Performance portability of an intermediate-complexity atmospheric research model in coarray Fortran

Extended Abstract

Damian Rouson[1], Ethan D Gutmann[2], Alessandro Fanfarillo[2], Brian Friesen[3]

[1]Sourcery Institute, USA
[2]National Center for Atmospheric Research, USA
[3]Lawrence Berkeley National Laboratory, USA

ABSTRACT

We examine the scalability and performance of an open-source, coarray Fortran (CAF) mini-application (mini-app) that implements the parallel, numerical algorithms that dominate the execution of The Intermediate Complexity Atmospheric Research (ICAR) [4] model developed at the the National Center for Atmospheric Research (NCAR). The Fortran 2008 mini-app includes one Fortran 2008 implementation of a collective subroutine defined in the Committee Draft of the upcoming Fortran 2018 standard. The ability of CAF to run atop various communication layers and the increasing CAF compiler availability facilitated evaluating several compilers, runtime libraries and hardware platforms. Results are presented for the GNU and Cray compilers, each of which offers different parallel runtime libraries employing one or more communication layers, including MPI, OpenSHMEM, and proprietary alternatives. We study performance on multi- and many-core processors in distributed memory. The results show promising scaling across a range of hardware, compiler, and runtime choices on up to ~100,000 cores.

CCS CONCEPTS

• **Software and its engineering → Parallel programming languages**; • **Applied computing** → *Environmental sciences*;

KEYWORDS

coarray Fortran, computational hydrometeorology

ACM Reference Format:

Damian Rouson[1], Ethan D Gutmann[2], Alessandro Fanfarillo[2], Brian Friesen[3]

[1]Sourcery Institute, USA [2]National Center for Atmospheric Research, USA [3]Lawrence Berkeley National Laboratory, USA . 2017.

Performance portability of an intermediate-complexity atmospheric research model in coarray Fortran. In *Proceedings of PAW17: Second Annual PGAS Applications Workshop, Denver, CO, USA, November 12–17, 2017 (PAW17),* 4 pages.

https://doi.org/10.1145/3144779.3169110

PAW17, November 12–17, 2017, Denver, CO, USA

ACM ISBN 978-1-4503-5123-2/17/11…$15.00
https://doi.org/10.1145/3144779.3169110

1 INTRODUCTION

1.1 Motivation and Background

Since the Fortran 2008 standard was published in 2010 [5], a Fortran program executes as a fixed number of instances that the standard refers to as "images." Each image has its own state, including data and input/output streams, corresponding to a single-program, multiple-data (SPMD) programming style. Images execute asynchronously except at program launch, termination, and program-specified synchronization points. Images may define so-called coarray data structures to provide one-sided access to data by other images in a partitioned global address space (PGAS).

A few published studies have provided encouraging assessments of using Fortran's PGAS features in research applications running at scale [3, 6, 8]. One of the most attractive characteristics of these features is the ability to express parallel algorithms in a single language without embedding compiler directives or referencing communication layers that are not part of the language standard (e.g., no MPI or OpenMP). In theory, this should delay choice of communication layers to link-time. Ideally, an application developer writes a standard, Fortran-only application, compiles it into object code, and then links to one or more communication layers.

Such flexibility poses the computing equivalent of what food author Michael Pollan termed "The Omnivore's Dilemma:" if I belong to a species that owes some of its evolutionary advantage to being able to eat a wide variety of foods, what food is best for me to eat? [7] Similarly, if I can express my parallel algorithm once using CAF and then consume cycles atop a multitude of software stacks and hardware platforms, where best to consume cycles and using which supporting software stack? Here we report the results of an initial study of several current options for compiling, linking, and executing a mini-app designed to be representative of the parallel numerical algorithms and physics employed in ICAR.

ICAR simulates the motion of the atmosphere at kilometer length scales and produces flow patterns with a fidelity that is attractive to the hydrology community for studying surface water. Figure 1 depicts results from ICAR simulations over North America.

1.2 Objectives

This paper evaluates alternatives for compiling, linking, and executing one mini-app CAF source code using the following technologies:

- MPI [1] and OpenSHMEM [10] communication layers,
- GNU Compiler Collection (GCC) and Cray compilers, and
- Multi- and many-core processors.

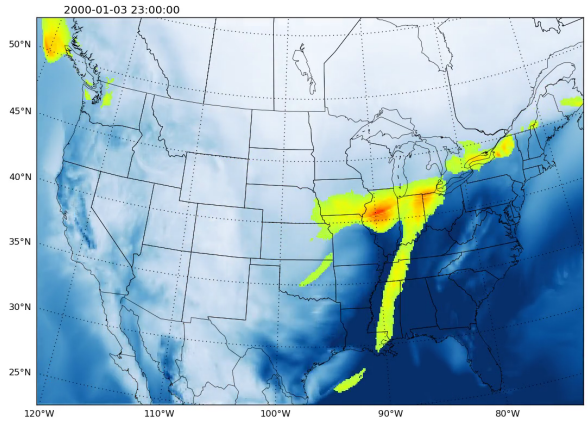

Figure 1: A visualization of the atmospheric distribution of water vapor (blues) and the resulting precipitation (green to red) simulated by ICAR.

We also tested the Intel Fortran compiler versions 16 and 17. The resulting executable program crashes on Cray platforms and hangs at launch on when run on more than 16 cores on the SGI platform.

We show performance for two one-sided data access patterns:

(1) Gets: an image retrieves data from memory managed by another image without the providing image's involvement, and
(2) Puts: an image stores data in memory managed by another image without the receiving image's involvement.

We also explore the performance and scalability of multi- versus many-core processors.

2 METHODOLOGY

2.1 Physics and numerics

The coarray-ICAR implemention employed here uses the Thompson Eidhammer microphysics parameterization [11] and the first-order MP-DATA advection algorithm [9]. The microphysics parameterization requires 11 primary input variables (e.g. air pressure, water vapor, cloud water), 9 of which are modified by the parameterization and advected throughout the domain requiring communication between processes.

The test case described here uses an idealized hill: a sine curve representing a 1000 m high mountain in the middle of the domain. The air is initially near 100% relative humidity with a background wind of approximately 10 m s^{-1}.

At every time step, the outer edges of the local domain are processed first, then passed to their neighbors while the interior of the domain is processed. This is tested using both a CAF "put" operation, in which one-sided communication is initialized asynchronously to send data to a neighbor, and a CAF "get" operation, in which one-sided blocking communication retrieves data from a neighbor.

2.2 Compilers, runtimes, and hardware

We performed the experiments presented here on two supercomputers at the National Energy Research Scientific Computing Center (NERSC), located at Lawrence Berkeley National Laboratoory (LBNL), and one supercomputer at NCAR. The LBNL systems are

- Edison: a Cray XC30 featuring 5586 nodes with two sockets of 12-core Intel Xeon Processor E5-2695 v2 ("Ivy Bridge"), running at 2.4 GHz.
- Cori: a Cray XC40 with 12 076 compute nodes, spanning two architectures; 2388 nodes each have two sockets of 16-core Intel Xeon Processor E5-2698 v3 ("Haswell") operating at 2.3 GHz; the other 9688 nodes are single-socket, 68-core Intel Xeon Phi Processor 7250 ("Knights Landing") at 1.4 GHz.

Edison and Cori have the same "Aries" interconnect with a dragonfly topology. All Cori measurements on the Xeon Phi nodes ran in the "quadrant" NUMA configuration with the MCDRAM configured as a transparent cache to the DDR4 memory.

The NCAR system is "Cheyenne," an SGI ICE XA Cluster with 4032 compute nodes, each containing two sockets of 18-core Intel Xeon Processor E5-2697 v4 ("Broadwell"), running at 2.3 GHz. Cheyenne uses a Mellanox EDR Infiniband interconnect with a partial 9D Enhanced Hypercube single-plane topology. We compiled coarray-ICAR on the NERSC systems using the Cray Fortran compiler version 8.6.0. We compiled at NCAR using the GCC version 6.3 Fortran front end, which uses the OpenCoarrays application binary interface (ABI) [2] to support CAF. We tested two parallel runtime libraries that implement the OpenCoarrays ABI:

(1) The default MPI library using the SGI MPT MPI,
(2) The recently released OpenSHMEM library.

We distributed CAF images as follows:

- 68 images per Xeon Phi node on Cori,
- 24 images per Xeon node on Edison, and
- 36 images per node on Cheyenne,

corresponding to one image per physical core in each case.

3 DISCUSSION OF RESULTS

Figure 2 shows results with two domain sizes ($500 \times 500 \times 20$ and $2000 \times 2000 \times 20$ grid cells) running on Cheyenne. These tests compare scaling with blocking "get" and non-blocking "put" operations, and compare scaling using MPI or OpenSHMEM. The differences between MPI and OpenSHMEM are minor, with OpenSHMEM performing only slightly better. However, the differences between the "get" and "put" operations are substantial. For the $500 \times 500 \times 20$ domain, "puts" scale well to 1800 cores, then scale poorly to 3600 cores; however, "gets" only scale well to 504 cores, scale poorly to 1224, and no further improvement comes from up to 3600 cores.

MPI "gets" are not reported because a problem with the MPI implementation makes them several orders of magnitude less efficient. Similarly, MPI "put" results are not reported beyond 1800 cores because a problem in the MPI implementation caused memory allocations to become a bottleneck at higher process counts. The $2000 \times 2000 \times 20$ domain exhibits the same pattern but the scaling of "puts" remains strong up to 3600 cores; whereas scaling of "gets" continues, albeit poorly, to 3600 cores.

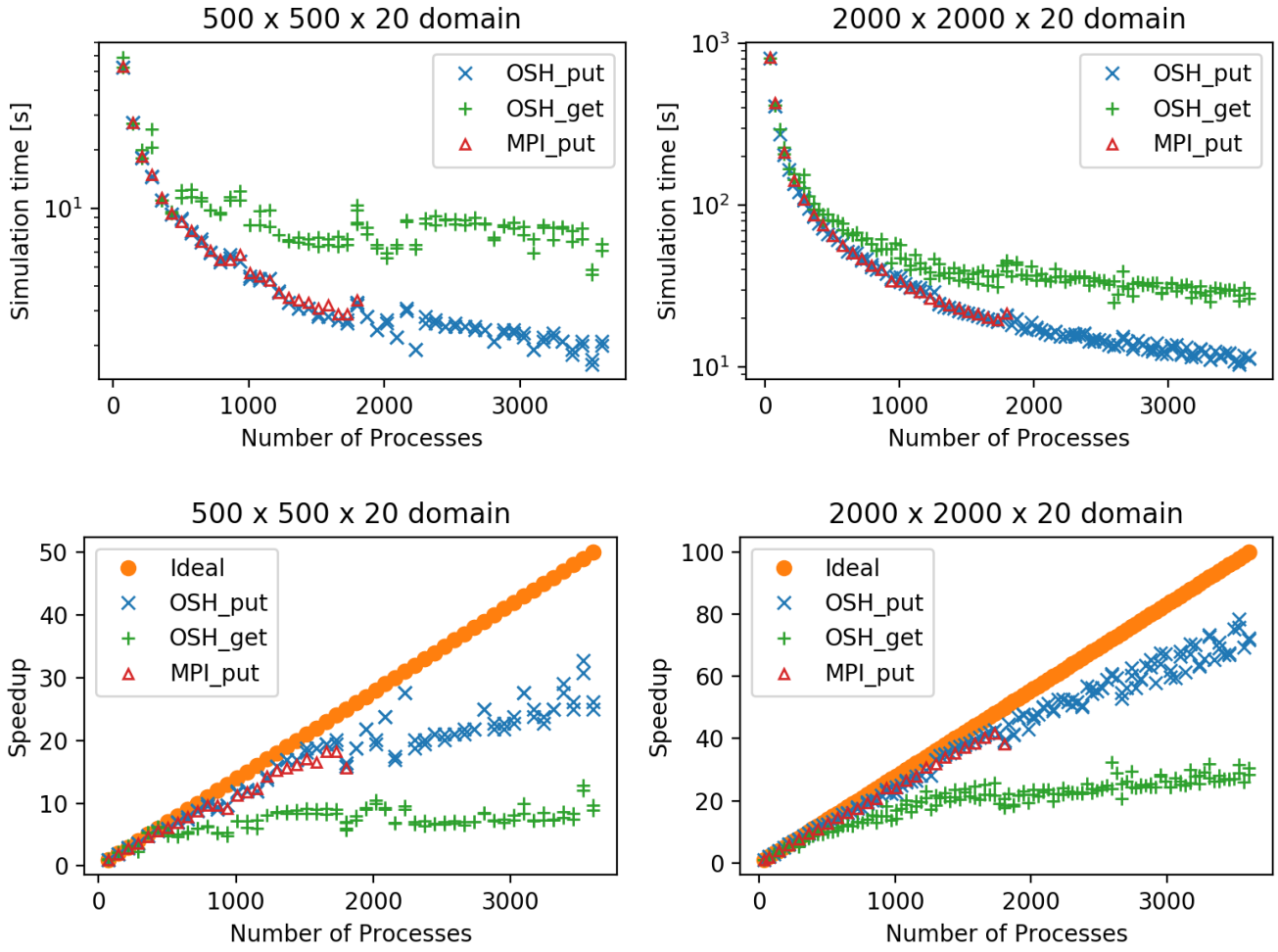

Figure 2: Speedup and timing results for two different domain sizes, two different communications backends (OSH=OpenSHMEM, MPI=MPI) and two different communication methods ("get" and "put") using Coarray ICAR.

Figure 3 (top) depicts results across multiple compilers, machines, and architectures. In this case, only "puts" are tested, and only for the 2000 × 2000 × 20 domain. The GNU compiler was used on Cheyenne (Xeons). The Cray compiler was used on Edison (Xeons) and Cori (KNL). To aid interpretation, the scaling results are reported as a fraction of the ideal scaling for each machine in the top left. These results show that the Cray compiler+system scales better than the gfortran+SGI system, with 75% efficiency at >10k cores, while on Cheyenne, only 55% of ideal was achieved. This also shows that the KNL system scales well out to large core counts (60% of the ideal with 19 200 cores), but that the total runtimes are significantly slower than the equivalent runtimes on Xeons (top right). The KNL performance might be improved in the future by implementing OpenMP threaded parallelism within a node.

To further test the scaling performance in the best case, "puts" on Cray Xeons, a limited set of tests were performed up to nearly 100 000 cores (fig. 3 bottom). This system scaled well to over 10 000 cores, and continued scaling with nearly 50% of the single-node efficiency at 98 304 cores.

4 CONCLUSIONS

We have presented scaling and performance analysis using CAF on different architectures, compilers, and communications backends. These tests have utilized modern Fortran standards and as a result no changes to the code were required for different configurations.

This work demonstrates the capability of one-sided non-blocking communications protocols to readily scale up to very high core counts. It is expected that with moderate performance optimizations and a larger total problem size, this system will be able scale to even higher core counts.

ACKNOWLEDGMENTS

The first author would like to thank the Visitor Programs of the Computational Information Systems Laboratory and the Research

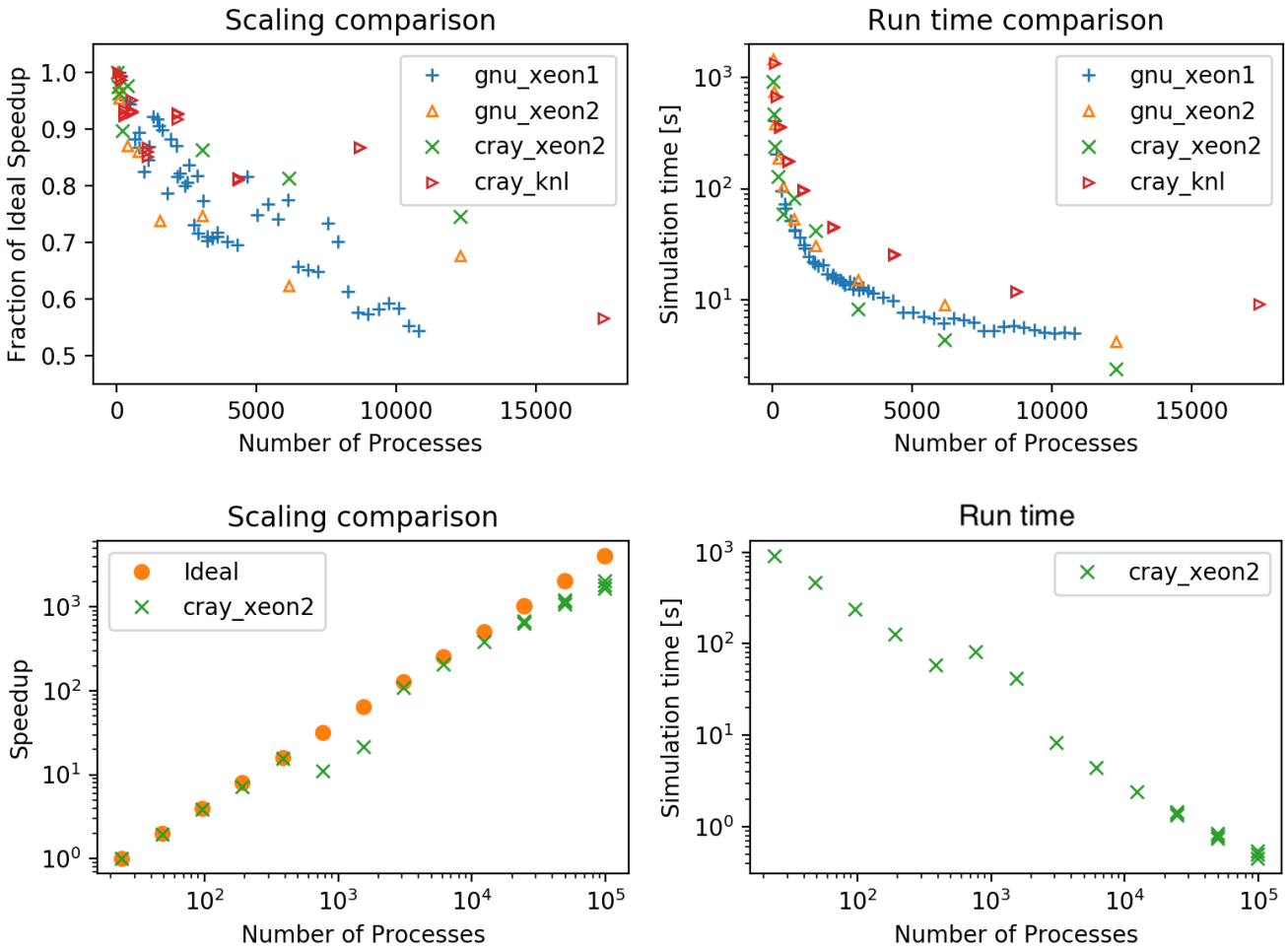

Figure 3: Cross-platform and extreme scaling results using Coarray ICAR.

Applications Laboratory of NCAR for travel support for a visit during which much of the work presented in this paper was performed.

This work was funded in part through a contract from the U.S. Army Corps of Engineers.

REFERENCES

[1] 2016. *MPI: A Message Passing Interfacew Standard.* Standard. University of Kentucky, Knoxville, Tennessee USA.

[2] Alessandro Fanfarillo, Tobias Burnus, Valeria Cardellini, Salvatore Filippone, Dan Nagle, and Damian Rouson. 2014. OpenCoarrays: open-source transport layers supporting coarray Fortran compilers. In *Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models*. ACM, 4.

[3] Sudip Garain, Dinshaw S Balsara, and John Reid. 2015. Comparing Coarray Fortran (CAF) with MPI for several structured mesh PDE applications. *J. Comput. Phys.* 297 (2015), 237–253.

[4] Ethan Gutmann, Idar Barstad, Martyn Clark, Jeffrey Arnold, and Roy Rasmussen. 2016. The intermediate complexity atmospheric research model (ICAR). *Journal of Hydrometeorology* 17, 3 (2016), 957–973.

[5] ISO/IEC 1539-1:2010 2010. *Information technology – Programming languages – Fortran – Part 1: Base language.* Standard. International Organization for Standardization, Geneva, CH.

[6] George Mozdzynski, Mats Hamrud, and Nils Wedi. 2015. A partitioned global address space implementation of the European centre for medium range weather forecasts integrated forecasting system. *The International Journal of High Performance Computing Applications* 29, 3 (2015), 261–273.

[7] Michael Pollan. 2006. *The omnivore's dilemma: A natural history of four meals.* Penguin.

[8] Robert Preissl, Nathan Wichmann, Bill Long, John Shalf, Stephane Ethier, and Alice Koniges. 2011. Multithreaded global address space communication techniques for gyrokinetic fusion applications on ultra-scale platforms. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 78.

[9] P K Smolarkiewicz and L G Margolin. 1998. MPDATA: A finite-difference solver for geophysical flows. *J. Comput. Phys.* 140, 2 (1998), 459–480.

[10] Monika ten Bruggencate, Matthew Baker, Barbara Chapman, Tony Curtis, Eduardo DàĂŹAzevedo, James Dinan, Karl Feind, Manjunath Gorentla Venkata, Jeff Hammond, Oscar Hernandez, David Knaak, Gregory Koenig, Jeff Kuehn, Jens Manser, Tiffany M. Mintz, Nicholas Park, Steve Poole, Wendy Poole, Swaroop Pophale, Michael Raymond, Pavel Shamis, Sameer Shende, Lauren Smith, and Aaron Welch. 2016. OpenSHMEM Application Programming Interface, Version 1.3. (February 2016). http://bit.ly/openshmem-1-3 Accessed on 3 October 2017.

[11] Gregory Thompson and Trude Eidhammer. 2014. A Study of Aerosol Impacts on Clouds and Precipitation Development in a Large Winter Cyclone. *Journal of the Atmospheric Sciences* 71, 10 (Sept. 2014), 3636–3658.