Parallel Runtime Interface for Fortran (PRIF)

Dan Bonachea\textsuperscript{1}, Katherine Rasmussen\textsuperscript{1}, Brad Richardson\textsuperscript{2}, Damian Rouson\textsuperscript{1}

\textsuperscript{1}Computer Languages and Systems Software (CLaSS) Group
National Energy Research Scientific Computing (NERSC) Center\textsuperscript{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)

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.”

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

Additional parallel features:

Collective subroutines, events, teams, failed-image handling, more atomics.
Background: Parallelism in Fortran

Co-Array Fortran for parallel programming
Robert W. Numrich, Victor Dagum, Inc.
and
Leslie Feibelman, Battelle Memorial Institute

Abstract
Co-Array Fortran provides simple parallel programming, accessible to users from many programming backgrounds. It is a Fortran with a small number of simple, explicit constructs for parallel computing. Co-Array Fortran is intended for complex, scientific applications where the programmer must reason about parallelism, but where the programmer needs help to understand how to reason about parallelism.

Introduction: Parallelism in Fortran

1998

Notified access

Do concurrent reductions

2010

2018

2023
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

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.”

CAF Motivations: Performance + Programmability

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”

CAF Motivations: Performance + Programmability

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

Programming models studied:
– CAF
– MPI Message-Passing

Highlights:
– Simulations on >60K cores
– “... performance improvement peaks at 21% around 40K cores”

CAF Motivations: Performance Portability

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”
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

dend
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
• 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:

```fortran
  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

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

`coarr` is a static array coarray with
- \([\text{lcobound} : \text{ucobound}] = [1 : \text{num\_images()}] \Rightarrow \text{corank} = 1\)  
- \([\text{lbound} : \text{ubound}] = [1 : 10] \Rightarrow \text{rank} = 1\)

```fortran
call prif\_allocate\_coarray(  
    lcobounds=[1_c\_intmax\_t], ucobounds=[prif\_num\_images()], &  
    lbounds=[1_c\_intmax\_t], ubounds=[10_c\_intmax\_t]), &  
    element\_length=size\_of\_default\_int\_in\_bytes, final\_func=c\_null\_funptr, &  
    coarray\_handle=coarr\_coarray\_handle, allocated\_memory=mem)  
```

! ^ 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

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

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](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](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:


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.