Title

Incremental caffeination of a terrestrial hydrological modeling framework using Fortran 2018 teams.

Authors

Rouson, Damian WI
McCreight, James L
Fanfarillo, Alessandro

Publication Date

2017

Peer reviewed

Incremental caffeination of a terrestrial hydrological modeling framework using Fortran 2018 teams

Extended Abstract

Damian Rouson
Sourcery Institute
Oakland, California
damian@sourceryinstitute.org

James L. McCreight
National Center for Atmospheric Research
Boulder, Colorado
jamesmcc@ucar.edu

Alessandro Fanfarillo
National Center for Atmospheric Research
Boulder, Colorado
elfanfa@ucar.edu

ABSTRACT

We present Fortran 2018 teams (grouped processes) running a parallel ensemble of simulations built from a pre-existing Message Passing Interface (MPI) application. A challenge arises around the Fortran standard's eschewing any direct reference to lower-level communication substrates, such as MPI, leaving any interoperability between Fortran's parallel programming model, Coarray Fortran (CAF), and the supporting substrate to the quality of the compiler implmentation. Our approach introduces CAF incrementally, a process we term "caffeination." By letting CAF initiate execution and exposing the underlying MPI communicator to the original application code, we create a one-to-one correspondence between MPI group colors and Fortran teams. We apply our approach to the National Center for Atmospheric Research (NCAR)'s Weather Research and Forcecasting Hydrological Model (WRF-Hydro). The newly caffeinated main program replaces batch job submission scripts and forms teams that each execute one ensemble member. To support this work, we developed the first compiler front-end and parallel runtime library support for teams. This paper describes the required modifications to a public GNU Compiler Collection (GCC) fork, an OpenCoarrays [1] application binary interface (ABI) branch, and a WRF-Hydro branch.

CCS CONCEPTS

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

KEYWORDS

coarray Fortran, computational hydrology, parallel programming

ACM Reference Format:
Damian Rouson, James L. McCreight, and Alessandro Fanfarillo. 2017. Incremental caffeination of a terrestrial hydrological modeling framework using Fortran 2018 teams. In *Proceedings of PAW17: Second Annual PGAS Applications Workshop, Denver, CO, USA, November 12–17, 2017 (PAW17),* 5 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 publication of the Fortran 2008 standard in 2010 [4], Fortran supports a Single-Program Multiple-Data (SPMD) programming style that facilitates the creation of a fixed number of replicas of a compiled program, wherein each replica executes asynchronously after creation. Fortran refers to each replica as an image. The primary mechanism for distributing and communicating data between images involves defining coarrays, entities that may be referenced or defined on one image by statements executing on other images. As such, a coarray defines a partitioned global address space (PGAS) in which one image referencing or defining a coarray on another image causes inter-image communication.

The seminal role that coarrays played in the development of Fortran's intrinsic parallel programming model have made it common to refer to all of modern Fortran's parallel programming features under the rubric of CAF. To date, most published CAF applications involve scenarios wherein the parallelization itself poses one of the chief challenges and necessitates the custom development of parallel algorithms. These include ordinary and partial differential equation solvers in domains ranging from nuclear fusion [7] and weather [5] to multidimensional fast Fourier transforms and multigrid numerical methods [2]. Much of the effort involved in expressing parallel algorithms for these domains centers on designing and using various coarray data structures. In such settings, the moniker CAF seems appropriate.

Less widely appreciated are the ways Fortran's intrinsic parallel programming model supports embarrassingly parallel applications, wherein the division into independent sub-problems requires little coordination between the sub-problems. To support such applications, a parallel programming model might provide for explicit sub-problem disaggregation and independent sub-problem execution without any need for PGAS data structures such as coarrays. The draft Fortran 2018 standard (previously named "Fortran 2015"[1]) offers several features that enable a considerable amount of parallel computation, coordination, and communication even without coarrays. A working definition of "embarrassingly parallel" Fortran might denote the class of use cases for which parallel algorithmic needs are met by the non-coarray parallel features, including

- Forming teams of images that communicate only with each other by default,
- Image synchronization: a mechanism for ordering the execution of program segments in differing images,

[1] A Committee Draft is at https://bit.ly/fortran-2015-draft.

- Collective subroutines: highly optimized implementations of common parallel programming patterns,
- Image enumeration: functions for identifying and counting team members,
- Global error termination.

We anticipate that a common use case will encompass an ensemble of simulations, each member of which executes as a parallel computation in a separate team. This paper presents such a use case for WRF-Hydro, a terrestrial hydrological model developed at NCAR.

1.2 Objectives

The objectives of the current work are threefold:

(1) To contribute the first-ever compiler front-end support for Fortran 2018 teams.
(2) To contribute the parallel runtime library functionality required to support the new compiler capabilities.
(3) To study and address issues arising form integrating teams into an existing MPI application, WRF-Hydro,

The compiler front-end described herein lives on the Sourcery Institute GCC fork's[2] teams branch. The parallel runtime library lives on the OpenCoarrays[3] opencoarrays-teams ABI. We are unaware of any compiler support for Fortran 2018 teams other than that developed for the current project.

2 METHODOLOGY

Fortran 2018 will facilitate forming nonoverlapping image groups, allowing more efficient and independent subproblem execution. Multiphysics applications, e.g., climate and weather models, may benefit from the consequent reduction in off-node communication, particularly when an entire team fits in a single compute node.

2.1 Teams in Fortran 2018

A team is a set of images that can execute independently of other images outside the team. At program launch, all images comprise a team designated by a language-defined `initial_team` integer identifier. Except for the initial team, every team has a parent team in a one-to-many parent/child hierarchy. A program executes a `form team` statement to create a mapping for subsequent grouping of images into new child teams. Each new team has an integer identifier that Fortran produces as the result of invoking the `team_number()` intrinsic function. Information about the team to which the current image belongs can be determined by the compiler from the collective value of the team variables on the images of the team. All images execute the `form team` statement as a collective operation.

An image changes teams by executing `change team` or `end team`. The former moves the executing image form the current team to a team specified by a derived-type `team_type` variable. Subsequent execution of a corresponding `end team` statement restores the current team back to that team to which it belonged immediately prior to execution of the most recent `change team`.

The `form team` statement takes an `team_number` argument uniquely identifying the team and a `team_type` argument encapsulating other team information in private components. Successful execution of a `form team` statement assigns the team-variable (of type `team type`) on each participating image a value that specifies the new team to which the image will belong. The `change team` statement takes as argument a `team type` variable that represents the new team to be used as current team. The execution of the `end team` statement restores the current team back to that immediately prior to execution of the `change team` statement. Figures 1-2 demonstrate the use of teams in two unit tests from the OpenCoarrays repository.

```
module assertions_module!Terminate globally if test fails
 implicit none; contains
 elemental subroutine assert(assertion,description)
   logical, intent(in) :: assertion
   character(len=*), intent(in), optional :: description
   integer, parameter :: max_digits=12
   character(len=max_digits) :: image_number
   if (.not.assertion) then
    write(image_number,*) this_image()
    error stop description//" failed on "//image_number
   end if
 end subroutine
end module

program main !! Test team_number intrinsic function
 use iso_fortran_env, only : team_type
 use assertions_module, only : assertions
 implicit none
 integer, parameter :: standard_initial_value=-1
 type(team_type), target :: home
 call assert(team_number()==standard_initial_value)
 associate(my_team=>mod(this_image(),2)+1)
  form team(my_team,home)!Map even|odd images->teams 1|2
  change team(home)
    call assert(team_number()==my_team,"correct mapping")
  end team
  call assert(team_number()==standard_initial_value)
 end associate
 sync all; if (this_image()==1) print *,"Test passed."
end program
```

Figure 1: A unit test for the team-number function.

2.2 Teams in GCC and OpenCoarrays

A Fortran team is comparable to an MPI communicator. OpenCoarrays uses `MPI_Comm_split` to support `form team`, passing the team id as the `color`, and storing the resulting communicator in an available-teams list. Every list element tracks the team/communicator pairings. The function returns the available-teams list stored in the `team type` variable.

We initialize the available- and used-teams lists to 1 (equivalent to `MPI_COMM_WORLD`) at the beginning of the execution. At a `change team`, the available-teams list element stored in the `team_type` variable gets passed to the corresponding OpenCoarrays function. The `current_team` variable used inside OpenCoarrays for representing the current communicator gets reassigned with the value contained into the element of the list passed as argument. Finally, a new element is added to the list of used teams. The list elements are pointers to the elements of the available-teams list. The insertion operation is always performed at the beginning of the list in order to keep track of the teams hierarchy. An execution of the `end team` statements is implemented by removing the first element of the list of used teams and reassigning the `current team` to the new first element of the list of used teams.

[2]https://github.com/sourceryinstitute/gcc
[3]https://github.com/sourceryinstitute/opencoarrays

```
1  program main  !! Test get_communicator language extension
2    use opencoarrays, only : get_communicator
3    use assertions_module, only : assert
4    use iso_fortran_env, only : team_type
5    type(team_type) :: league
6    integer, parameter :: num_teams=2 !! number of child teams to form
7    implicit none
8    call mpi_matches_caf(get_communicator()) !! verify rank & image numbering
9    associate(initial_image=>this_image(), initial_num_images=>num_images(), new_team=>mod(this_image()+1,num_teams)+1)
10     form team(new_team,league) !! create mapping
11     change team(league) !! join child team
12       call mpi_matches_caf(get_communicator()) !! verify new rank/image numbers
13       associate(my_team=>team_number())
14         call assert(my_team==new_team,"assigned team matches chosen team")
15         associate(new_num_images=>initial_num_images/num_teams+merge(1,0,my_team<=mod(initial_num_images,num_teams)))
16          call assert(num_images()==new_num_images,"block distribution of images")
17       end associate; end associate
18     end team
19     call assert( [initial_image==this_image(),initial_num_images==num_images()],"correct rank/image remapping")
20   end associate
21   sync all; if (this_image()==1) print *,"Test passed."
22 contains
23   subroutine mpi_matches_caf(comm) !! verify num. ranks =  num. images & image num. = rank num. + 1
24     use iso_c_binding, only : c_int
25     use mpi, only : MPI_COMM_SIZE, MPI_COMM_RANK
26     integer(c_int), intent(in) :: comm  !! MPI communicator
27     integer(c_int) :: isize,ierror,irank
28     call MPI_COMM_SIZE(comm, isize, ierror)
29     call assert( [ierror==0, isize==num_images()] ,"correct rank/image cardinality" )
30     call MPI_COMM_RANK(comm, irank, ierror)
31     call assert( [ierror==0, irank==this_image()-1,"correct rank/image numbering correspondence" )
32   end subroutine
33 end program
```

Figure 2: A unit test for the get_communicator function.

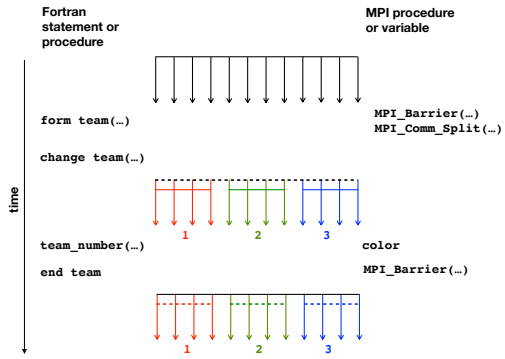

Figure 3: Schematic of program execution over time (left axis) in 12 images (top) communicating globally and then within subgroups. Horizontal lines show communication mechanisms (default=solid, optional=dashed). Fortran concepts (left). Underlying MPI concepts (right).

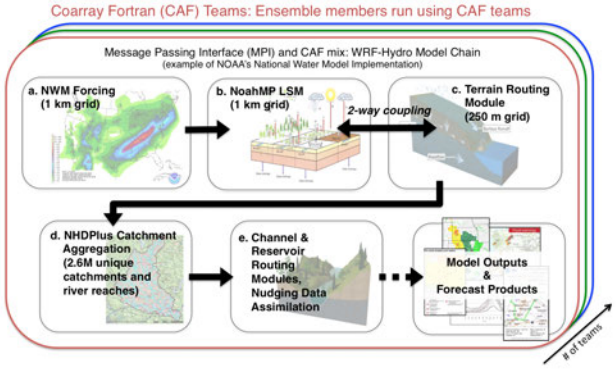

Figure 4: WRF-Hydro caffeination via Fortran 2018 teams: example components of the National Water Model. Different MPI colors represent independent teams, each of which is an ensemble member.

Figure 3 depicts schematically an initial team of images (black arrows) executing over time (progressing downward) and able to coordinate and communicate through a global mechanism (black horizontal line). At the point of executing `form team` and `change team` statements, the compiler inserts references to the OpenCoarrays ABI into the executable program. Those references cause invocations of `MPI_Split`, which in turn creates the colored groupings that correspond to teams in Fortran 2018.

The teams unit tests in Figures 1–2 use a block distribution of images, dividing the initial team into three new teams, each with the same number of images except some teams with one extra. The number of teams with an extra image equals the remainder of integer division of the total number images by the number of teams. In Figure 2, an assertion procedure terminates across all images if `assertion` is false. The optional second argument in `assert` describes the checks performed.

2.3 A language extension

Line 2 in Figure 2 imports a `get_communicator()` function via Fortran's use-association mechanism for accesing entities in Fortran modules: an `opencoarrays` module that provides language

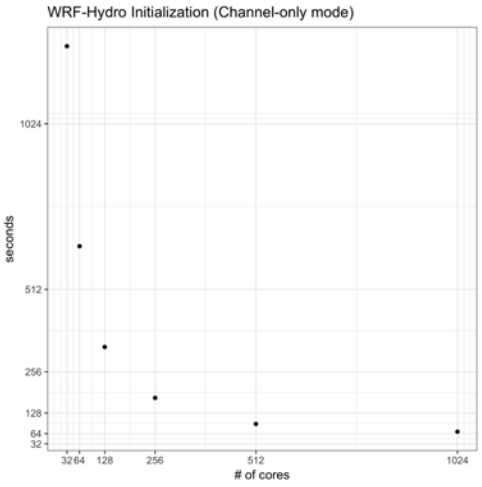

Figure 5: Scaling of the channel-only initialization time to support Amdhal's law calculation of potential speedup of ensemble computation with shared initialization.

extensions. Lines 8 and 12 invoke this function to provide the MPI communicator to the test subroutine `mpi_mpatches_caf`. Assertions in the latter subroutine verify the expected correspondences between MPI image and rank numbering. The MPI/CAF correspondence enables the newly caffeinated WRF-Hydro to interoperate safely with the existing WRF-Hydro MPI code.

3 DISCUSSION OF RESULTS

WRF-Hydro is a community hydrologic model providing a parallel-computing framework for coupling weather prediction (Figure 4a), land surface (Figure 4b), and hydrologic routing to handle spatial water redistribution via overland flow (Figure 4c), subsurface (soil column) flow (Figure 4c), baseflow (deep groundwater, 4d), and stream channel transport (Figure 4e) [3].

Developed to couple land hydrology to the atmosphere, WRF-Hydro usually runs "offline" with forcing from upper-boundary (weather) conditions (Fig. 4). The chief example is the National Weather Model (NWM) of the National Oceanic and Atmospheric Administration (NOAA) [6], a special configuration of WRF-Hydro providing operational, real-time analysis and forecasts over the U.S.

Ensemble forecasting and ensemble data assimilation are growing research areas for WRF-Hydro. Running ensembles under one executable program and job submission as shown in Fig. 4 can reduce the labor in workflow design and can open up possibilities for improving data flows and optimizing ensemble execution time.

An optimization that we expect teams to facilitate will involve sharing initialization work over all ensemble members using the full image set before changing teams. Amortizing common initialization work across all available computational resources rather than repeating common initialization work in each ensemble member using only that member's fraction of the resources might save significant computational costs in runs for which initialization occupies a large fraction of the run time such as in production runs of short-term NWM forecasts. We are exploring moving from deterministic (single ensemble member) runs to ensemble runs for the stream channel submodel in isolation. Fig. 5 presents the scaling behavior of channel-only runs. Running 50 ensemble members concurrently in teams occupying 20 cores each, the initialization speedup will be approximately 20x. Currently, for the short-range forecasts, initialization requires approximately 50% of the run time on NCAR's yellowstone computer and greater than that on NOAA WCOSS system [8]. These numbers are similar for the channel-only model. In our proposed application for an ensemble size of 50, we estimate that approximately half the runtime ($f = .5$) can be sped up by a factor of about 20 ($S_f = 20$), based on Fig. 5. From these estimates, Amdhalś law yields a total speedup of 1.9:

$$S = 1/f/S_f + (1 - f) = 1/(.5/20 + (1 - .5)) = 1.9 \qquad (1)$$

We converted the WRF-Hydro main program to a subroutine called in a loop over the ensemble members. We deleted `MPI_Init` and `MPI_FInalize` calls in the converted subroutine and instead pass a communicator to the subroutine. To loop over the ensemble also required deleting the `stop` statement in the subroutine.

4 CONCLUSIONS AND FUTURE WORK

We applied Fortran 2018 teams to WRF-Hydro ensemble simulation and forecasting. We developed the first-ever compiler front-end and parallel runtime library support for teams and a language extension that exposes CAF's underlying MPI communicator for use in WRF-Hydro. This approach facilitates incremental introduction of CAF, i.e.,"caffeination," of the pre-existing MPI program. We predict a speedup of 1.9 in ensemble execution when spreading common initializations across all images prior to changing teams. Such a speedup would grealy impact WRF-Hydro operational use cases.

ACKNOWLEDGMENTS

The first author thanks the NCAR Visitor Program for supporting a visit to NCAR to perform much of the work described herein.

REFERENCES

[1] 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.

[2] 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.

[3] D Gochis, W Yu, and D Yates. 2014. The WRF-Hydro model technical description and users guide, version 2.0. *NCAR Technical Documentation* (2014), 1–120.

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

[5] 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.

[6] National Oceanic and Atmospheric Administration (NOAA). 2016. The National Water Model. *Office of Water Prediction* (2016). http://water.noaa.gov/about/nwm

[7] 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.

[8] W Yu, D Gochis, D Yates, A Dugger, K Sampson, J McCreight, L Pan, Y Wu, B Cosgrove, Z Cui, C Pham, and G Sood. 2017. Development and Implementation of the NOAA National Water Model Using High Performance Computing Resource on the NSF/NCAR and NOAA Supercomputing Systems. (2017). https://ams.confex.com/ams/97Annual/webprogram/Paper316318.html