BERKELEY LAB
Bringing Science Solutions to the World

U.S. DEPARTMENT OF ENERGY
Office of Science

Parallel Runtime Interface for Fortran (PRIF)

Dan Bonachea[1], Katherine Rasmussen [1], Brad Richardson[2], Damian Rouson[1]
[1]Computer Languages and Systems Software (CLaSS) Group
National Energy Research Scientific Computing (NERSC) Center[2]

Platform for Advanced Scientific Computing (PASC24), Zurich, Switzerland, 5 June 2024

https://fortran.lbl.gov/

Overview

01

Background:
Coarray Fortran (CAF)

02

Motivations:
CAF and PRIF

03

The Compiler Landscape

04

PRIF Design:
Overview & Status

05

PRIF Implementation:
Caffeine

06

Future Work

Overview

01

Background:
Coarray Fortran

02

Motivations:
CAF and PRIF

03

The Compiler Landscape

04

PRIF Design :
Overview & Status

05

PRIF Implementation:
Caffeine

06

Future Work

Background: Co-Array Fortran (CAF)

Co-Array Fortran for parallel programming

Robert W. Numrich, Silicon Graphics, Inc.

and

John Reid, Rutherford Appleton Laboratory

Abstract

Co-Array Fortran, formerly known as F⁻⁻, is a small extension of Fortran 95 for parallel processing. A Co-Array Fortran program is interpreted as if it were replicated a number of times and all copies were executed asynchronously. Each copy has its own set of data objects and is termed an image. The array syntax of Fortran 95 is extended with additional trailing subscripts in square brackets to give a clear and straightforward representation of any access to data that is spread across images.

References without square brackets are to local data, so code that can run independently is uncluttered. Only where there are square brackets, or where there is a procedure call and the procedure contains square brackets, is communication between images involved.

There are intrinsic procedures to synchronize images, return the number of images, and return the index of the current image.

We introduce the extension; give examples to illustrate how clear, powerful, and flexible it can be; and provide a technical definition.

1 Introduction

We designed Co-Array Fortran to answer the question 'What is the smallest change required to convert Fortran 95 into a robust, efficient parallel language?'. Our answer is a simple syntactic extension to Fortran 95. It looks and feels like Fortran and requires Fortran programmers to learn only a few new rules.

The few new rules are related to two fundamental issues that any parallel programming model must resolve, work distribution and data distribution. Some of the complications encountered with other parallel models, such as HPF (Koebel, Loveman, Schrieber, Steele, and Zosel 1994), CRAFT (Pase, MacDonald, and Meltzer 1994) or OpenMP (1997) result from the intermixing of these two issues. They are different and Co-Array Fortran keeps them separate.

First, consider work distribution. Co-Array Fortran adopts the Single-Program-Multiple-Data (SPMD) programming model. A single program is replicated a fixed number of times, each replication having its own set of data objects. Such a model is new to Fortran, which assumes a single program executing alone with a single set of data objects. Each replication of the program is called an **image.** Each image executes asynchronously and the normal rules of Fortran apply, so the execution path may differ from image to image. The programmer determines the actual path for the image with the help of a unique image index by using

1

Numrich & Reid (1998) "Co-Array Fortran for parallel programming," *ACM SIGPLAN Fortran Forum* 17:2, 1-31.

- Numrich invented CAF at Cray as Fortran 95 extensions
- Numrich & Reid incorporated CAF into Fortran 2008
- CAF = SPMD + PGAS

"The underlying philosophy of our design is to make the smallest number of changes to the language required to obtain a robust and efficient parallel language without requiring the programmer to learn very many new rules."

Reid & Numrich (2007) "Co-arrays in the next Fortran standard," *Scientific Programming*, 15(1), 9-26.

Background: Parallelism in Fortran

Co-Array Fortran for parallel programming

Robert W. Numrich, Silicon Graphics, Inc.

and

John Reid, Rutherford Appleton Laboratory

Abstract

Co-Array Fortran, formerly known as F--, is a small extension of Fortran 95 for parallel processing. A Co-Array Fortran program is interpreted as if it were replicated a number of times and all copies were executed asynchronously. Each copy has its own set of data objects and is termed an image. The array syntax of Fortran 95 is extended with additional trailing subscripts in square brackets to give a clear and straightforward representation of any access to data that is spread across images.

References without square brackets are to local data, so code that can run independently is uncluttered. Only where there are square brackets, or where there is a procedure call and the procedure contains square brackets, is communication between images involved.

There are intrinsic procedures to synchronize images, return the number of images, and return the index of the current image.

We introduce the extension; give examples to illustrate how clear, powerful, and flexible it can be; and provide a technical definition.

1 Introduction

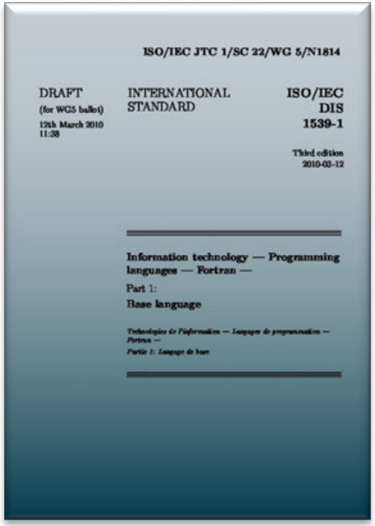
We designed Co-Array Fortran to answer the question "What is the smallest change required to convert Fortran 95 into a robust, efficient parallel language?". Our answer is a simple syntactic extension to Fortran 95. It looks and feels like Fortran and requires Fortran programmers to learn only a few new rules.

The few new rules are related to two fundamental issues that any parallel programming model must resolve, work distribution and data distribution. Some of the complications encountered with other parallel models, such as HPF (Koebel, Loveman, Schreiber, Steele, and Zosel 1994), CRAFT (Pase, MacDonald, and Meltzer 1994) or OpenMP (1997) result from the intermixing of these two issues. They are different and Co-Array Fortran keeps them separate.

First, consider work distribution. Co-Array Fortran adopts the Single-Program-Multiple-Data (SPMD) programming model. A single program is replicated a fixed number of times, each replication having its own set of data objects. Such a model is new to Fortran, which assumes a single program executing alone with a single set of data objects. Each replication of the program is called an **image**. Each image executes asynchronously and the normal rules of Fortran apply, so the execution path may differ from image to image. The programmer determines the actual path for the image with the help of a unique image index by using

1

1998



ISO/IEC JTC 1/SC 22/WG 5/N1814

DRAFT
(for WG5 ballot)
12th March 2010
11:38

INTERNATIONAL STANDARD

ISO/IEC
DIS
1539-1

Third edition
2010-03-12

Information technology — Programming languages — Fortran —

Part 1:

Base language

Technologies de l'information — Langages de programmation — Fortran —

Partie 1: Langage de base

2010

Fortran 2008 (published in 2010):

- Single Program Multiple Data (SPMD): images
- Partitioned Global Address Space (PGAS): coarrays
- Synchronization, locks, critical sections, termination, image enumeration, *some* atomic subroutines/variables.
- `do concurrent`: offload, vectorize, or multithread

"Enable programmers to communicate properties of their code rather than to mandate specific optimizations that exploit those properties"

Dan Nagle (c. 2013)

Background: Parallelism in Fortran

Co-Array Fortran for parallel programming

Robert W. Numrich, Silicon Graphics, Inc.

and

John Reid, Rutherford Appleton Laboratory

Abstract

Co-Array Fortran, formerly known as F--, is a small extension of Fortran 95 for parallel processing. A Co-Array Fortran program is interpreted as if it were replicated a number of times and all copies were executed asynchronously. Each copy has its own set of data objects and is termed an image. The array syntax of Fortran 95 is extended with additional trailing subscripts in square brackets to give a clear and straightforward representation of any access to data that is spread across images.

References without square brackets are to local data, so code that can run independently is uncluttered. Only where there are square brackets, or where there is a procedure call and the procedure contains square brackets, is communication between images involved.

There are intrinsic procedures to synchronize images, return the number of images, and return the index of the current image.

We introduce the extension; give examples to illustrate how clear, powerful, and flexible it can be; and provide a technical definition.

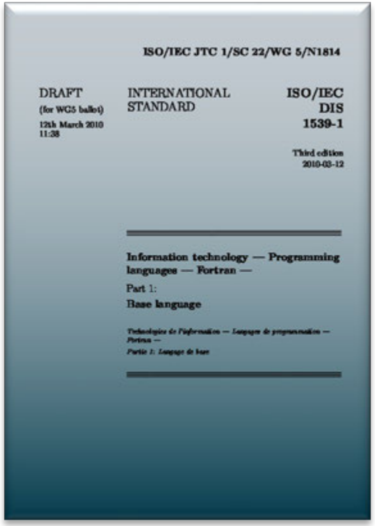
1 Introduction

We designed Co-Array Fortran to answer the question "What is the smallest change required to convert Fortran 95 into a robust, efficient parallel language?". Our answer is a simple syntactic extension to Fortran 95. It looks and feels like Fortran and requires Fortran programmers to learn only a few new rules.

The few new rules are related to two fundamental issues that any parallel programming model must resolve, work distribution and data distribution. Some of the complications encountered with other parallel models, such as HPF (Koebel, Loveman, Schreiber, Steele, and Zosel 1994), CRAFT (Pase, MacDonald, and Meltzer 1994) or OpenMP (1997) result from the intermixing of these two issues. They are different and Co-Array Fortran keeps them separate.

First, consider work distribution. Co-Array Fortran adopts the Single-Program-Multiple-Data (SPMD) programming model. A single program is replicated a fixed number of times, each replication having its own set of data objects. Such a model is new to Fortran, which assumes a single program executing alone with a single set of data objects. Each replication of the program is called an **image**. Each image executes asynchronously and the normal rules of Fortran apply, so the execution path may differ from image to image. The programmer determines the actual path for the image with the help of a unique image index by using

1



ISO/IEC JTC 1/SC 22/WG 5/N1814

DRAFT
(for WG5 ballot)
12th March 2010
11:38

INTERNATIONAL STANDARD

ISO/IEC
DIS
1539-1

Third edition
2010-03-12

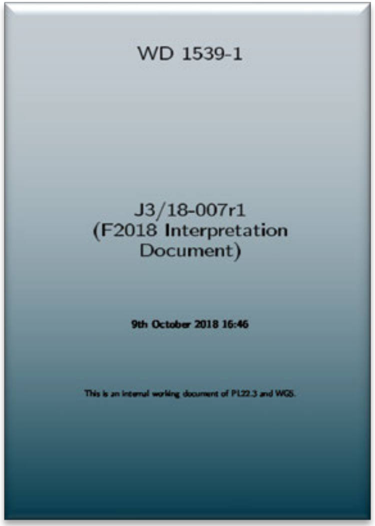
Information technology — Programming languages — Fortran —

Part 1:

Base language

Technologies de l'information — Langages de programmation — Fortran —

Partie 1: Langage de base



WD 1539-1

J3/18-007r1
(F2018 Interpretation
Document)

9th October 2018 16:46

This is an internal working document of PL22.3 and WG5.

1998

2010

2018

Additional parallel features:

Collective subroutines, events, teams, failed-image handling, *more* atomics.

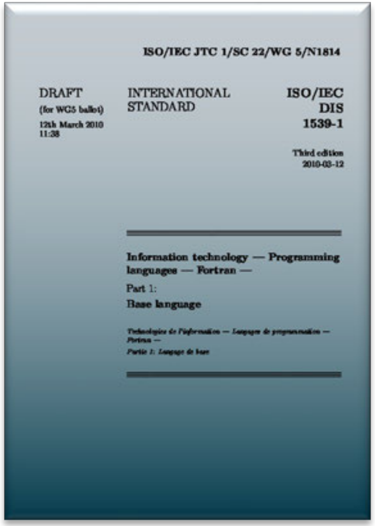
Background: Parallelism in Fortran


Co-Array Fortran for parallel programming

Abstract

1 Introduction

1998


ISO/IEC JTC 1/SC 22/WG 5/N1814

DRAFT (for WG5 ballot) 12th March 2010 11:38

INTERNATIONAL STANDARD

ISO/IEC DIS 1539-1

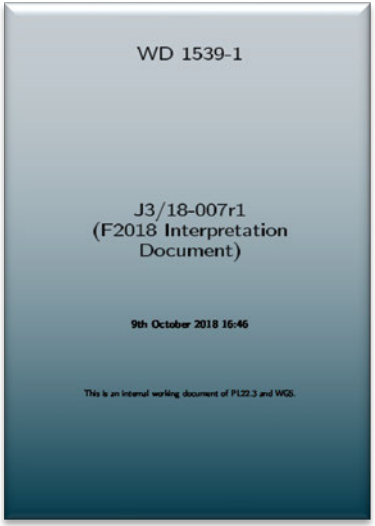
Third edition 2010-03-12

Information technology — Programming languages — Fortran —

Part 1:

Base language

2010


WD 1539-1

J3/18-007r1 (F2018 Interpretation Document)

9th October 2018 16:46

This is an internal working document of PL22.3 and WG5.

2018


WD 1539-1

J3/23-007r1 (Draft Fortran 2023)

13th June 2023 16:07

This is an internal working document of INCITS/Fortran and ISO/IEC JTC1/SC22/WG5.

2023

- Notified access
- `Do concurrent` reductions

Overview

01

Background:
Coarray Fortran

02

Motivations:
CAF and PRIF

03

The Compiler Landscape

04

PRIF Design :
Overview & Status

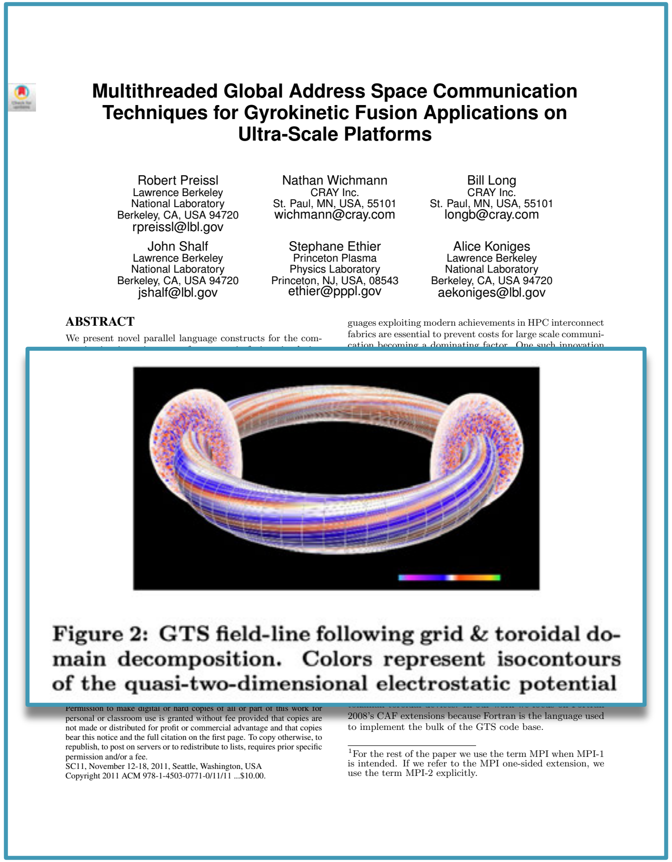
05

PRIF Implementation:
Caffeine

06

Future Work

CAF Motivations: Performance + Programmability



Multithreaded Global Address Space Communication Techniques for Gyrokinetic Fusion Applications on Ultra-Scale Platforms

Robert Preissl
Lawrence Berkeley National Laboratory
Berkeley, CA, USA 94720
rpreissl@lbl.gov

Nathan Wichmann
CRAY Inc.
St. Paul, MN, USA, 55101
wichmann@cray.com

Bill Long
CRAY Inc.
St. Paul, MN, USA, 55101
longb@cray.com

John Shalf
Lawrence Berkeley National Laboratory
Berkeley, CA, USA 94720
jshalf@lbl.gov

Stephane Ethier
Princeton Plasma Physics Laboratory
Princeton, NJ, USA, 08543
ethier@pppl.gov

Alice Koniges
Lawrence Berkeley National Laboratory
Berkeley, CA, USA 94720
aekoniges@lbl.gov

ABSTRACT

We present novel parallel language constructs for the com-

guages exploiting modern achievements in HPC interconnect fabrics are essential to prevent costs for large scale communi-

Figure 2: GTS field-line following grid & toroidal domain decomposition. Colors represent isocontours of the quasi-two-dimensional electrostatic potential

2008's CAF extensions because Fortran is the language used to implement the bulk of the GTS code base.

[1]For the rest of the paper we use the term MPI when MPI-1 is intended. If we refer to the MPI one-sided extension, we use the term MPI-2 explicitly.

Application focus:

– The shift phases of charged particles in a tokamak simulation code

Programming models studied:

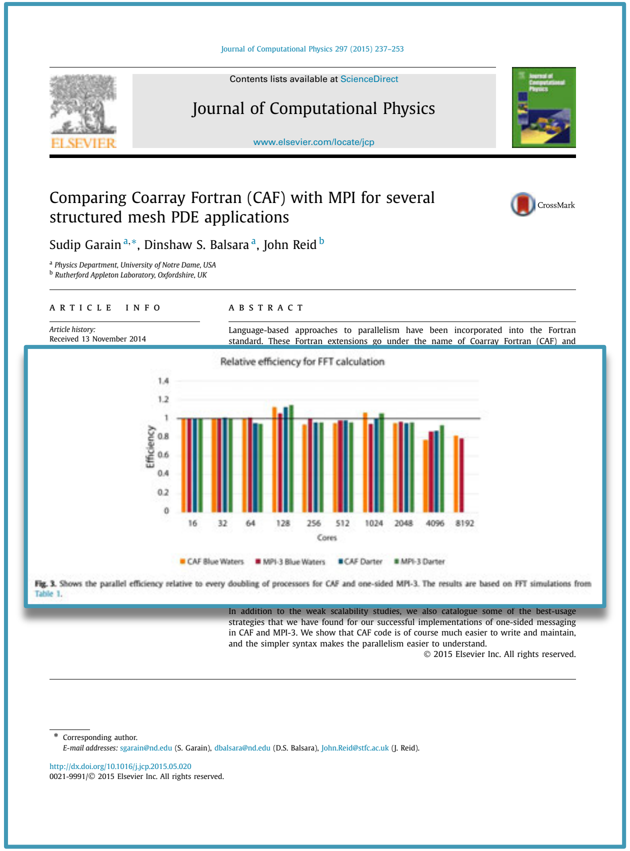
– CAF + OpenMP or
– MPI Message-Passing + OpenMP

Highlights:

– Experiments on up to 130,560 processors
– 58% speedup with CAF relative to best multithreaded MPI shifter algorithm on largest problem
– “the complexity required to implement… MPI-2 one-sided, in addition to several other semantic limitations, is prohibitive.”

Preissl, R., Wichmann, N., Long, B., Shalf, J., Ethier, S., & Koniges, A. (2011, November). 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* (pp. 1-11).

CAF Motivations: Performance + Programmability


Journal of Computational Physics 297 (2015) 237–253
Contents lists available at ScienceDirect
Journal of Computational Physics
www.elsevier.com/locate/jcp

Comparing Coarray Fortran (CAF) with MPI for several structured mesh PDE applications

Sudip Garain[a,*], Dinshaw S. Balsara[a], John Reid[b]

[a] Physics Department, University of Notre Dame, USA
[b] Rutherford Appleton Laboratory, Oxfordshire, UK

A R T I C L E   I N F O

Article history:
Received 13 November 2014

A B S T R A C T

Language-based approaches to parallelism have been incorporated into the Fortran standard. These Fortran extensions go under the name of Coarray Fortran (CAF) and

Fig. 3. Shows the parallel efficiency relative to every doubling of processors for CAF and one-sided MPI-3. The results are based on FFT simulations from Table 1.

In addition to the weak scalability studies, we also catalogue some of the best-usage strategies that we have found for our successful implementations of one-sided messaging in CAF and MPI-3. We show that CAF code is of course much easier to write and maintain, and the simpler syntax makes the parallelism easier to understand.

* Corresponding author.
E-mail addresses: sgarain@nd.edu (S. Garain), dbalsara@nd.edu (D.S. Balsara), John.Reid@stfc.ac.uk (J. Reid).

http://dx.doi.org/10.1016/j.jcp.2015.05.020

Applications and algorithms studied:

- Magnetohydrodynamics (MHD)
- 3D Fast Fourier Transforms (FFTs)
- Multigrid methods with point-wise smoothers requiring fine-grained data transfers

Programming models studied:

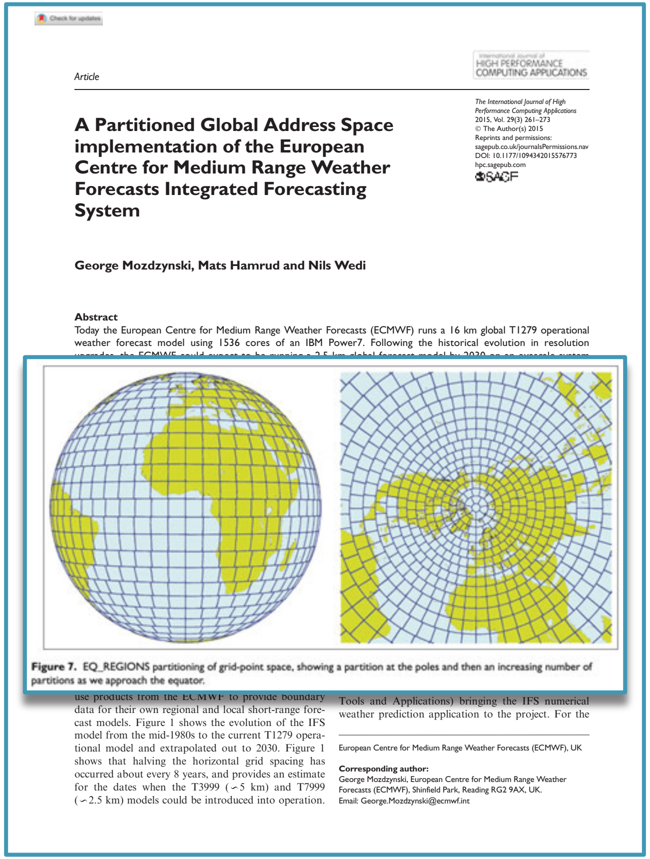
- CAF or
- One-sided MPI RMA

Highlights:

- Simulations on up to 65,536 cores
- "… CAF either draws level with MPI-3 or shows a slight advantage over MPI-3"
- "CAF code is of course much easier to write and maintain"

Garain, S., Balsara, D. S., & Reid, J. (2015). Comparing Coarray Fortran (CAF) with MPI for several structured mesh PDE applications. *Journal of Computational Physics*, *297*, 237-253.

CAF Motivations: Performance + Programmability



Check for updates

Article

International Journal of HIGH PERFORMANCE COMPUTING APPLICATIONS

The International Journal of High Performance Computing Applications 2015, Vol. 29(3) 261–273 © The Author(s) 2015 Reprints and permissions: sagepub.co.uk/journalsPermissions.nav DOI: 10.1177/1094342015576773 hpc.sagepub.com SAGE

A Partitioned Global Address Space implementation of the European Centre for Medium Range Weather Forecasts Integrated Forecasting System

George Mozdzynski, Mats Hamrud and Nils Wedi

Abstract
Today the European Centre for Medium Range Weather Forecasts (ECMWF) runs a 16 km global T1279 operational weather forecast model using 1536 cores of an IBM Power7. Following the historical evolution in resolution

Figure 7. EQ_REGIONS partitioning of grid-point space, showing a partition at the poles and then an increasing number of partitions as we approach the equator.

use products from the ECMWF to provide boundary data for their own regional and local short-range forecast models. Figure 1 shows the evolution of the IFS model from the mid-1980s to the current T1279 operational model and extrapolated out to 2030. Figure 1 shows that halving the horizontal grid spacing has occurred about every 8 years, and provides an estimate for the dates when the T3999 (~5 km) and T7999 (~2.5 km) models could be introduced into operation.

Tools and Applications) bringing the IFS numerical weather prediction application to the project. For the

European Centre for Medium Range Weather Forecasts (ECMWF), UK

Corresponding author:
George Mozdzynski, European Centre for Medium Range Weather Forecasts (ECMWF), Shinfield Park, Reading RG2 9AX, UK.
Email: George.Mozdzynski@ecmwf.int

Application:

– European Centre for Medium Range Weather Forecasts (ECMWF) operational model

Programming models studied:

– CAF or
– MPI Message-Passing

Highlights:

– Simulations on >60K cores
– "… performance improveme peaks at 21% around 40K c

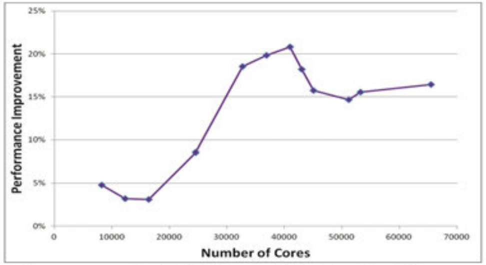
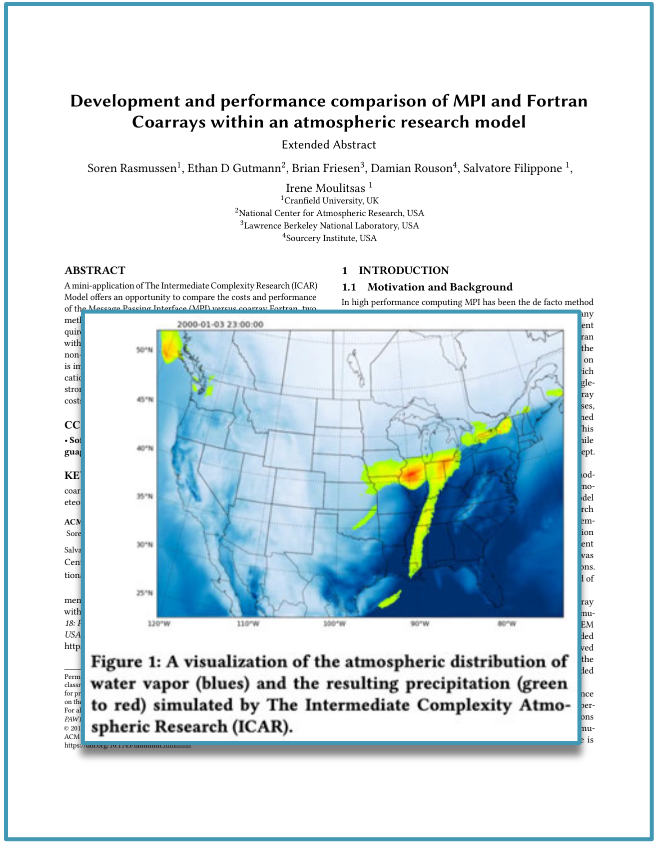


Figure 14. Performance improvement of the T2047 (~10 km) model with 137 levels by using Fortran2008 coarrays on HECToR (Cray XE6).

Mozdzynski, G., Hamrud, M., & Wedi, N. (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), 261-273.

CAF Motivations: Performance Portability



Development and performance comparison of MPI and Fortran Coarrays within an atmospheric research model

Extended Abstract

Soren Rasmussen[1], Ethan D Gutmann[2], Brian Friesen[3], Damian Rouson[4], Salvatore Filippone [1], Irene Moulitsas [1]

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

ABSTRACT

A mini-application of The Intermediate Complexity Research (ICAR) Model offers an opportunity to compare the costs and performance

1 INTRODUCTION

1.1 Motivation and Background

In high performance computing MPI has been the de facto method

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

Application:

- Intermediate Complexity Atmospheric Research (ICAR) model
- Regional impacts of global climate change

Programming models studied:

- CAF over one-sided MPI RMA
- CAF over OpenSHMEM
- MPI Message-Passing
- Cray CAF implementation

Highlights:

- "… we used up to 25,600 processes and found that at every data point OpenSHMEM was outperforming MPI."
- "The coarray Fortran with MPI backend stopped being usable as we went over 2,000 processes… the initialization time started to increase exponentially"

Rasmussen, S., Gutmann, E. D., Friesen, B., Rouson, D., Filippone, S., & Moulitsas, I. (2018). Development and performance comparison of MPI and Fortran Coarrays within an atmospheric research model. In *Workshop*.

Overview

01
Background:
Coarray Fortran

02
Motivations:
CAF and PRIF

03
The Compiler Landscape

04
PRIF Design:
Overview & Status

05
PRIF Implementation:
Caffeine

06
Future Work

Compiler Status

Supporting CAF features:

- Cray
- Intel
- GNU
- NAG

Automatic offloading of `do concurrent`:

- NVIDIA
- Intel
- Cray

LLVM Flang:

- Parses and verifies CAF syntax and semantics
- Does not yet lower CAF features
- Berkeley Lab develops

  -- Frontend unit tests

  -- Frontend bug fixes

  -- PRIF: a specification

  -- Caffeine: a PRIF implementation using GASNet-EX

The World's Shortest Bug Reproducer

```
end
```

Overview

01

Background:
Coarray Fortran

02

Motivations:
CAF and PRIF

03

The Compiler Landscape

04

PRIF Design :
Overview & Status

05

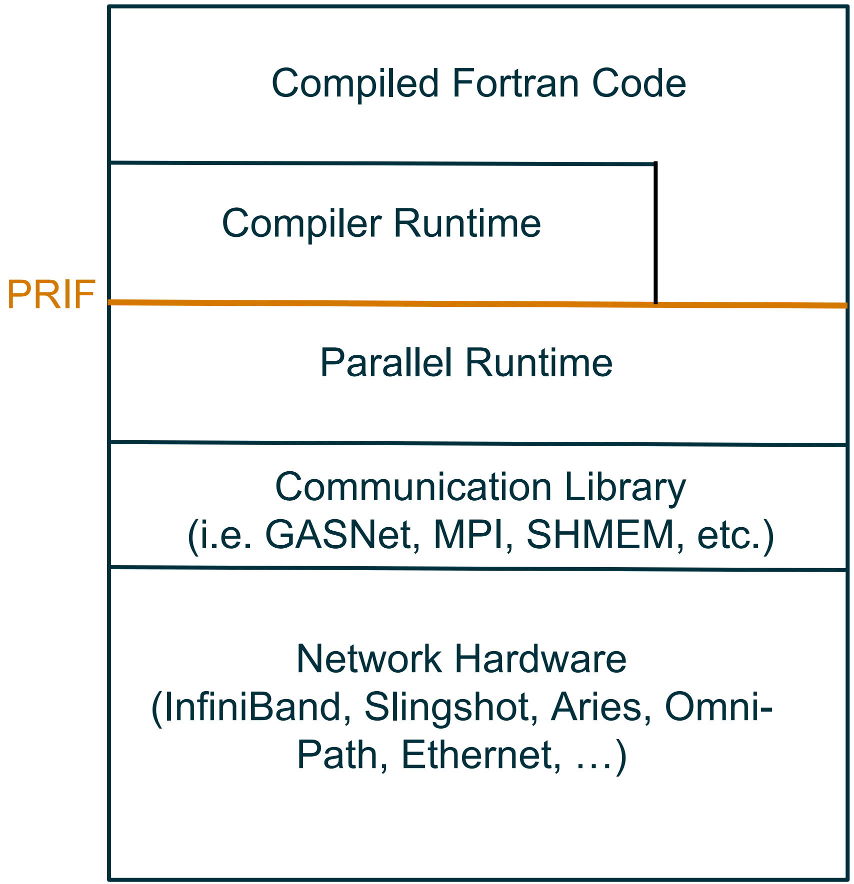
PRIF Implementation:
Caffeine

06

Future Work

PRIF

- Enable a compiler to leverage multiple alternative PRIF implementations
  - e.g., vendor-specific ones
- Enable a PRIF implementation to support multiple compilers
- Isolate a compiler's support of the parallel features of the language from any particular details of the communication infrastructure
- Our group's experience with UPC and OpenCoarrays has shown this to be valuable



Fortran Parallel Source Code & PRIF Equivalents

- Compiler responsible for processing user's source code and producing calls to PRIF implementation
- PRIF specific types
  - `prif_coarray_handle`, etc.
- PRIF provides procedures for:
  - associated intrinsic subroutines and functions
    - `num_images` supported by `prif_num_images`, etc
  - coarray allocation and accesses
    - `prif_allocate_coarray`, `prif_put`, `prif_get`, etc
- Intrinsic derived types that PRIF provides:
  - `prif_team_type`, `prif_event_type`, `prif_lock_type`, `prif_notify_type`
- ISO_FORTRAN_ENV constants that PRIF provides:
  - `prif_atomic_int_kind`, `prif_current_team`, `prif_stat_unlocked`, etc.

Intrinsic Functions and Subroutines

```
me = this_image()
```

⬇

```
call prif_this_image(image_index=me)
```

```
call co_sum(a, 1)
```

⬇

```
call prif_co_sum(a=a, result_image=1_c_int)
```

PRIF Design Overview: Responsibilities

Compiler

- Establish and initialize static coarrays prior to main
- Track corank of coarrays
- Track local coarrays for implicit deallocation when exiting a scope
- Initialize a coarray with `source=` as part of `allocate`
- Provide `prif_critical_type` coarrays for `critical`
- Provide `final` subroutine for all derived types that are finalizable or that have allocatable components that appear in a coarray
- Variable allocation status tracking, including use of `move_alloc`

PRIF Implementation

- Allocate and deallocate a coarray
- Reference a coindexed object
- Team statements/constructs:
- Team stack abstraction
- Track coarrays for implicit deallocation at `end team`
- Intrinsics functions related to parallelism, e.g., `num_images`, `coshape`, `image_index`
- Intrinsic subroutine: Atomics, collectives
- Synchronization statements
- Events, locks, critical, notify

prif_coarray_handle

- Derived type with private components → opaque to compiler
- Returned by call to `prif_allocate_coarray`
- Serves as a reference to a coarray descriptor
- Passed back and forth across PRIF for coarray operations

```
! Caffeine’ descriptor:

type, private, bind(C) :: handle_data
    private
    type(c_ptr) :: coarray_data
    integer(c_int) :: corank
    integer(c_size_t) :: coarray_size
    integer(c_size_t) :: element_length
    type(c_funptr) :: final_func
    type(c_ptr) :: previous_handle, next_handle
    integer(c_intmax_t) :: lcobounds(15), ucobounds(15)
  end type
```

Coarray allocation

```
integer :: coarr(10)[*]
```

`coarr` is a static *array* coarray with

- [lcobound : ucobound] = [1 : num_images()] => corank = 1
- [lbound : ubound] = [1 :10] => rank = 1

```
call prif_allocate_coarray( &
  lcobounds=[1_c_intmax_t], ucobounds=[prif_num_images()], &          ! in
  lbounds=[1_c_intmax_t], ubounds=[10_c_intmax_t)], &                 ! in
  element_length=size_of_default_int_in_bytes, final_func=c_null_funptr, & ! in
  coarray_handle=coarr_coarray_handle, allocated_memory=mem)          ! out
                                       ! ^ facilitate local access w/o calling PRIF
```

Coarray accesses

```
coarr[1] = func_call()
```

```
call prif_put( &
  coarray_handle=coarr_coarray_handle, cosubscripts=int([1], c_intmax_t), &
  value=func_call(), first_element_addr=c_loc(coarr))
```

PRIF Progress

- Initial publication: PRIF 0.2 (December 2023), LBNL Tech. Report
- Submitted PRIF pull request on **`GitHub.com/llvm-project`**
- PRIF 0.3 (May 2024) reflects feedback from SiPearl
- Future Work: PRIF 0.4 will reflect feedback from NVIDIA

Bonachea, D., Rasmussen, K., Richardson, B., Rouson, D. (2024) **"Parallel Runtime Interface for Fortran (PRIF) Specification, Revision 0.3",** *Lawrence Berkeley National Laboratory Technical Report,* LBNL 2001590, doi: 10.25344/S4501W

Overview

01
Background:
Coarray Fortran

02
Motivations:
CAF and PRIF

03
The Compiler Landscape

04
PRIF Design :
Overview & Status

05
PRIF Implementation:
Caffeine

06
Future Work

Berkeley Lab's Caffeine: PRIF Implementation Status (Some Partial)

Program launch & termination:
- `prif_init`
- `prif_stop`
- `prif_error_stop`

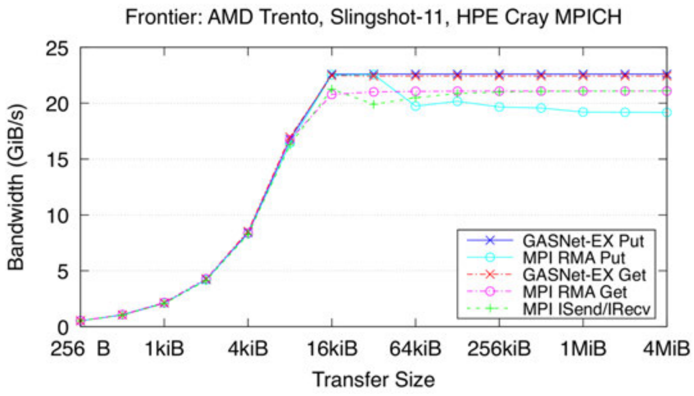
Image inquiry functions:
- `prif_this_image`
- `prif_num_images`
- `prif_image_index`

Coarray communication:
- `prif_put`
- `prif_get`

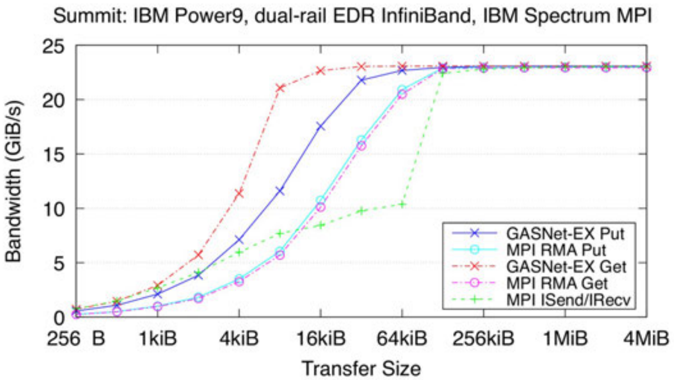
Coarray & component allocation:
- `prif_allocate_coarray`
- `prif_deallocate_coarray`
- `prif_allocate`
- `prif_deallocate`

Synchronization:
- `prif_sync_all`

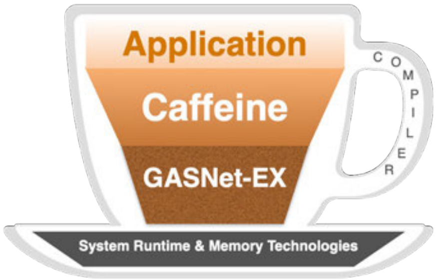
Collective Subroutines
- `prif_co_min`
- `prif_co_max`
- `prif_co_sum`
- `prif_co_broadcast`
- `prif_co_reduce`



gasnet.lbl.gov/performance



Hargrove P, Bonachea D. "GASNet-EX RMA Communication Performance on Recent Supercomputing Systems", 2022 IEEE/ACM Parallel Applications Workshop, Alternatives To MPI+X (PAW-ATM), November 2022. doi:10.25344/S40C7D



go.lbl.gov/caffiene

Overview

01

Background:
Coarray Fortran

02

Motivations:
CAF and PRIF

03

Backdrop:
The Compiler Landscape

04

PRIF Design :
Overview & Status

05

PRIF Implementation:
Caffeine

06

Future Work

Future Work

- Lower CAF syntax to PRIF invocations after PR has been approved and merged
- Complete Caffeine support of PRIF
- Track progress: https://github.com/BerkeleyLab/flang-testing-project/projects/7
- For more information or to provide feedback:
  - We welcome issues and PRs at the above GitHub Repository
  - Discourse Post
  - Email: lbl-flang@lbl.gov
  - Specification Working Draft: https://go.lbl.gov/prif

Acknowledgements

- This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research.
- This research was supported in part by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration
- This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility located at Lawrence Berkeley National Laboratory, operated under Contract No. DE-AC02-05CH11231

Thank You

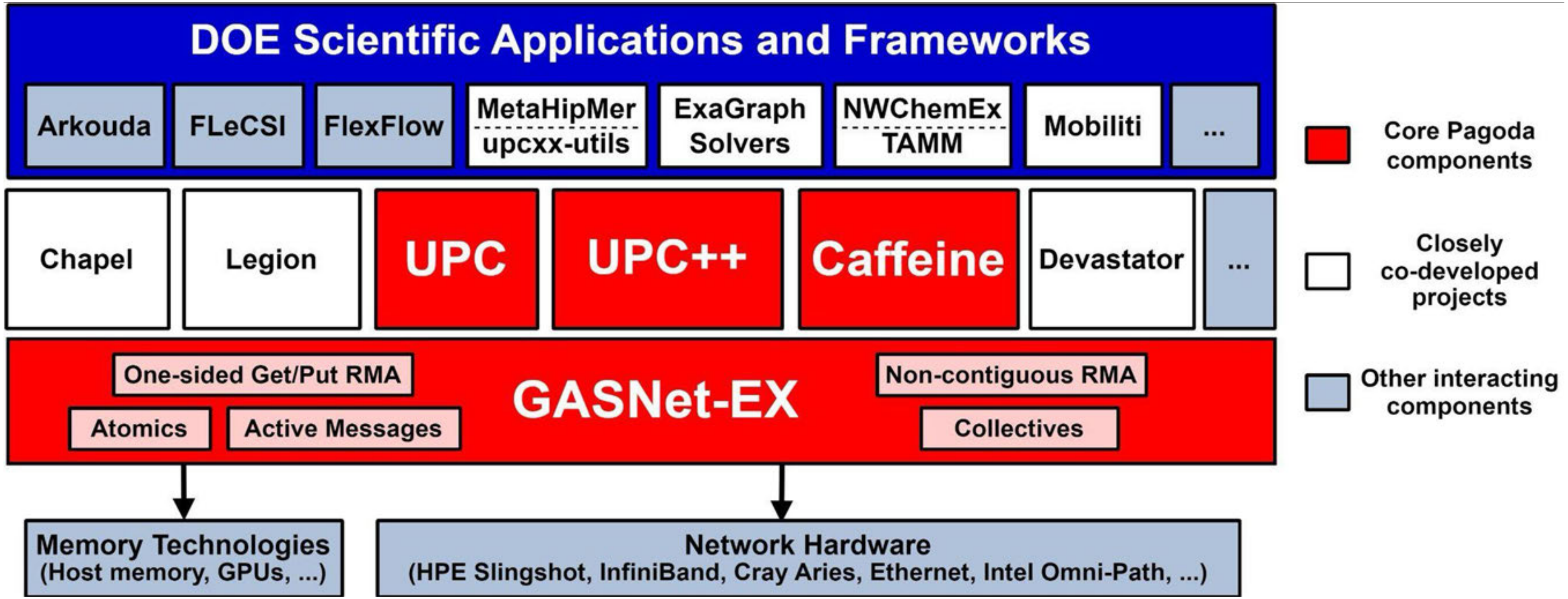
Questions

Email: fortran@lbl.gov

Fortran efforts at LBNL: fortran.lbl.gov

Specification Working Draft: go.lbl.gov/prif

What is GASNet?



Who We are

We have experience developing parallel runtimes, parallel applications, Flang frontend parallel features, and parallel unit tests:

- **OpenCoarrays:** Fanfarillo, A., Burnus, T., Cardellini, V., Filippone, S., Nagle, D., & Rouson, D. (2014). “OpenCoarrays: open-source transport layers supporting coarray Fortran compilers.” In *Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models* (pp. 1-11). doi: 10.1145/2676870.2676876
- **Caffeine:** Rouson, D., & Bonachea, D. (2022). “Caffeine: CoArray Fortran Framework of Efficient Interfaces to Network Environments.” In *2022 IEEE/ACM Eighth Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC)* (pp. 34-42). IEEE. doi: 10.25344/S4459B
- **Flang:** Rasmussen, K., Rouson, D., George, N., Bonachea, D., Kadhem, H., & Friesen, B. (2022) "Agile Acceleration of LLVM Flang Support for Fortran 2018 Parallel Programming", Research Poster at the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC22). doi: 10.25344/S4CP4S
- **Berkeley UPC**: Chen, Bonachea, Duell, Husbands, Iancu, Yelick,, "A Performance Analysis of the Berkeley UPC Compiler", Proceedings of the International Conference on Supercomputing (ICS), ACM, June 23, 2003, 63--73, doi: 10.1145/782814.782825
- **UPC++:** Bachan, Baden, Hofmeyr, Jacquelin, Kamil, Bonachea, Hargrove, Ahmed, "UPC++: A High-Performance Communication Framework for Asynchronous Computation", 33rd IEEE International Parallel & Distributed Processing Symposium (IPDPS'19), May 2019, doi: 10.25344/S4V88H

Why not OpenCoarrays?

- Is hardwired to gfortran, e.g., many procedures manipulate gfortran-specific descriptors
- The interface implicitly assumes a MPI backend
- Only the MPI layer is maintained (GASNet & OpenSHMEM layers are legacy codes)
- Lacks full support for some parallel features (e.g., teams).
- Has a bus factor of ~1.