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
Natl. Ctr. for Atmospheric Research
Boulder, Colorado
jamesmcc@ucar.edu

Alessandro Fanfarillo
Natl. Ctr. for Atmospheric Research
Boulder, Colorado
elfanfa@ucar.edu

ABSTRACT

We present the use of Fortran 2018 teams (grouped processes) running an parallel ensemble of simulations built around a pre-existing Message Passing Interface (MPI) application. A challenge arises around the Fortran standard's eschewing any direct reference to lower-level communication substrate, such as MPI, leaving any interoperability between Fortran's intrinsic parallel programming model and the supporting substrate to the quality of the Fortran compiler implmentation. We demonstrate an approach for incrementally introducing Fortran's intrinsic parallel programming model, Coarray Fortran (CAF), a process we term "caffeination." Our approach puts CAF in charge of program initiation and exposes the underlying CAF MPI communicator to the original application code, resulting in a one-to-one correspondence between MPI group colors and Fortran teams. We demonstrate our approach using the Weather Research and Forecasting Hydrological Model (WRF-Hydro) developed at the National Center for Atmospheric Research (NCAR). 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 necessary 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 PGAS Applications Workshop, Denver, Colorado USA, November 2017 (PAW17),* 5 pages.
https://doi.org/xx.xxx/xxx_x

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

2 METHODOLOGY

Fortran 2008 does not provide for an environment in which a subset of the images can easily work on part of an application without affecting images outside the subset. Grouping the images of a program into nonoverlapping teams allows for more efficient and independent execution of parts of a larger problem. Multiphysics codes such as climate and weather models may benefit from the consequent reduction in off-node communication, particularly when an entire team of images fits within 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.

During execution, each image has a current team, which is only changed by execution of `change team` and `end team` statements. Executing `change team` moves the executing image form the current team to a team specified by a derived-type `team_type` variable.

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

```
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 "//
     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.

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.

2.2 Teams in GCC and OpenCoarrays

From an MPI perspective, a Fortran team is comparable to an MPI communicator. The `form team` statement is implemented in OpenCoarrays using `MPI_Comm_split` and passing the team id argument as `color` for `MPI_Comm_split`. In case of success, the resulting communicator is stored into a list of available teams. Every element of the list keeps track of the association of team identifier and a MPI communicator. The elements of the list of available teams gets returned by the function and stored into the `team type` variable.

The list of available and used teams are both initialized to team 1 (equivalent to `MPI_COMM_WORLD`) at the beginning of the execution. When the `change team` statement gets invoked, the element of the list of available teams stored into the `team_type` variable gets

```
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(), destination_team=>mod(this_image()+1,
       num_teams)+1)
10     form team(destination_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==destination_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.

passed as argument to the correspondent 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. This list of element is nothing but a list of pointers to the elements of the list of available teams. 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.

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 a number of teams with one extra. The number of teams with one extra equals the remainder of integer division of the total number images by the number of teams. Figure 2 contains an assertion utility that executes global error termination across all images if the logical expression passed as its first argument is false. The optional second argument in some

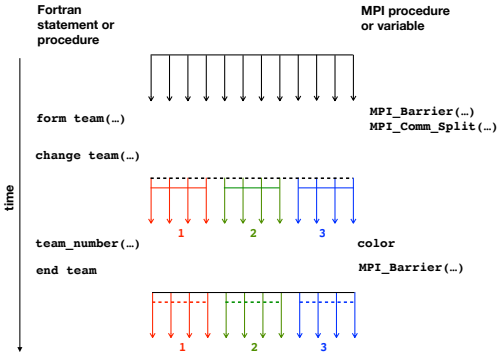

Figure 3: Schematic of a program executing over time (left axis) in 12 images (top) communicating globally (black horizontal bar) and later within subgroups (colored horizontal bars). Horizontal lines represent the communication mechanisms (default=solid, optional=dashed). Fortran concepts are on the left. Underlying MPI concepts are on the right.

`assert` subroutine invocations in Figures 1–2 describes the checks performed for the teams features discussed in this paper.

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

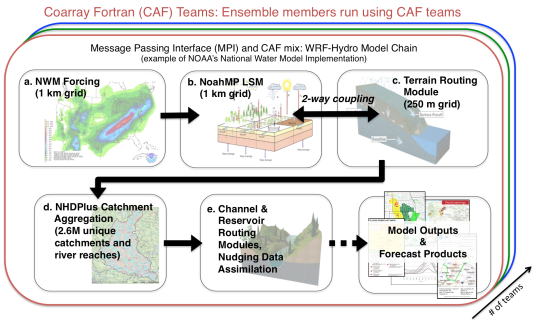

Figure 4: Caffeination of NCAR's WRF-Hydro model using Fortran 2018 teams. Example components of NOAA's National Water Model are shown. Different MPI colors represent independent teams, each of which is one member of an ensemble of simulations. This approach supports introducting CAF incrementally into the WRF-Hydro Model Chain

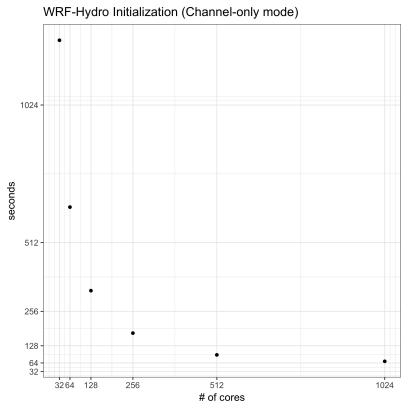

Figure 5: Scaling of the channel-only initialization time to support Amdhal's law calculation of potential speedup of ensemble computation with shared initialization.

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 modeling system that provides a parallel-computing framework for coupling Numerical Weather Prediction models (Figure 4a), land surface models (Figure 4b), and a suite of hydrologic routing modules that handle spatial water redistribution via surface (overland) flow (Figure 4c), subsurface (soil column) flow (Figure 4c), baseflow (deep groundwater, 4d), and stream channel transport (Figure 4e) [3].

While originally developed to couple land hydrology to the atmospheric processes of the WRF model, WRF-Hydro is most commonly run in "offline" mode where it is one-way coupled to (forced by) prescribed upper boundary (weather) conditions (4). The chief example is NOAA's operational National Water Model (NWM) [6]; a special configuration of WRF-Hydro which provides real-time analysis and forecasts of hydrologic states over the contiguous U.S (4). Here we

Ensemble forecasting and ensemble data assimilation are active and growing areas of research with WRF-Hydro. Running ensembles under a single executable and job submission on HPC platforms as shown in 4 can greatly reduce the amount of labor involved in designing workflows and can open up new possibilities for improving data flows and optimizing ensemble run performance.

We will investigate restructuring the code to share model initialization over all ensemble members using the full number of images prior to forming teams. Amortizing common model initialization across all available computational resources rather than repeating common aspects of the initialization in each ensemble member using only that member's fraction of the resources has the potential to provide notable computational savings for ensemble model runs where initialization comprises a large fraction of the total run time. Such an example is in the production run of the National Water Model short-term forecasts. Here we are currently researching the possibility of moving from the current deterministic (single ensemble member) model run to an ensemble run for the stream channel submodel run in isolation. Scaling of the initialization of the channel-only model is presented in 5. If 50 ensemble members are run concurrently with 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's 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 **??**. From these estimates, Amdhalś law yields a total speedup of 1.9:

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

Obtaining speed up of 1.9 when running ensembles by using CAF teams and restructuring the initialization would be welcome news in the operational (NWM) setting where it would help either reduce forecast latency or allow reduction of 247 dedicated resources.

In the application of teams to WRF Hydro, the main WRF-Hydro program is wrapped in a loop over the individual ensemble members. The number of images per team and the number of ensemble members are specified in a namelist. Along with the number of available images, these parameters determine how many teams are formed and which teams handle which ensemble members. It is worth noting several details of separate runs which may potentially need altering when running ensembles via teams: 1) `MPI_INIT`, 2) `MPI_FINALIZE`, 3) variable initialization and allocation, 4) `STOP`. We have already identified the potential for shared model initialization prior to the forming of teams for the embarrassingly parallel calculations. We expect other design choices which surface with further experience programming under the teams functionality.

4 CONCLUSIONS AND FUTURE WORK

We applied Fortran 2018 to ensemble simulation and forecasting with the WRF-Hydro model. We implemented the first-ever compiler front-end and parallel runtime library support for teams, obviating the need for modifying MPI code in WRF-Hydro. To combine CAF and MPI harmoniously, we developed a language extension that exposes CAF's underlying MPI communicator for use in WRF-Hydro. This approach facilitates optional, incremental introduciton of CAF, i.e,"caffeination" of the, pre-existing MPI program. The application revealed opportunities for further program optimizations supporting by the newly caffeinated infrustructure: namely, we predict a 1.9 speedup in the ensemble execution when common model initializations can be performed across all images prior to defining teams. A speedup of this magnitude would have a welcome and signficant impact on operational uses of WRF-Hydro.

ACKNOWLEDGMENTS

The first author thanks the Visitor Programs of the Computational Information Systems Laboratory and the Research Applications Laboratory of NCAR for travel support for a visit during which much of the work presented in this paper was performed.

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