Coarray Fortran Runtime Library Interface Design Document Draft
Version 0.1

Damian Rouson, Brad Richardson, Dan Bonachea, Katherine Rasmussen, Hussain Kadhem
Lawrence Berkeley National Laboratory, USA
lbl-flang@lbl.gov

WORK IN PROGRESS: This is a very early draft that remains incomplete and will continue to evolve throughout the requirements-gathering process.

1 Problem description

In order to comply with Fortran 2018, Flang must support parallel programming features commonly referred to as “coarray fortran,” including the following entities:

- **Statements:**
  - Synchronization: `sync all`, `sync images`, `sync memory`, `sync team`
  - Events: `event post`, `event wait`
  - Error termination: `error stop`
  - Locks: `lock`, `unlock`
  - Failed images: `fail image`
  - Teams: `form team`, `change team`
  - Critical sections: `critical`, `end critical`

- **Intrinsic functions:** `num_images`, `this_image`, `lcobound`, `ucobound`, `team_number`, `get_team`, `failed_images`, `stopped_images`, `image_status`, `coshape`, `image_index`

- **Intrinsic subroutines:**
  - Collective subroutines: `co_sum`, `co_max`, `co_min`, `co_reduce`, `co_broadcast`
  - Atomic subroutines: `atomic_add`, `atomic_and`, `atomic_or`, `atomic_cas`, `atomic_define`, `atomic_fetch_add`, `atomic_fetch_and`, `atomic_fetch_or`, `atomic_fetch_xor`, `atomic_or`, `atomic_ref`, `atomic_xor`
  - Other subroutines: `event_query`

- **Types, kind type parameters, and values:**
  - Intrinsic derived types: `event_type`, `team_type`, `lock_type`
  - Atomic kind type parameters: `atomic_int_kind` and `atomic_logical_kind`
  - Values: `stat_failed_image`, `stat_locked`, `stat_locked_other_image`, `stat_stopped_image`, `stat_unlocked`, `stat_unlocked_failed_image`

In addition to being able to support syntax related to the above features, compilers will also need to be able to handle new execution concepts such as image control. The image control concept affects the behaviors of some statements that were introduced in Fortran expressly for supporting parallel programming, but image control also affects the behavior of some statements that pre-existed parallelism in standard Fortran:

- **Image control statements:**
  - Pre-existing statements: `allocate`, `deallocate`, `stop`, `end`, `a call referencing move_alloc with coarray arguments`
  - New statements: `sync all`, `sync images`, `sync memory`, `sync team`, `change team`, `end team`, `critical`, `end critical`, `event post`, `event wait`, `form team`, `lock`, `unlock`

One consequence of the statements being categorized as image control statements will be the need to restrict code movement by optimizing compilers.
2 Proposed solution

This design document proposes an interface to support the above features, named Coarray Fortran Parallel Runtime Interface. By defining a library-agnostic interface, we envision facilitating the development of alternative parallel runtime libraries that support the same interface. One benefit of this approach is the ability to vary the communication substrate. A central aim of this document is to use a parallel runtime interface in standard Fortran syntax, which enables us to leverage Fortran to succinctly express various properties of the procedure interfaces, including argument attributes. See Rouson and Bonachea (2022) for additional details.

2.1 Coarray Fortran (CAF) Parallel Runtime Interface

The Coarray Fortran Parallel Runtime Interface is a proposed interface in which the runtime library is responsible for coarray allocation, deallocation and accesses, image synchronization, atomic operations, events, and teams. In this interface, the compiler is responsible for transforming the source code to add Fortran procedure calls to the necessary runtime library procedures. Below you can find a table showing the delegation of tasks between the compiler and the runtime library. The interface is designed for portability across shared and distributed memory machines, different operating systems, and multiple architectures. As described in Section 2.4.1, the developers of the Caffeine implementation of the Coarray Fortran Parallel Runtime Interface plan to support the following architectures: x86_64, PowerPC64, AArch64, with the possibility of supporting more as requested. Implementations of this interface are intended to augment the compiler’s own parallel runtime library. While the interface can support multiple implementations, we envision needing to build the runtime library as part of installing the compiler.

2.2 Delegation of tasks between the Fortran compiler and the runtime library

The following table outlines which tasks will be the responsibility of the Fortran compiler and which tasks will be the responsibility of the runtime library. An “X” in the “Fortran compiler” column indicates that the compiler has the primary responsibility for that task. An “X” in the “Runtime library” column indicates that the compiler will invoke the runtime library to perform the task and the runtime library has primary responsibility for the task’s implementation. See the Runtime Interface Procedures for the list of runtime library procedures that the compiler will invoke.

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Fortran compiler</th>
<th>Runtime library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Establish and initialize static coarrays prior to main</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Track corank of coarrays</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Assigning variables of type team-type</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Track locals coarrays for implicit deallocation when exiting a scope</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Initialize a coarray with SOURCE= as part of allocate-stmt</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Track coarrays for implicit deallocation at end-team-stmt</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Allocate and deallocate a coarray</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Reference a coindexed-object</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Team stack abstraction</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>form-team-stmt, change-team-stmt, end-team-stmt</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Intrinsic functions related to Coarray Fortran, like num_images, etc</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Atomic subroutines</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Collective subroutines</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Synchronization statements</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Events</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Locks</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>critical-construct</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
2.3 Types

Provided Fortran types: caf_event_type, caf_team_type, caf_lock_type

Runtime library specific types: caf_co_handle_t, caf_async_handle_t, caf_source_loc_t

2.4 Runtime Interface Procedures

Collectives: caf_co_broadcast, caf_co_max, caf_co_min, caf_co_reduce, caf_co_sum

Program startup and shutdown: caf_init, caf_finalize, caf_error_stop, caf_stop, caf_fail_image

Allocation and deallocation: caf_allocate, caf_deallocate

Coarray Access: caf_put, caf_get, caf_get_async

Operation Synchronization: caf_async_wait_for, caf_async_try_for, caf_sync_memory

Image Synchronization: caf_sync_all, caf_sync_images, caf_lock, caf_unlock, caf_critical, caf_end_critical

Events: caf_event_post, caf_event_wait, caf_event_query

Teams: caf_change_team, caf_end_team, caf_form_team, caf_sync_team, caf_get_team, caf_team_number

Atomic Memory Operation: caf_atomic_add, caf_atomic_and, caf_atomic_cas, caf_atomic_define, caf_atomic_fetch_add, caf_atomic_fetch_and, caf_atomic_fetch_or, caf_atomic_fetch_xor, caf_atomic_or, caf_atomic_ref, caf_atomic_xor

Coarray Queries: caf_lcobound, caf_ucobound, caf_coshape, caf_image_index

Image Queries: caf_num_images, caf_this_image, caf_failed_images, caf_stopped_images, caf_image_status

2.4.1 Caffeine - LBL’s Implementation of the Coarray Fortran Parallel Runtime Interface

Implementations of some parts of the Coarray Fortran Parallel Runtime Interface exist in Caffeine, a parallel runtime library targeting coarray Fortran compilers. Caffeine will continue to be developed in order to fully implement the proposed Coarray Fortran Parallel Runtime Interface. Caffeine uses the GASNet-EX exascale networking middleware but with the library-agnostic interface and the ability to vary the communication substrate, it might also be possible to develop wrappers that would support the proposed interface with OpenCoarrays, which uses the Message Passing Interface (MPI).

2.5 Types Descriptions

2.5.1 Fortran Intrinsic Derived types

These types will be defined in the runtime library and it is proposed that the compiler will use a rename to use the runtime library definitions for these types in the compiler’s implementation of the ISO_Fortran_Env module.

caf_team_type: implementation for team_type from ISO_Fortran_Env
caf_event_type: implementation for event_type from ISO_Fortran_Env
caf_lock_type: implementation for lock_type from ISO_Fortran_Env

Acknowledgment

This research is supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.