Fortran 2015 and Coarrays in GNU Fortran

Salvatore Filippone

Cranfield University
salvatore.filippone@cranfield.ac.uk

Fortran Specialist Group Annual General Meeting 2017

Outline

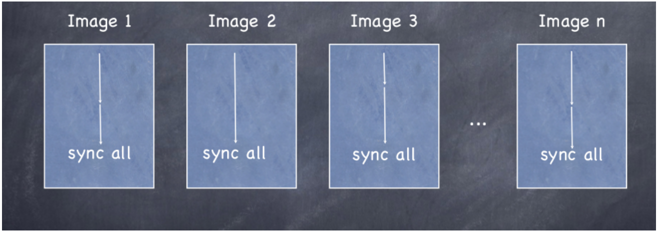
- Introduction: GNU Fortran and the OpenCoarrays project;
- Implementation status;
- Transport layers: performance results;
- Example applications: Load Balancing on Phi, Linear solvers, Climate Modeling;
- C749 considered harmful.

Credits

- Damian Rouson, Sourcery Institute, Berkeley (CA)
- Zaak Beekman, Princeton Univ. (NJ)
- Alessandro Fanfarillo, Dan Nagle, NCAR, Boulder (CO)
- Ambra Abdullahi, Valeria Cardellini, Univ. Rome “Tor Vergata” (IT)
- Salvatore Filippone, Soren Rasmussen, Cranfield University (UK)

Coarray basics

There are a bunch of copies of the program called `images`, and they perform their own computations until reaching an explicit or implicit synchronization point



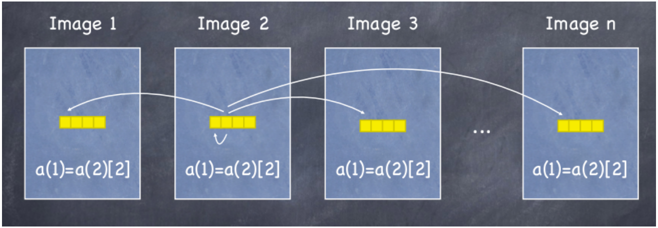
The sync points are:

```
sync all
sync images
allocate
```

TS18508 introduces `EVENT` facility for more sophisticated sync strategies, as well as `ATOMIC`s and collective communications.

Coarray variables

Each image has its data, but some data can be accessed remotely. Variable access is extended with the square brackets []



The index in the square bracket refers to an image index, running from 1 to *n*; it is a visual clue as to where communication happens.
Rules:

- The square brackets can be on either left or right hand side of an assignment;
- If dropped, the local image is intended;
- However, it is legal to address explicitly the local image

OpenCoarrays

`http://www.opencoarrays.org/`

OpenCoarrays

Provides an interface through which the compiler interacts with any one of several communication layers (e.g. MPI, OpenSHMEM)

Composed by three parts:

- **Compiler wrapper**: "caf" provide appropriate arguments to swap layers transparently;
- **Run-time library**: supports compiler communication and synchronization requests by invoking a lower level communication library (MPI by default).
- **Executable file launcher**: unified program launcher "cafrun" (again minimizing impact of layer swap).

GNU Fortran and OpenCoarrays

http://www.opencoarrays.org/

OPENCOARRAYS

HOME // ACTIVITIES // PUBLICATIONS // ABOUT US // MORE...

OpenCoarrays is an open-source software project that produces an application binary interface (ABI) used by the GNU Compiler Collection (GCC) Fortran front-end to build executable programs that leverage the parallel programming features of the draft Fortran 2015 standard.

DOWNLOADS

Installation via package management is generally the easiest and most reliable option. See below for the package-management installation options for Linux, macOS, and FreeBSD. Alternatively, download and build the latest OpenCoarrays release via the contained installation scripts or with CMake.

Linux

Linux users will find it easiest to install via the package managers linuxbrew, APT, or aur, or may use the pre-installed OpenCoarrays in HPCLinux. Please review the installation documentation Linux informatio

macOS

macOS users will find it easiest to install using the homebew package manager. Please review the installation documentation macOS information.

Windows

Windows users may install in the Windows Subsystem for Linux via the OpenCoarrays Windows installation script. Please see the installation documentation Windows information.

FreeBSD

FreeBSD users will find it easiest to install using the FreeBSD port. Please see the FreeBSD installation documentation.

Contributing: Potential contributors, please fork our git repository and submit a pull request with with any suggested changes.

COMPILERS

The GCC Fortran front-end (gfortran) v. 5.1 and later employ OpenCoarrays to support parallel execution.

OPEN CHAT

GNU Fortran and OpenCoarrays

Timeline

- Initial support for Coarrays in GNU Fortran 4.6 (2011): partial *single image*;
- Full *single image* and initial parallel support in GCC 4.7 (2012);
- Development stagnated until 2014 when OpenCoarrays project started;
- From version 5.1.0 includes OpenCoarrays support for transport layer;
- Many features of TS18508 (including collectives, atomics and EVENTs) covered in 6.1 (first compiler to do so! 2016);
- Support for FAILED IMAGES (initial in 7.1);
- Support for TEAMs: patch submitted for review on Sep.12th.

Transport layers

Translation from Fortran runtime API into an actual communication library:

- MPI (one-sided features of MPI ≥ 2, fault tolerance features under consideration for MPI 4);
- SHMEM
- GASNET

Typical coarray statement(s):

```
z(:)[dst]    = w(:)                    ! PUT
y(lb:ub:str) = x(lb:ub:str)[src]       ! GET
```

The PUT version allows for overlap

```
y(:)[dst]    = x(:)                    ! PUT
call foo_bar() ! This MAY overlap with communication
sync images(dst)
```

Transport layers

What do you need to use Coarrays effectively?

CoArray programmers (need to) embrace 1-sided communications; hence need for "Communication progress"

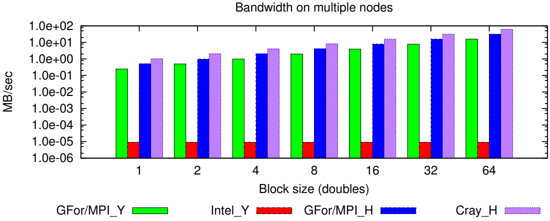
Common wisdom holds that `MPI_ISend` and `MPI_IRecv` achieve *overlap* between communications and computation.

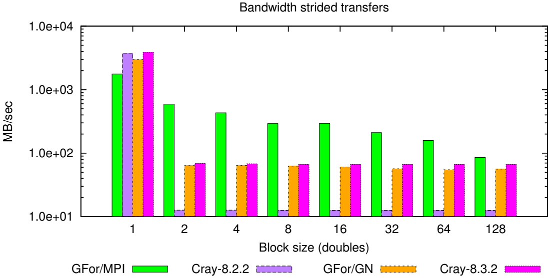
> *Unfortunately, the MPI standard allows for implementations to actually move the data only upon subsequent test/probe/wait calls*

And the implementation of "Communication progress" is entirely non-trivial

Cardellini V, Fanfarillo A, Filippone S, 2016 http://hdl.handle.net/2108/140530

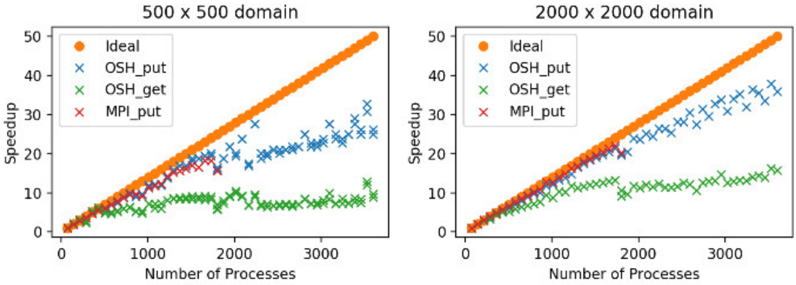
Performance data, take 1: strided copy





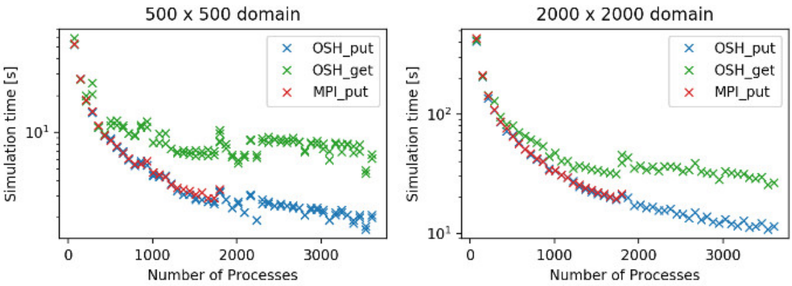
Fanfarillo, A., Filippone, S., Burnus, T., Nagle, D., Cardellini, V. and Rouson, D, PGAS 2014, Eugene, OR.

Climate modeling: MPI vs SHMEM



Rouson, D., Gutmann, E., Fanfarillo, A., and Friesen, B., PAW17, Denver, CO, Nov. 2017

Climate modeling: MPI vs SHMEM



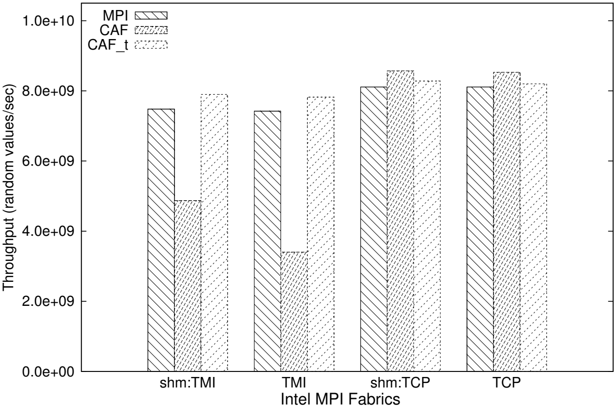
Rouson, D., Gutmann, E., Fanfarillo, A., and Friesen, B., PAW17, Denver, CO, Nov. 2017

Load balancing: Better MPI than MPI?

- Monte Carlo method for pricing Asian options (embarrassingly parallel algorithm).
- Original code taken from *Parallel programming and optimization with Intel Xeon Phi coprocessors**.
- Xeon Phis and CPUs used in *symmetric mode* (each device considered as a compute node).
- Approach presented by Colfax based on Master-Slave paradigm using MPI two-sided functions.
- **Proof of concept for dynamic load balancing on a single heterogeneous node.**
- Use the `ATOMIC_FETCH_ADD` intrinsic through the compiler wrapper

* Colfax International (http://www.colfax-intl.com/)

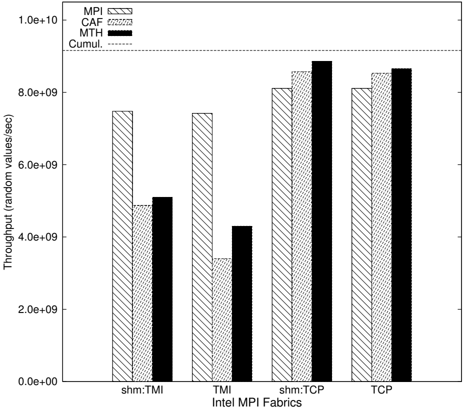
Load balancing: Better MPI than MPI?



Fortran code faster than MPI, despite using the same underlying Fortran compiler and MPI implementation!

Cardellini, V., Fanfarillo, A. and Filippone, S., Parallel Computing, 2017, to appear

Load balancing: Better MPI than MPI?



Cardellini, V., Fanfarillo, A. and Filippone, S., Parallel Computing 2017, to appear

Example applications

Krylov solvers: the essential step is the “halo exchange”, i.e. a variable and sparse all-to-all communication.

```
y(rcv_idxs(1:nrcv(img))) = x(rmt_idxs(1:nrcv(img)))[img]
y(rmt_idxs(1:nsnd(img)))[img] = x(snd_idxs(1:snd(img)))
```

This is quite difficult to translate into efficient code, so we tested multiple alternatives, some with a more “MPI-like” cooperative approach, also testing variants of EVENTS vs SYNC and PUT vs GET.
Besides, we ran against C749 (see later).

Example applications

Krylov solvers: time to prepare a preconditioner and time to apply the iteration to convergence

	MPI		CAF	
np	tprec	tsolve	tprec	tsolve
1	37.5	35.8	37.4	35.6
2	19.9	27.1	19.8	26.9
4	10.3	15.5	10.5	15.3
8	5.56	8.46	5.37	9.87
16	3.71	5.64	2.35	5.87
32	2.19	3.68	2.44	3.72
64	1.45	4.0	2.12	3.72

3D PDE problem, centered differences, strong scalability

Abdullahi Hassan, A., Cardellini, V., Filippone, S., PARCO 2017, Bologna, IT

Krylov solvers: events are not always faster overall

np	idim	MPI	CAF (sync images)	CAF (events)
1	250	0.64	0.90	0.89
2	350	0.99	1.03	1.0
4	500	1.37	1.58	1.33
8	700	2.00	2.20	3.88
16	1000	3.03	3.82	4.41
32	1400	5.07	5.36	6.10
64	2000	6.52	6.81	7.79

2D PDE problem, centered differences, weak scalability

Abdullahi Hassan, A., Cardellini, V., Filippone, S., PARCO 2017, Bologna, IT

Development directions

- Full support for transport layers: SHMEM, GASNET, etc.
- Full support for TEAMs and FAILED IMAGE
- Full integration in the GNU distribution machinery
- More in-depth analysis of performance issues

C824 considered harmful

C824

An entity whose type has a coarray ultimate component shall be a nonpointer nonallocatable scalar and shall not be a coarray.

What does it mean? (A plea to our standards officer)

The (size of the) set of entities that either are coarrays or contain coarray components must be fixed at compile time!!!!!!!!

```
type vector
  real, allocatable :: component(:),component_buffer(:)[:]
end type
type(vector) :: field ! Ok
type(vector), allocatable ::: bundle(:) ! Forbidden
type buffer_list_item
   real, allocatable :: buffer(:)[:]
   type(buffer_list_item), pointer :: next    ! Forbidden
end type
```

This means that in the midst of Fortran 2015 you are yanking the handbrake and reverting to FORTRAN 77 style!

Thank You!