BERKELEY LAB
Bringing Science Solutions to the World

U.S. DEPARTMENT OF ENERGY
Office of Science

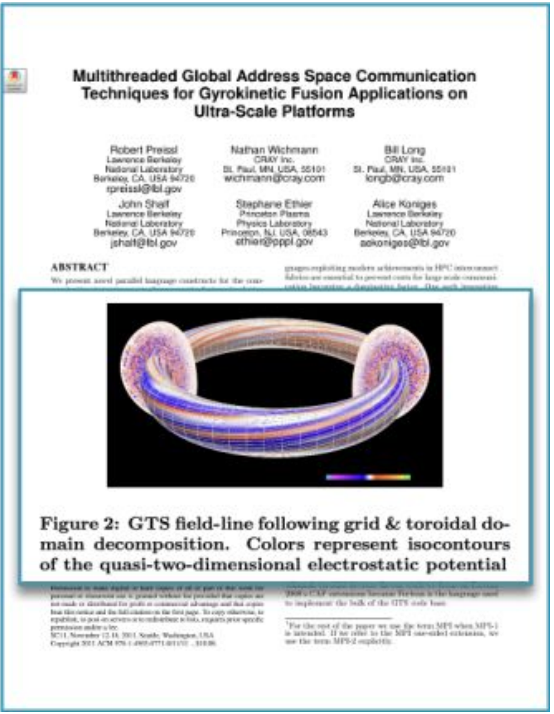
Parallel Runtime Interface for Fortran

A Multi-Image Solution for LLVM Flang

Dan Bonachea, Katherine Rasmussen, **Brad Richardson**, Damian Rouson
https://fortran.lbl.gov

November 2024

Outline


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

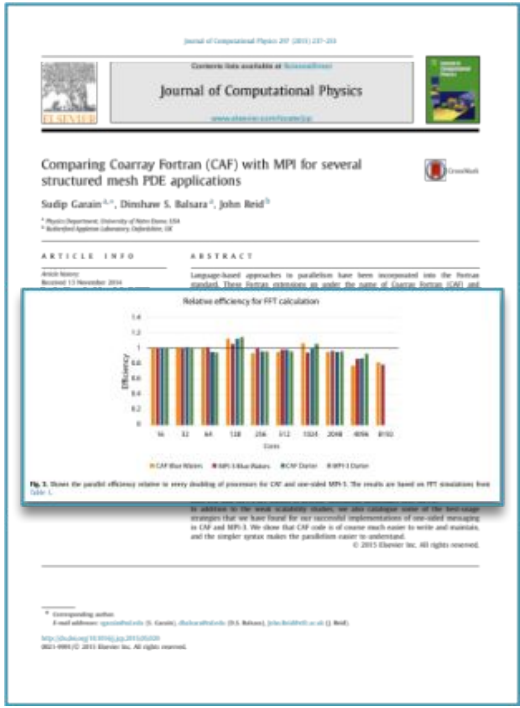
ABSTRACT

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

Why Parallel Fortran Matters

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).

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.


Journal of Computational Physics

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

Sudip Garain, Dinshaw S. Balsara, John Reid

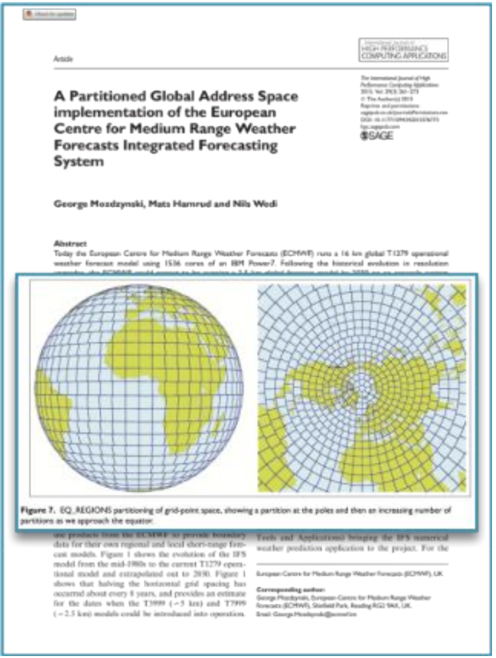
ARTICLE INFO

ABSTRACT

- 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.”

- 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”

Why Parallel Fortran Matters


Article

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

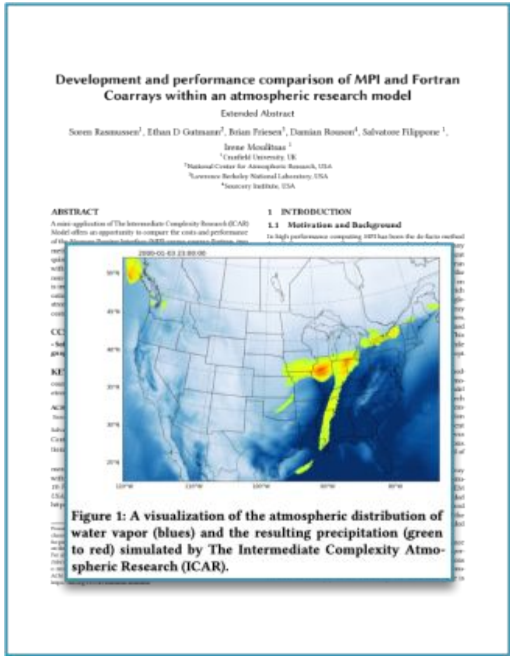
George Mozdzynski, Mats Hamrud and Nils Wedi

Abstract

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.

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.

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 *Proceedings of PAW-ATM 18: Parallel Applications Workshop, Alternatives to MPI.*


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

Extended Abstract

ABSTRACT

1 INTRODUCTION

1.1 Motivation and Background

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).

- Simulations on >60K cores
- "… performance improvement from switching to CAF peaks at 21% around 40K cores"

- "… 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"

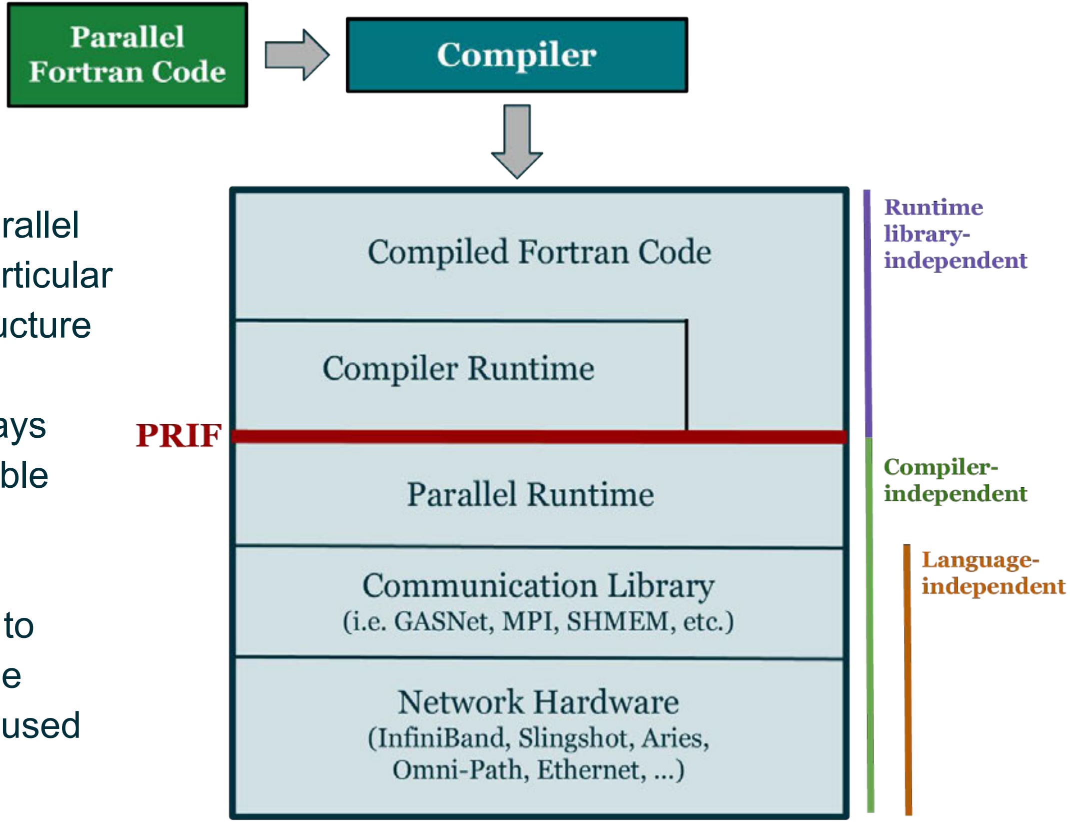
Parallel Features in Modern Fortran

- Statements
  - Synchronization
    - Explicit: `SYNC ALL`, `SYNC IMAGES`, `SYNC MEMORY`, `SYNC TEAM`
    - Implicit: `ALLOCATE`, `DEALLOCATE`, `STOP`, `END`, `MOVE_ALLOC`
  - Events: `EVENT POST`, `EVENT WAIT`
  - Notify: `NOTIFY WAIT`
  - Error termination: `ERROR STOP`
  - Locks: `LOCK`, `UNLOCK`
  - Failed images: `FAIL IMAGE`
  - Teams: `FORM TEAM`, `CHANGE TEAM`
  - Critical sections: `CRITICAL`, `END CRITICAL`
- Coarray Accesses (`[...]`)
- Intrinsic functions: `NUM_IMAGES`, `THIS_IMAGE`, `LCOBOUND`, `UCOBOUND`, `TEAM_NUMBER`, `GET_TEAM`, `FAILED_IMAGES`, `STOPPED_IMAGES`, `IMAGE_STATUS`, `COSHAPE`, `IMAGE_INDEX`
- Intrinsic subroutines
  - Collective subroutines: `CO_SUM`, `CO_MAX`, `CO_MIN`, `CO_REDUCE`, `CO_BROADCAST`
  - Atomic subroutines: `ATOMIC_ADD`, `ATOMIC_AND`, `ATOMIC_CAS`, `ATOMIC_DEFINE`, `ATOMIC_FETCH_ADD`, `ATOMIC_FETCH_AND`, `ATOMIC_FETCH_OR`, `ATOMIC_FETCH_XOR`, `ATOMIC_OR`, `ATOMIC_REF`, `ATOMIC_XOR`
  - Other subroutines: `EVENT_QUERY`
- Types, kind type parameters, and values
  - Intrinsic derived types: `EVENT_TYPE`, `TEAM_TYPE`, `LOCK_TYPE`, `NOTIFY_TYPE`
  - Atomic kind type parameters: `ATOMIC_INT_KIND` and `ATOMIC_LOGICAL_KIND`
  - Values: `STAT_FAILED_IMAGE`, `STAT_LOCKED`, `STAT_LOCKED_OTHER_IMAGE`, `STAT_STOPPED_IMAGE`, `STAT_UNLOCKED`, `STAT_UNLOCKED_FAILED_IMAGE`

Motivation

What's this for?

- 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 Berkeley UPC and OpenCoarrays has shown this approach valuable
- Enable a compiler to target multiple implementations of PRIF
  - e.g. enable a hardware vendor to supply their own parallel runtime
- Enable a PRIF implementation to be used by multiple compilers



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` statement
- Provide `prif_critical_type` coarrays for `CRITICAL` constructs
- 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`

Parallel Runtime

- Track coarrays for implicit deallocation at `END TEAM`
- Allocate and deallocate a coarray
- Reference a coindexed object
- Team stack abstraction
- `FORM TEAM`, `CHANGE TEAM`, `END TEAM`
- Intrinsic functions related to parallel Fortran, like `NUM_IMAGES`, etc
- Atomic subroutines
- Collective subroutines
- Synchronization statements
- Events, notify
- Locks
- `CRITICAL` construct

PRIF Design Overview

Parallel Features Directly Translatable to Use of Fortran Library

```
me = THIS_IMAGE()
```

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

```
call CO_SUM(a, result_image=1)
```

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

```
co_arr[1] = some_calc()
```

```
call prif_put( &
    co_arr_coarray_handle, &
    INT([1], c_intmax_t), &
    some_calc(), &
    INT(STORAGE_SIZE(arr)/8, c_size_t), &
    C_LOC(co_arr))
```

Why Define the Interface in Fortran?

- **Motivation:** community contributions and use by any Fortran compiler
- **Proposition:** Only need a subset of non-parallel Fortran 2018 features
    - assumed-type and assumed-rank arguments:
      `type(*), intent(in) :: array(..)`
    - C interoperability:
      `type(c_ptr), integer(c_int)`, etc.
- **Prototype:** Caffeine

Some Examples Influencing Design Decisions

Program Startup and Shutdown

Shortest Parallel Fortran Program

```fortran
end
```

→

```fortran
use prif
...
call prif_init(...)
...
call prif_stop(...)
end
```

* Note: `prif_stop` also used in place of Fortran `STOP` statement

Program Startup and Shutdown Design Requirement

- `prif_init`
  - idempotent initialization of the PRIF library
- `prif_stop`
  - collective program exit
  - arguments equivalent to `STOP` statement options
- `prif_error_stop`
  - non-collective program exit
  - arguments equivalent to `ERROR STOP` statement options

Coarray Design Requirements

- Need procedures to allocate and deallocate coarray
  - `prif_allocate_coarray`
  - `prif_deallocate_coarray`
- Need a handle for a coarray descriptor, separate from the coarray variable
  - gives PRIF implementation ability to track state for different coarrays
  - without requiring PRIF to understand compiler specific variable “descriptors”
  - `prif_coarray_handle`

Static coarrays need allocated and (potentially) initialized prior to main

```
program main
   integer :: i[*]
end program
```

→

```
program main
   ...
   type(prif_coarray_handle) :: h
   ...
   call prif_init(...)
   ...
   call prif_allocate_coarray(h, ...)
   ...
   call prif_deallocate_coarray(h, ...)
   ...
   call prif_stop(...)
end program
```

Local coarrays are implicitly deallocated when exiting a scope

```
subroutine sub
   integer, allocatable :: i[:]
   call might_allocate(i)
end subroutine
```

→

```
subroutine sub
   ...
   call might_allocate(...)
   if (allocated(...)) &
        call prif_deallocate_coarray(...)
end subroutine
```

The compiler must ensure that coarray (de)allocation updates the appropriate variable's allocation status so that it can know if `prif_deallocate_coarray` should be called at end of scope.

Derived types that are finalizable or that have allocatable components can appear in a coarray

```
program main
   type :: t
       integer, allocatable :: a
   end type
   type(t), allocatable :: c[*]
   allocate(c[*])
   allocate(c%a)
   deallocate(c)
end program
```

→

```
program main
   ...
   call prif_allocate_coarray(...)
   call prif_allocate(...)
   call prif_deallocate_coarray(...)
   ...
end program
```

- Non-collective, non-coarray `prif_allocate` and `prif_deallocate`
- Communication (put/get) operations that work on non-coarray storage
- Compiler provided call-back to perform deallocation of components, call final subroutine, and update a variable's allocation status

Coarrays can be implicitly deallocated at end team statement

```
program main
   ...
   type(team_type) :: my_team
   type(t), allocatable :: c[:]
   ...
   form team (..., my_team)
   change team (my_team)
       call sub
   end team
contains
   subroutine sub
       allocate(c[*])
       allocate(c%a)
   end subroutine
end program
```

→

```
program main
   ...
   call prif_form_team(...)
   call prif_change_team(...)
   call sub
   call prif_end_team(...)
   ...
contains
   subroutine sub
       call prif_allocate_coarray(...)
       call prif_allocate(...)
   end subroutine
end program
```

Teams Design Requirements

PRIF Implementation must track which coarrays are allocated during a `CHANGE TEAM` construct so that they can be deallocated at the matching `END TEAM` statement

PRIF passes coarray handle to coarray cleanup call-back, and provides a way for the compiler to store and query information in the coarray descriptor to keep track of which coarray variable is being deallocated

Progress

- Finished PRIF draft specification Version 0.4
- Have submitted PRIF specification in a design doc to LLVM-Project Repository
  - We've received some review comments and are working on PRIF 0.5
- Caffeine, LBL's implementation of PRIF, has partial or full support for the following features:
  - Program launch and termination: `prif_init`, `prif_stop`
  - Image enumeration: `prif_this_image` and `prif_num_images`
  - Image Queries: `prif_image_index`
  - Coarray allocation: `prif_allocate_coarray`, `prif_deallocate_coarray`, `prif_allocate`, `prif_deallocate`
  - Contiguous RMA: `prif_put`, `prif_get`, `prif_put_indirect`, `prif_get_indirect`
  - Global synchronization: `prif_sync_all`
  - Collective subroutines: `prif_co_min`, `prif_co_max`, `prif_co_sum`, `prif_co_broadcast`, and `prif_co_reduce`
  - Teams: `prif_form_team`, `prif_change_team`, `prif_end_team`

Next Steps

- Produce PRIF Version 0.5
- Finish implementation in Caffeine
- Integration into flang
  - We are actively collaborating with SiPearl on this front
  - We'd love more help
- Solicit Feedback:
  - LLVM Discourse Post
  - Email: fortran@lbl.gov
  - Specification Working Draft: https://go.lbl.gov/prif
  - We welcome issues and PRs at https://go.lbl.gov/caffeine

Acknowledgements

- This research was supported 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 material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research.
- 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

Questions?

Email: fortran@lbl.gov

Fortran efforts at LBNL: fortran.lbl.gov

Specification Working Draft: go.lbl.gov/prif

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 an 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.

What is GASNet?