CAF Events Implementation Using MPI-3 Capabilities

Alessandro Fanfarillo[1], Jeff Hammond[2]

[1]National Center for Atmospheric Research, Boulder, CO, USA

[2]Intel Labs, Portland, OR, USA

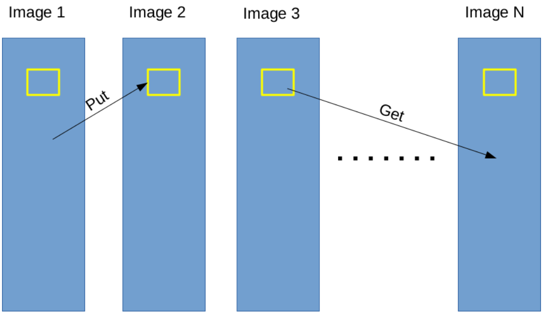
Sept. 28th, 2016

UCAR UNIVERSITY CORPORATION FOR ATMOSPHERIC RESEARCH

NCAR NATIONAL CENTER FOR ATMOSPHERIC RESEARCH

Coarray Fortran (CAF)

- Coarray Fortran (CAF) is a syntactic extension of Fortran 95/2003 and is now part of the Fortran 2008 standard.
- Fortran 2008 implements the Partitioned Global Address Space programming model.
- The PGAS model assumes a global memory address space that is logically partitioned and a portion of it is local to each process or thread.



CAF Example (Fortran 2008)

```
real, dimension(10) :: a
real, dimension(10), codimension[*] :: x, y
integer :: num_img, me

np = num_images();  me = this_image()

!Initialization of x and y

sync all

if(me /= np) x(2) = x(3)[np] ! get value from last image
if(me == np) y(1:5)[1] = y(6:10) ! put values on first image

if(me == np) then
  !update y
  sync images(1)
else if(me == 1) then
  sync images(np)
  y(:) = y(:)[np] ! get array from last image
endif
```

Coarray in Fortran 2015

Fortran 2008 does not allow to express more complex and useful mechanisms for synchronization, images organization and failure management.

Technical Specification 18508 proposes the following extensions to the coarray facilities defined in Fortran 2008:

- Teams
- Failed Images
- **Events**
- New atomic and collective procedures

Events

Events is how Fortran spells semaphores.

Events provide a fine grain synchronization mechanism based on a limited implementation of semaphores.

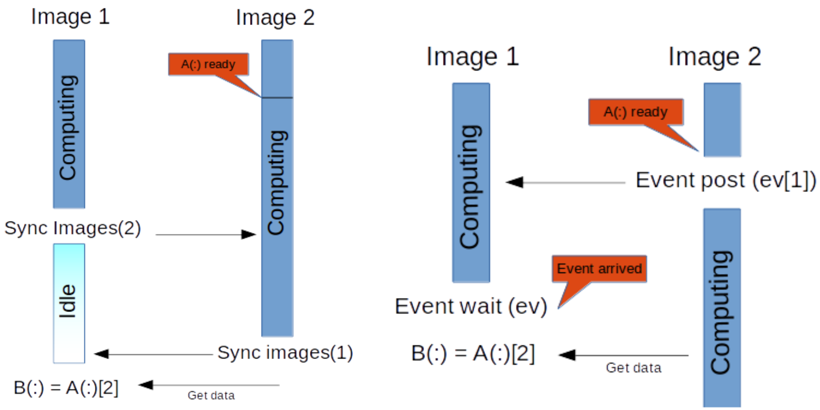
An event coarray variable is like a counter. It can be incremented by any image using the (non-blocking) statement `event post`.

An image can wait for the event variable to reach a certain amount of posted events using the `event wait` statement.

`event_query` is used to check, without blocking, the number of posted events on a local event.

Events and semaphores differ because of the local applicability of the `event wait` and `event_query`.

Events Example



PGAS on MPI-3

MPI currently incorporates direct remote memory access (RMA) through active and passive one-sided functions (good for PGAS).

Passive MPI one-sided functions may require the MPI implementation to be invoked on the target in order to *progress* the communication (no truly asynchronous communication).

Hardware support for some MPI RMA routines is rare (atomics).

MPI RMA implemented using a mix of hardware and software, at some loss of efficiency relative to a pure hardware implementation.

MPI implementations still do not provide high performing MPI RMA (in particular for MPI Atomics).

OpenCoarrays

- Since 2014 OpenCoarrays provides CAF support to the GNU Fortran Compiler (GCC 5.1+).
- Main version based on MPI-3. Experimental version based on GASNet.
- Currently provides almost complete support for all Fortran 2008 features, full support for Events, Atomics and Collectives procedures (experimental support for Failed Images).
- Competitive with commercial compilers (Intel, Cray).
- Freely available on GitHub and www.opencoarrays.org.

Events using MPI One-Sided Routines

Naive way: use atomic operations and spin-locks (from now on called RMA-events).

- Event is assumed to be a counter initialized to zero during the start-up phase.
- An invocation to `event post` is translated into an atomic increment of the target event variable (MPI_Accumulate).
- An invocation to `event wait` is translated into a spin-lock waiting for the counter (local event variable) to reach a predefined value (reset using `MPI_Fetch_and_op`).

Pros: Easy to implement and robust.

Cons: It relies completely on the performance of RMA atomic MPI operations. (high latency and low performance).

Events using MPI Two-Sided (P2P-Events)

MPI_Send is fast and MPI_Recv solves the MPI progress issue.

Asynchronous behavior on the sender side possible only with *eager protocol*.

The receiver must rely on the Unexpected Message Queue.

- Every event has a unique id.
- An invocation to event post is translated into a MPI_Send. The message contains the event id and the index of the event variable (in case of arrays).
- An invocation to event wait is translated into a loop with a MPI_Recv waiting for messages from MPI_ANY_SOURCE with a specific event tag. During the event wait the UMQ is shrunk.

Pros: Very fast and truly asynchronous under certain conditions.

Cons: Eager protocol might not be implemented, UMQ might not be used. Relies on theoretical unsafe mechanism.

MPI Tool Information Interface (MPI_T)

MPI 3 provides a standard way to access performance data and (interact with) control variables belonging to the MPI implementation.

A typical information that can be useful to know (used also in this work) is *how many messages are in the Unexpected Message Queue waiting to be received?*

Different MPI implementations expose different variables (they may represent the same concept with different names).

In this work we use MPI_T in order to make a run-time library more robust while using (theoretical) unsafe mechanisms.

OpenCoarrays may be tied to a specific MPI implementation.

The Idea

```
Initialize RMA-events and P2P-events variables;
if UMQ perf var AND eager control var then
    decide threshold for UMQ;
    select P2P-events;
else
    select RMA-events;
end
selection check;
while program not terminated do
    if P2P-events then
        if UMQ > threshold then
            broadcast switch to RMA;
            empty UMQ queue from events;
            select RMA-events;
        end
        if switch_to_RMA received then
            empty UMQ from events;
            select RMA-events;
        end
    end
end
```

Parallel Research Kernels and Experimental Platforms

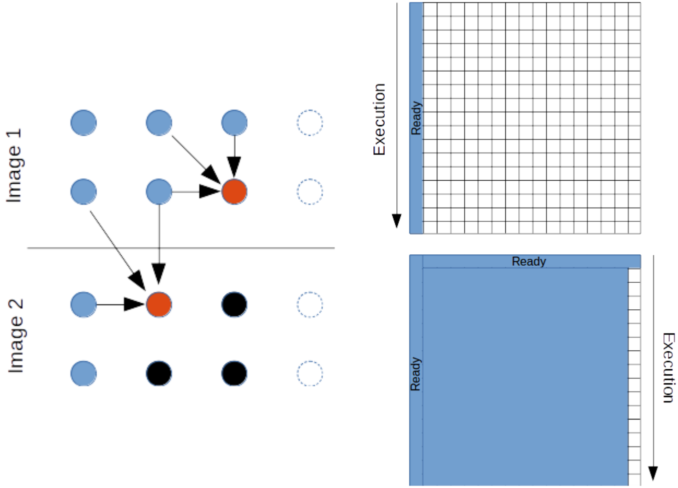
PRK provides a set of kernels that covers the most common patterns of communication, computation and synchronization encountered in parallel HPC applications.

- Sync_P2P represents a pipelined stencil computation, typical of numerical methods such as wavefront-parallel algorithms.
- Stencil applies a data-parallel stencil operation to a two-dimensional array. It represents one of the most common communication pattern in scientific computing: the halo exchange.

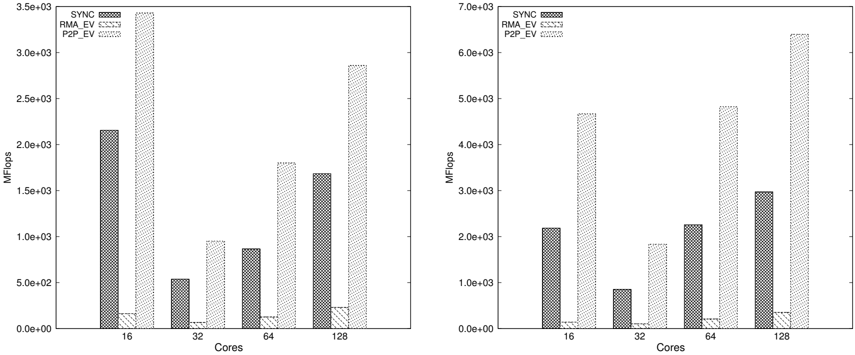
We run our tests on two supercomputers:

- Galileo (CINECA): Two Intel Xeon E5-2630v3 at 2.40 GHz on each node. Nodes interconnected with Intel True Scale QDR InfiniBand.
- Stampede (TACC): Two Intel Xeon E5-2680v3 at 2.7GHz on each node. Nodes interconnected with Mellanox FDR Infiniband.

Sync_P2P and Stencil kernels

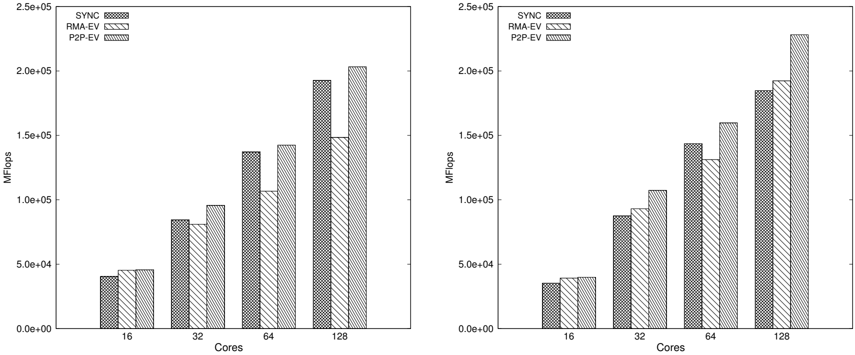


P2P-Sync Performance



Galileo (CINECA) - Stampede (TACC)

Stencil Performance



Galileo (CINECA) - Stampede (TACC)

Conclusions

- Events are capable of improving the performance of parallel applications by reducing the amount of idle time.
- Implementing events on top of MPI two-sided routines provides a performance improvement on every platform and test case.
- MPI_T allows to select the best algorithm based on the support provided by the MPI implementation.
- Besides being used by tuning and analysis tool, MPI_T can be a powerfull functionality also for run-time libraries.

Thanks