PGAS Languages

Salvatore Filippone

salvatore.filippone@uniroma2.it

Partitioned Global Address Space (PGAs) Languages

Evolution of parallel progamming interfaces:

- Portions of the data space can be declared as *accessible*;
- Way to exploit RMA on modern hardware (at a small fraction of the associated headaches)

Currently in use:

1. (CoArray) Fortran;
2. Universal Parallel C;
3. UPC++.

We are currently exploring CoArray C++.

A PGAS Language: CoArray Fortran

The basic idea

CoArrays were invented by Bob Numrich of Cray to answer the following question:

What is the smallest syntax extension that can transform a serial Fortran program into a parallel program?

And the answer is:

The introduction of square brackets []

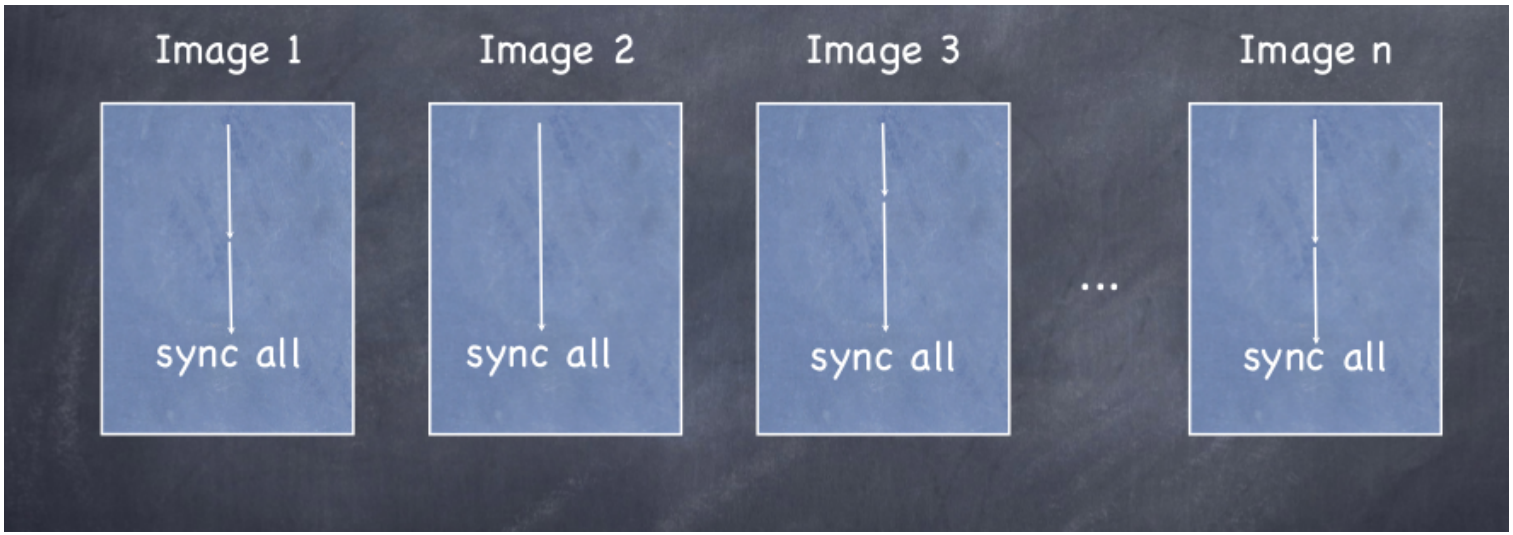
A PGAS Language: CoArray Fortran

Hello world

```
program helloCoarrays
  use iso_fortran_env
  implicit none
  integer, parameter :: MAX_STRING=100
  character(len=MAX_STRING) :: greeting[*] ! Scalar coarray
  integer image, num
  num=num_images()
  write(greeting,"(2(a,i2))") "Greetings from image ",this_ima
  sync all
  if (this_image()==1) then
    do image=1, num
      print *,greeting[image]
    end do
  end if
end program helloCoarrays
```

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

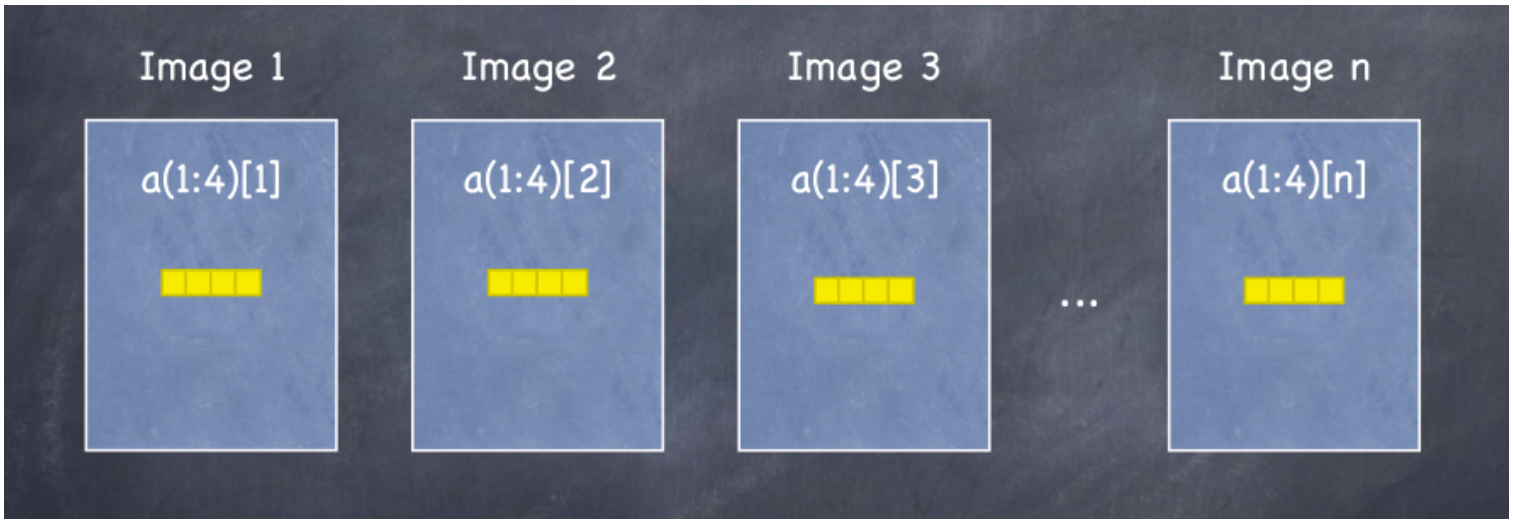


The sync points are:

```
sync all
sync images
allocate
```

Coarray variables

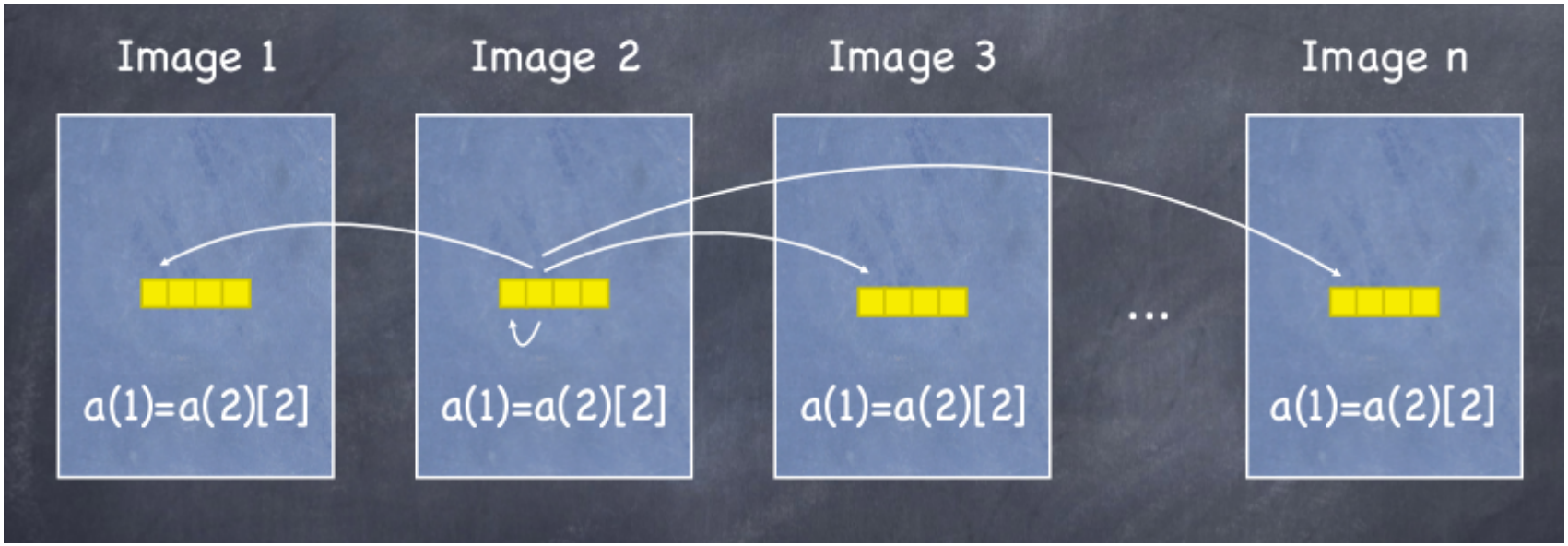
Variable access is extended with the square brackets []



The index in the square bracket refers to an image index, running from 1 to n.

With the square brackets you immediately have a visual clue as to where communication is required

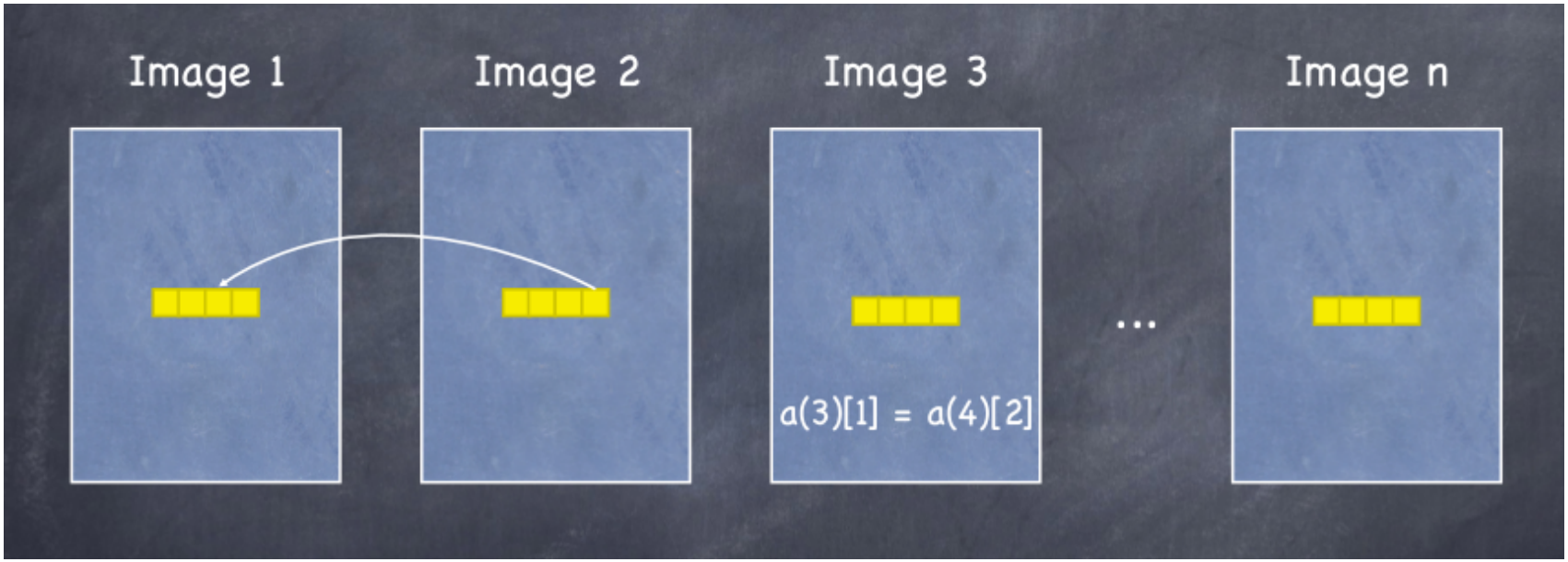
Coarray variables



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

Coarray variables



Try this with MPI!

The details

`sync all` Barrier for all images

`sync images ()` Barrier for subset of images

`this_image()` Who am I ?

`num_images()` How many of us are around?

Fine points

- The last dimension of a coarray must always be `*`;

```
real :: mat(:,:)[1,2,*]
```

- This applies also to allocatable coarrays

```
real, allocatable :: mat(:,:)[:]
allocate(mat(m,n)[*])
```

- The local size of an allocatable array coarray *must* be the same on all images

TEAMs, EVENTs and Collectives

In the Fortran 2018 standard we have a number of extensions.

- TEAM: A subset of the images that can operate independently of the others;
- EVENT: A facility for exchange of notifications among images;
- COLLECTIVE: As in MPI, functionality that computes a result in a coordinated fashion within a (sub)set of images.

Support under active development in the GNU Compiler Collection.

Collectives

- Each non-failed imaged in the current team contributes to performing the task;
- All collectives have optional `STAT` and `ERRMSG` arguments;
- All collectives have an argument `A` which is `INTENT(INOUT)` holding input and result;
- Implicit synchronization

Collectives

- CO_BROADCAST (A, SOURCE IMAGE [, STAT, ERRMSG])!
- CO_MAX (A [, RESULT IMAGE, STAT, ERRMSG])!
- CO_MIN (A [, RESULT IMAGE, STAT, ERRMSG])!
- CO_REDUCE (A, OPERATOR [, RESULT IMAGE, STAT, ERRMSG])!
- CO_SUM (A [, RESULT IMAGE, STAT, ERRMSG])!

```
! Copy the source image's local value to all other images
subroutine real_scalar_co_broadcast(a,source_image)
  real(real64), intent(inout) :: a[*]
  integer(int32), intent(in)  :: source_image

  sync all
  a = a[source_image]

end subroutine real_scalar_co_broadcast
```

Collectives

```
  subroutine real_scalar_co_broadcast(a)
    real(real64), intent(inout) :: a[*]
    integer(int32)  :: i, ni, ip
    integer(int32), allocatable :: k(:)
    i  = this_image()
    ni = num_images()
    ip=i/2
    if (ip>0) then
      sync images(ip)  ! Sync with my parent
      a = a[ip]
    end if
    allocate(k(0)) !
    if (2*i+1<=ni) k = [2*i+1, k]
    if (2*i<=ni) k = [2*i, k]

    sync images(k)  ! Sync with children if any

  end if
end subroutine real_scalar_co_broadcast
```

EVENTs

A variable of type EVENT_TYPE can be used to handle signaling between images.

```
use iso_fortran_env

real :: var[*], tmp
type(event_type) :: rcvready[*]
integer :: right,left
...
EVENT POST(rcvready[right])

EVENT WAIT(rcvready)

tmp = var[left]
```

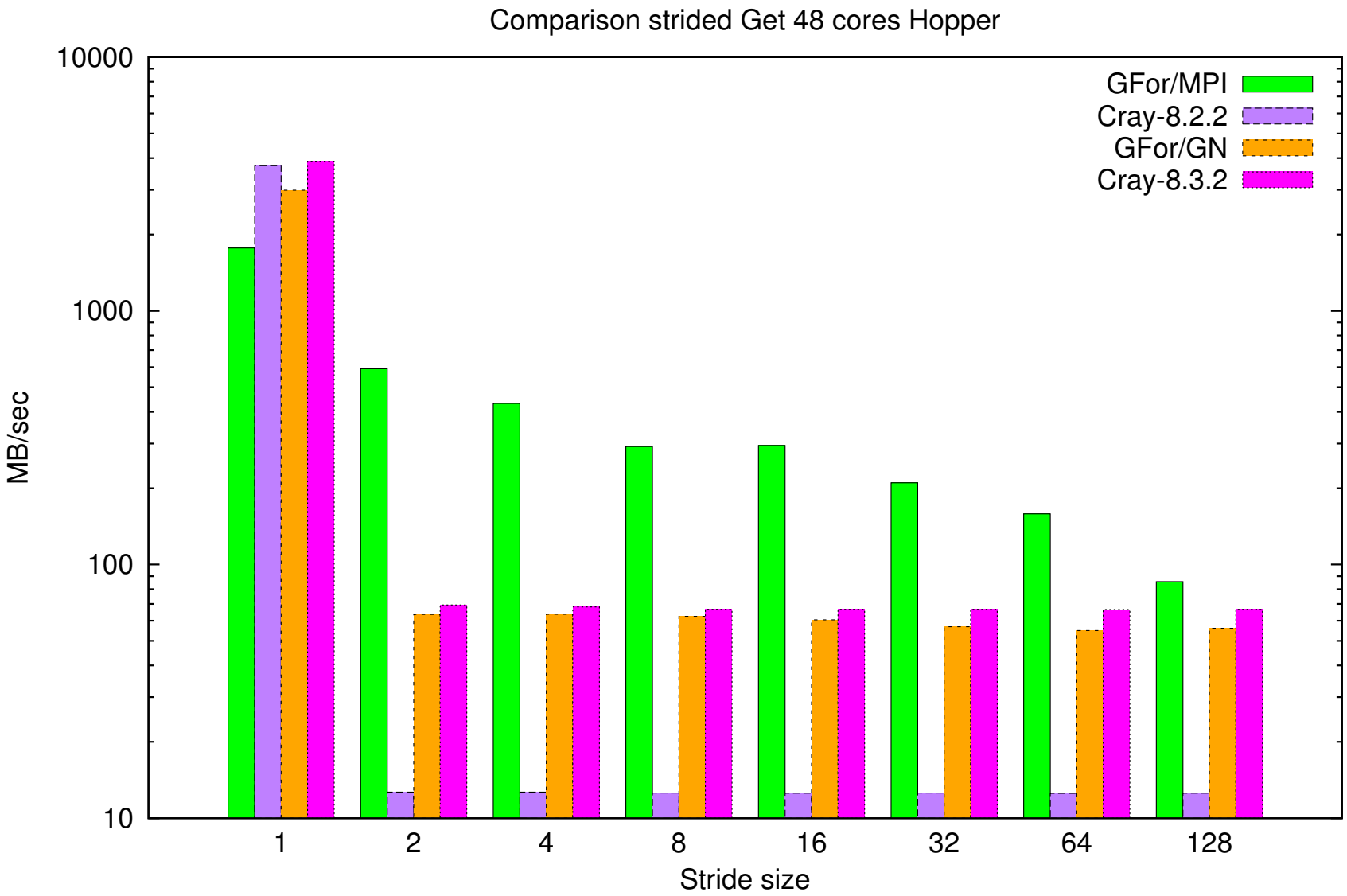
The OpenCoarrays Project

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

- Open-source project, providing communication layer for Fortran compilers;
- Used in the GNU Fortran project since version 5.1;
- Implementing basic and extended Coarray support
- Interface with other compilers
- High performance

The default implementation uses MPI; experimental implementations exist using other transport layers (e.g. GASNET).

```
x(1:n:2) = x(1:n:2)[4] ! Stride 2 GET
```



When do CoArrays work well?

The programming model implemented by CAF maps naturally onto MPI *one-sided* communications, exploting *RDMA remote direct memory access*.
To work properly:

- Should have hardware supported RDMA (many do);
- Should provide *overlap*;
- Should guarantee *progress*;

The first two characteristics are determined by the network hardware and system software layer; in particular, *overlap* is the ability of the combined network hardware/software to process data transfers without the direct involvement of the CPU.
Progress on the other hand is a feature of the MPI implementation.

When do CoArrays work well?

Progress on the other hand is a feature of the MPI implementation mandated by the MPI standard, however there are different implementations:

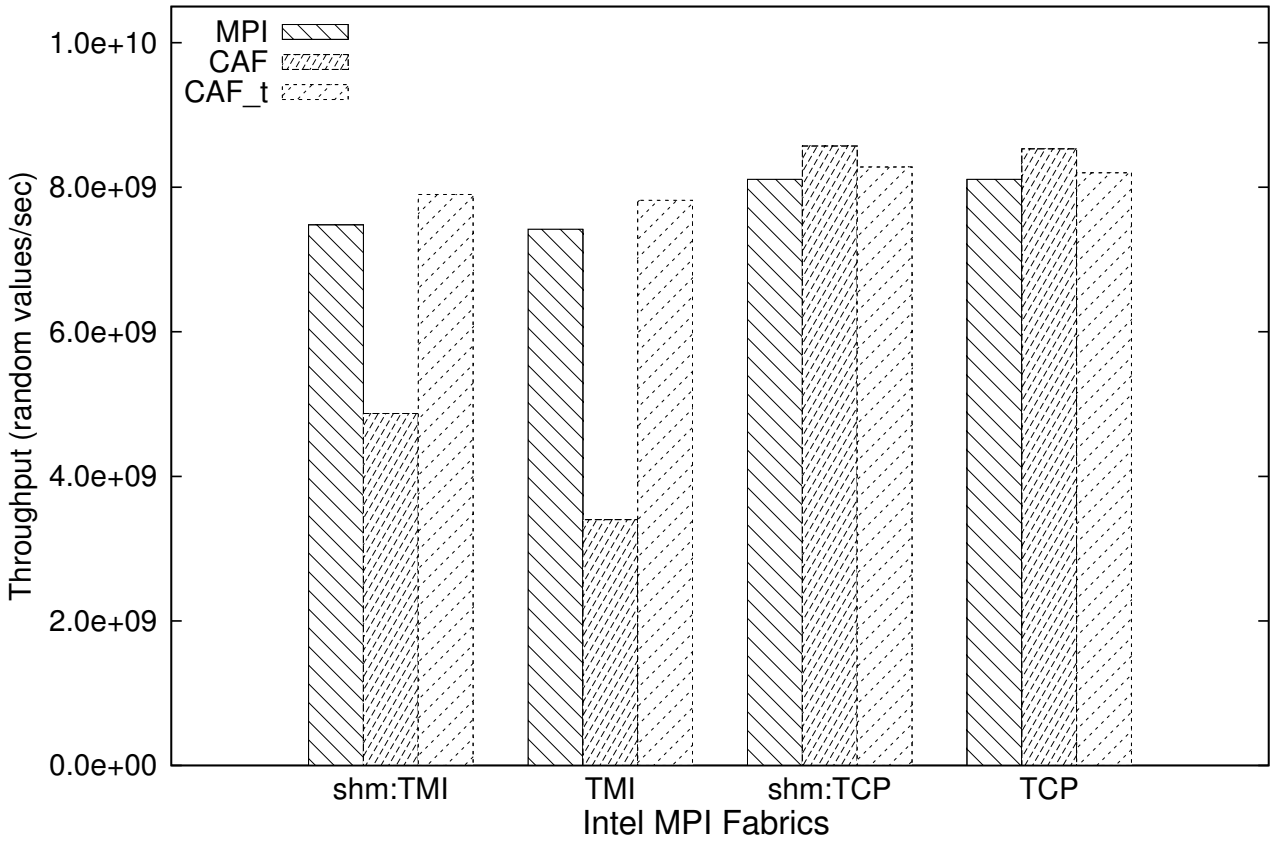
- Under a strict interpretation, after a non-blocking send (receive), once the matching receive (send) is posted, the send must progress even if the process does not make any further calls;
- Under a weak interpretation, actual progress happens only when the initiating process makes further library calls.

It is possible to have overlap without necessarily supporting progress; moreover, on the same hardware fabric, there may be multiple transport layers.
The most common way to guarantee progress is to *implement MPI using a service thread*.

An application

Treatment of Asian options on an Intel Xeon PHI using CAF and/or MPI.
Essential element:

Use `ATOMIC_FETCH_ADD` to grab the next chunk of work, achieving dynamic load balancing



References:

- Cardellini, V., Fanfarillo, A., and Filippone, S. (2016) "Overlapping Communication with Computation in MPI Applications", University of Rome "Tor Vergata", DICII, Technical Report RR-16.09, February.
- Cardellini, V., Fanfarillo, A., Filippone, S., and Rouson, D. (2015) "Hybrid coarrays: A PGAS feature for many-core architectures", Proceedings of the International Conference on Parallel Computing - ParCo2015, Edinburgh, UK, September 2015.
- Fanfarillo, A., Burnus, T., Filippone, S., Cardellini, V., Nagle, D., and Rouson, D (2014) "OpenCoarrays: open-source transport layers supporting coarray Fortran compilers", PGAS 2014, Eugene, Oregon, USA, Oct. 7-10, 2014.

Exercises

- Implementing your own collective;
- The Game of Life;
- Halo data exchange;