI module, that is ABI compatible with the MPI library in the container, has to be loaded if it even exists on the system, and then add the appropriate paths pointing to the `libmpi.so` location as mounts in the container with the `-bind` flag. This can be a process of trial and error when it comes to Cray MPICH installations because in addition to the `libmpi.so`, one also needs to make sure that all the supporting libraries that `libmpi.so` depends on are also mounted in the container. This includes Cray specific libraries like alps, ugni, udreg, xpemem which provide host specific optimizations for MPI communications. Part of the reason for the slowness of the hybrid model runs may come from the container's internal MPI libraries not having access to these optimizations. There is no standard documentation for mounting these for container runs so figuring it out took some time and effort. Being able to automatically discover these libraries and mount them would be a useful feature in our system and also useful if the containers we build are passed to other users elsewhere who just want to be able to run it with maximum performance without needing to go through the excess effort in finding all the host libraries they need. e4s-cl[21] is a software package was built for this, where it is able to automatically discover the MPI libraries and creates a profile of the libraries to mount into the container. A container run using e4s-cl will automatically bind the discovered libraries that are saved in the specified profile into the container. However, this is currently still a work in progress and the automatic discovery is limited since it requires the use of mpicc as part of the discovery process, which GAEA does not have.

The need to bind mount the host's MPI libraries into the container in order to get significantly better performance does break some portability. It is not so simple as just taking a container image and running. It is possible with future improvements to Cray's PMI could bring performance of the hybrid model runs of containers closer to that of the bind model and native runs, which would make usage of containers on HPC so much more seamless and save users the hassle of needing to manage ABI compatible libraries for binding.

The ESM4 model's reliance on heterogeneous jobs exposed issues in both host and container runtime environments. Slurm allows both heterogeneous job allocations and heterogeneous job steps within a slurm allocation. Running jobs with the host MPI library resulted in hangs on `MPI_Init` with heterogeneous allocations, but ran successfully with heterogeneous job steps within an allocation. Intel's MPI library was incapable of receiving the heterogeneous task distribution, resulting in incorrect creation and access attempts of shared memory resources. The MPICH library in the GNU MPICH container ran successfully only if the job allocation was made with the `hetjob` flag, and hung on `MPI_Init` otherwise. Updating Slurm on a similar test system has resolved the host MPI library hang on `MPI_Init` while the other libraries maintain the same behavior. This has made the linking of the host MPI library into the container's executable important for both performance and basic functionality.

When running performance tests for containers with Cray MPI libraries, it was initially found that the container with the GNU toolchain did not benefit from the addition of the library path to `LD_LIBRARY_PATH`. This was due to the container's `mpicc` compiler wrapper setting `RPATH` for

the container's MPI library. Multiple workarounds were successfully tested including compilation with `-Wl,-enable-new-dtags` to instead set `RUNPATH`, setting `LD_PRELOAD` for the Cray MPI library, and launching the executable in the container by invoking the dynamic linker directly with the `-inhibit-rpath` flag. Each method was found to have its drawbacks. For the first, the user of the container may not have the means to rebuild the application. For the second, loading the library through the environment variable applies to a wider scope than just the executable we wish to run. In the last case, explicitly excluding search for MPI libraries in `RPATH` when a matching host library cannot be found can result in several different outcomes. Since version 2.30, the GNU dynamic linker offers a `-preload` flag to preload a library only for the specified executable. When available within the container this option offers the ability to reliably load host libraries in the proper scope.

The Slurm scheduler (used on GAEA and other HPC sites) allows users to specify which environment variables are passed from the submitting environment (login/head node) to the batch environment (compute nodes). While the default is to pass all environment variables through `-export=ALL`, the most common use on Gaea is to pass no environment variables through `-export=NONE`, which then populates the batch job environment using the system default shell initialization. This helps ensure a clean batch environment for reproducible experiments. However, we discovered that using `-export=NONE` caused heterogeneous jobs (simple tests as well as ESM4) to hang immediately after launching. More work is needed to discover which environment variables in the login user environment are needed for heterogeneous jobs.

Containerizing the climate models can be time consuming, but should not be detrimental to the overall development of climate model code. The most time consuming part of dealing with climate model containers is compiling packages and dependencies that are required, such as HDF5 and NetCDF. There is currently an effort to cut down this overhead to make containers even easier to build. Also, providing containers that can be used with the Intel compiler poses unique licensing challenges because there is not clarity as to whether we can share containers that have the compilers necessary to compile the model. Once these hurdles are overcome, there were only slight modifications to the GFDL workflow that were needed to compile and run the model. The portability and shareability of the container make it an attractive choice for the future of climate modeling.

7 | FUTURE WORK

There are some further things we could evaluate as we move forward in getting containers ready for regular use on GAEA. Testing how well the Lustre filesystem handles reads from the container images for very large scale deployments is of importance in the exascale era especially if the SIF files are large and aren't properly striped. Automatically discovering and bind mounting the right libraries from the host into the container is another useful area to look at. e4s-cl[21] has done some work in this regard but currently relies on the presence of mpicc and mpirun, which does not work on a Cray system if it doesn't have those executables and may not correctly pull in the necessary Cray libraries. Exploring faster builds for model software dependencies packages is planned to be done through a partnership with E4S utilizing spack caches and large base containers. Using the containers to run on other HPC platforms and the cloud is a planned goal for NOAA. The next-generation models will likely be shared with containers instead of just a repository on GitHub.

8 | CONCLUSION

In this paper, we explored the containerization of MPI based applications and climate models on the GAEA supercomputer then evaluated their performance under different conditions. We identified that binding the host's MPI libraries into the container nets the best performance when done correctly, as well as identifying some odd behavior during benchmark runs for some hybrid model containers. We explored the challenges and figured out solutions for containerizing and running applications that specifically rely on the ability to run heterogeneous jobs. We articulate steps needed to effectively run those applications at native performance without needing to modify the applications themselves to suit the more conventional non-heterogeneous job blueprint. We discuss the future benefits of containerization for distributing climate models and engendering collaboration in the climate science community.

ACKNOWLEDGMENT

This research used resources of the National Climate-Computing Research Center which is located within the National Center for Computational Sciences at the Oak Ridge National Laboratory and supported under a Strategic Partnership Project between the DOE and NOAA. This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.

References

1. LXC. https://linuxcontainers.org/lxc/. Accessed: November 25 2019.
2. Beserra D, Moreno ED, Endo PT, Barreto J, Sadok D, Fernandes S. Performance analysis of LXC for HPC environments. In: IEEE. ; 2015: 358–363.
3. Gamblin T, LeGendre M, Collette MR, et al. The Spack package manager: bringing order to HPC software chaos. In: ; 2015: 1–12.
4. GAEA Supercomputer. https://www.noaa.gov/organization/information-technology/gaea. Accessed: April 14 2022.
5. Younge AJ, Pedretti K, Grant RE, Brightwell R. A tale of two systems: Using containers to deploy hpc applications on supercomputers and clouds. In: ; 2017.
6. Singularity and MPI. https://sylabs.io/guides/3.5/user-guide/mpi.html. Accessed: April 14 2022.
7. MPICH ABI Compatibility Initiative. https://wiki.mpich.org/mpich/index.php/ABI_Compatibility_Initiative. Accessed: April 14 2022.
8. Morabito R. A performance evaluation of container technologies on internet of things devices. In: IEEE. ; 2016: 999–1000.
9. Xavier MG, Neves MV, Rossi FD, Ferreto TC, Lange T, De Rose CA. Performance evaluation of container-based virtualization for high performance computing environments. In: IEEE. ; 2013.
10. Torrez A, Randles T, Priedhorsky R. HPC Container Runtimes have Minimal or No Performance Impact. In: ; 2019: 37-42.
11. Rudyy O, Garcia-Gasulla M, Mantovani F, Santiago A, Sirvent R, Vázquez M. Containers in HPC: A Scalability and Portability Study in Production Biological Simulations. In: ; 2019.
12. Le E, Paz D. Performance Analysis of Applications using Singularity Container on SDSC Comet. In: ; 2017; New Orleans, LA, USA.
13. Ruiz C, Jeanvoine E, Nussbaum L. Performance Evaluation of Containers for HPC. In: ; 2015.
14. Abraham S, Paul AK, Khan RIS, Butt AR. On the use of containers in high performance computing environments. In: IEEE. ; 2020: 284–293.
15. Zhao M, Golaz JC, Held I, et al. The GFDL global atmosphere and land model AM4. 0/LM4. 0: 1. Simulation characteristics with prescribed SSTs. *Journal of Advances in Modeling Earth Systems* 2018; 10(3): 691–734.
16. Zhao M, Golaz JC, Held I, et al. The GFDL global atmosphere and land model AM4. 0/LM4. 0: 2. Model description, sensitivity studies, and tuning strategies. *Journal of Advances in Modeling Earth Systems* 2018; 10(3): 735–769.
17. Silvers LG, Robinson T. Clouds and Radiation in a Mock-Walker Circulation. *Journal of Advances in Modeling Earth Systems* 2021; 13(2). doi: 10.1029/2020ms002196
18. Burnett AC, Sheshadri A, Silvers LG, Robinson T. Tropical Cyclone Frequency Under Varying SSTs in Aquaplanet Simulations. *Geophysical Research Letters* 2021; 48(5). doi: 10.1029/2020gl091980
19. Dunne J, Horowitz L, Adcroft A, et al. The GFDL Earth System Model version 4.1 (GFDL-ESM 4.1): Overall coupled model description and simulation characteristics. *Journal of Advances in Modeling Earth Systems* 2020; 12(11): e2019MS002015.
20. OSU Micro Benchmarks. https://mvapich.cse.ohio-state.edu/benchmarks/. Accessed: March 28 2022.
21. E4S-CL. https://github.com/E4S-Project/e4s-cl. Accessed: April 14 2022.

How to cite this article: