

Lowering and Runtime Support for Fortran's Multi-Image Parallel Features using LLVM Flang, PRIF, and Caffeine

Dan Bonachea 
Katherine Rasmussen 
Damian Rouson 
fortran@lbl.gov
Lawrence Berkeley National
Laboratory
Berkeley, California, USA

Jean-Didier Pailleux 
Etienne Renault 
jean-didier.pailleux@sipearl.com
etienne.renault@sipearl.com
SiPearl, R&D
Maisons-Laffitte, France

Brad Richardson 
brad.richardson@nasa.gov
Amentum Services, Inc.
Houston, Texas, USA

ABSTRACT

This paper provides an overview of the multi-image parallel features in Fortran 2023 and their implementation in the LLVM Flang compiler and the Caffeine parallel runtime library. The features of interest support a Single-Program, Multiple-Data (SPMD) programming model based on executing multiple “images”, each of which is a program instance. The features also support a Partitioned Global Address Space (PGAS) in the form of “coarray” distributed data structures. The paper discusses the lowering of multi-image features to the Parallel Runtime Interface for Fortran (PRIF) and the implementation of PRIF in the Caffeine parallel runtime library. This paper also provides an early view into the design of a new multi-image dialect of the LLVM Multi-Level Intermediate Representation (MLIR). We describe validation and testing of the resulting software stack, and demonstrate that performance compares favorably to another open-source compiler and runtime library: GNU Compiler Collection (GCC) gfortran and OpenCoarrays, respectively.

CCS CONCEPTS

• **Software and its engineering** → **Runtime environments**; **Parallel programming languages**; • **Computing methodologies** → *Parallel programming languages*.

KEYWORDS

Fortran, Parallel programming, HPC, PGAS, RMA, LLVM Flang, Exascale Computing, Runtime Libraries, Caffeine, GASNet-EX

ACM Reference Format:

Dan Bonachea , Katherine Rasmussen , Damian Rouson , Jean-Didier Pailleux , Etienne Renault , and Brad Richardson . 2025. Lowering and Runtime Support for Fortran's Multi-Image Parallel Features using LLVM Flang, PRIF, and Caffeine. In *Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC Workshops '25)*, November 16–21, 2025, St Louis, MO, USA. ACM, New York, NY, USA, 9 pages. <https://doi.org/10.25344/S4G883>

1 INTRODUCTION

LLVM [26] provides a highly modular, extensible compilation pipeline, and its maturity for the C and C++ languages is well-established. Following a similar pattern, an effort has been made to focus the Fortran ecosystem [41] and to converge towards an LLVM-based solution, called LLVM Flang [24] (the Fortran front-end of LLVM, denoted flang in the rest of this paper). flang is under active development and contains robust support for many of the features in Fortran 2023 [15, 25]. However, the latest release (version 21) lacks support for the multi-image parallel features of Fortran¹. Adding support for these features is on the LLVM roadmap. In 2022, an initial series of patches were integrated to provide the front-end support for multi-image features, but lowering and runtime support remains a work-in-progress. As a consequence, while other prominent Fortran compilers (e.g., GNU gfortran, Intel ifort/ix, HPE crayftn) have production-ready support for a large portion of the multi-image features, this is still missing in flang.

Fortran plays important roles in fields ranging from weather [40] and climate [8] to nuclear energy [10], aerospace engineering [2], and fire protection engineering [27]. If you looked at a weather forecast today, received electricity from a power plant licensed by the U. S. Nuclear Regulatory Commission, rode in any one of numerous car or aircraft models, or live in one of 195 countries that signed the Paris climate accord, then Fortran codes impacted your life in one or more ways today even before you encountered this paper. Teams are also writing Fortran software in emerging disciplines such as deep learning [32] and to develop tools in areas where Fortran has not been typically employed, such as package management [16]. Fortran thus remains a key component in the High-Performance Computing (HPC) software ecosystem [1]. The multi-image features of Fortran allow developers of scientific software to fully utilize the capabilities of HPC systems while still expressing their algorithms in standard Fortran, and increased compiler support for these features will benefit the Fortran developer ecosystem.

LLVM-HPC'25, November 17, 2025, St Louis, MO, USA

© 2025 Copyright held by the owner/author(s).

This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC Workshops '25)*, November 16–21, 2025, St Louis, MO, USA, <https://doi.org/10.1145/3731599.3767480>.

¹We use the term multi-image in this paper to encapsulate coarray features (allocation, deallocation, accesses, ...) as well as all other multi-image parallel features (synchronization, collective subroutines, atomic subroutines, image teams, etc.).

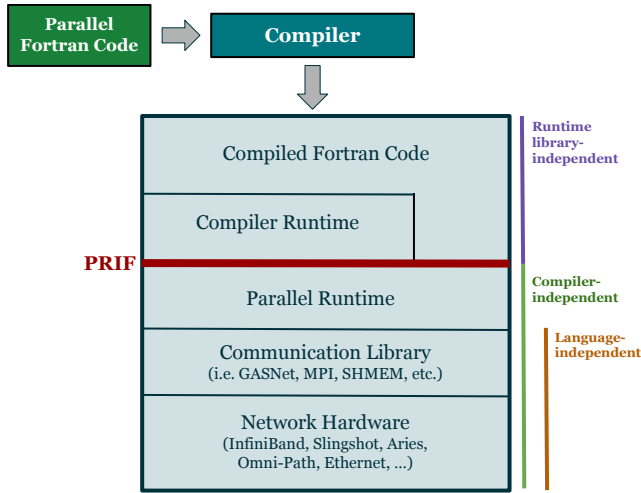


Figure 1: PRIF's role in a Parallel Fortran software stack

2 SOFTWARE ARCHITECTURE

2.1 PRIF

The Parallel Runtime Interface for Fortran (PRIF) [5] is a compiler- and runtime-library-agnostic interface, written in Fortran, to support the multi-image parallel features of Fortran. As seen in Figure 1, PRIF sits in between the compiler and the parallel runtime library. The PRIF specification [31] is an open standard, governed by a multi-institutional committee. In the aforementioned Fortran ecosystem where compiler support for the multi-image features is incomplete, PRIF aims to provide a generalized interface that can simplify adding this support to a compiler, as any compiler that lowers to PRIF calls can use any runtime library that follows the interface. It also adds flexibility to the Fortran ecosystem and produces the possibility that one PRIF runtime library could provide support for multiple compilers, or that one compiler could support several PRIF runtime libraries.

PRIF was developed to be portable across shared- and distributed-memory systems, with varying operating systems, toolchains and architectures. It achieves this by defining a set of Fortran procedure interfaces corresponding to each of the multi-image parallel features

defined in the Fortran standard that may be invoked by a Fortran compiler and implemented by a runtime library. The interface itself is expressed as a standard-compliant Fortran 2018 [14] module that deliberately avoids any compiler-specific details, in order to maximize portability.

2.2 Lowering in LLVM

In compiler terminology, “lowering” commonly describes the process of translating high-level language constructs into more concrete, lower-level constructs. In practice, this transformation can be broken down into several distinct stages: *parsing* (which converts source code into an Abstract Syntax Tree (AST)), *desugaring* (which simplifies the source language by removing syntactic sugar), *Intermediate Representation (IR) transformation* (which breaks down complex constructs into simpler, lower-level operations), *canonicalization* (which rewrites code into a standardized form to make analysis and optimization easier), *legalization* (which ensures that operations are valid for the target architecture), and *codegen* (which produces the architecture-specific machine code).

Although multi-image features (including coarrays) were introduced in the Fortran 2008 standard [13], LLVM has only supported basic parsing of coarray syntax since the early 2020s. In contrast, other compilers have offered much more complete support for multi-image features for several years: the Cray Compiling Environment (CCE) [18] since before 2008, OpenUH [11, 19] since 2010, Intel ifort/ix [21] since 2011, GCC gfortran [12] since 2015, and nAG [30] since 2022 (among others). Each of those compilers uses a dedicated, proprietary runtime (based on MPI [12], GASNet [17], or private implementations) to provide the communication support for multi-image features.

While LLVM has become the most dominant framework for compilers, the absence of robust multi-image support could discourage Fortran users. Consequently, this work aims to close this functionality gap between existing compilers and flang. Therefore, we opted to provide a *lowering step* (IR transformation pass) towards PRIF, a backend-agnostic abstraction layer. Figure 2 illustrates the overall compilation flow of flang and the work proposed in this paper (depicted by the violet box).

Our transformation pass is inserted during the conversion from High-level Fortran Intermediate Representation (HLFIR) to Fortran

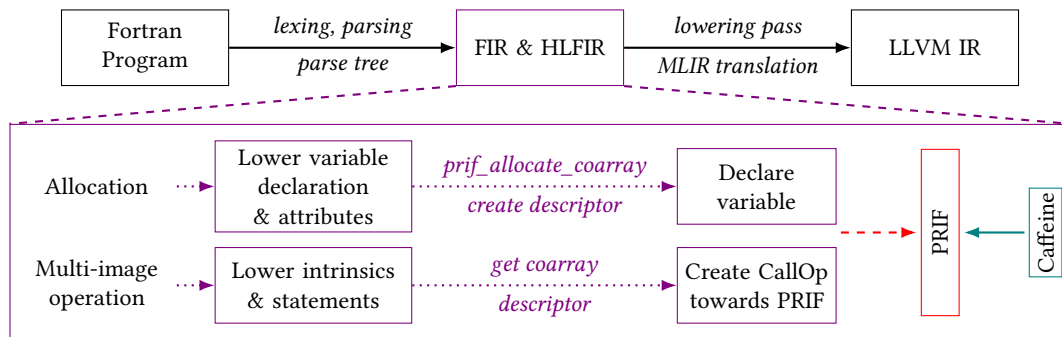


Figure 2: Proposed compilation flow to add support for Fortran's multi-image parallel features into LLVM Flang.

Listing 1: Minimal multi-Image example code

```

program main
  integer :: i
  if (this_image() .eq. 1) then
    i = num_images()
  endif
end program main

```

Intermediate Representation (FIR). **HLFIR** and **FIR** are two Fortran-specific dialects developed to represent Fortran constructs and concepts. Their supported features include variable allocation, variable declaration, and procedure calls. This is a natural point for the transformation, as it aligns directly with the point in the lowering pipeline where specific low-level Fortran constructs are introduced. Placing it here also makes future integration with other **MLIR** dialects more straightforward (see §4 for more details). Using the various operations and types of these dialects, we are able to generate calls to different procedures managed by an implementation of the **PRIF** runtime.

Our approach includes two major aspects:

- For allocations, an attribute has been introduced to the semantic symbol to tag all variables with coarray type. As a result, regardless of whether it is an explicit allocation or a static declaration, our pass is able to intercept the allocation and replace it with a call to 'prif_allocate_coarray'. This procedure handles the actual coarray allocation according to the library implementation and returns a descriptor handle (which is also implementation-specific) back to the LLVM-generated code. It should be noted that for variables that have been declared globally or for members of a derived type, a slightly more complicated process is performed. In this case, a global declaration at the **HLFIR** level (GlobalOp operation) of the descriptor reference is made.
- For coindexed expressions (coarray accesses) and/or intrinsics related to the multi-image parallel features (e.g. 'this_image', 'image_index', 'sync all', etc.), our flow translates Fortran multi-image operations into direct calls to the **PRIF** library. These calls are directly inserted at the **HLFIR/FIR** level. However, for calls involving a coarray, **PRIF** requires the caller to pass the coarray descriptor handle associated with the appropriate variable. As a consequence, some lookup has been implemented to retrieve it directly from the declaration of the coarray variable.

Listing 1 shows a minimal example of a multi-image Fortran program using the `this_image` and `num_images` intrinsic functions. The `this_image` function returns the 1-based index of the calling image and `num_images` returns the number of images in the SPMD parallel execution. When this program executes, the first image will define the variable `i` to the value of `num_images`. The relevant compilation output after lowering this program to HLFIR is shown in Listing 2, where we can observe the function calls (`fir.call`) to **PRIF** module procedures that `flang` has generated.

This approach has the advantage of being concise and positioned early in the compilation process, alongside other **FIR/HLFIR** translation phases. As a result, it allows subsequent (generic) optimizations

Listing 2: HLFIR output from compiling Listing 1 using our fork of flang

```

func.func @_QQmain() attributes {fir.bindc_name = "MAIN"} {
  ...
  %3 = fir.absent !fir.box<none>
  %4 = fir.alloca i32
  fir.call @_QMprifPprif_this_image_no_coarray(%3, %4) fastmath<contract> : (!fir.box<none>, !fir.ref<i32>) -> ()
  %5 = fir.load %4 : !fir.ref<i32>
  %c1_i32 = arith.constant 1 : i32
  %6 = arith.cmpi eq, %5, %c1_i32 : i32
  fir.if %6 {
    %7 = fir.alloca i32
    fir.call @_QMprifPprif_num_images(%7) fastmath<contract> : (!fir.ref<i32>) -> ()
    %8 = fir.load %7 : !fir.ref<i32>
    hlfr.assign %8 to %2#0 : i32, !fir.ref<i32>
  }
  ...
}

```

at the LLVM IR level to take full effect. Also, to support selection of the **PRIF** implementation, we've introduced a new LLVM flag (`-fcoarray`). For command-line portability across compilers, the flag's name matches the analogous flag in the GNU toolchain.

2.3 Caffeine

CoArray Fortran Framework of Efficient Interfaces to Network Environments (**Caffeine**) [6, 33] is a parallel runtime library that is the first implementation of **PRIF**. The **Caffeine** runtime library is mostly written in Fortran and demonstrates that a subset of Fortran's non-parallel features can be used to provide support for the multi-image features. **PRIF** and **Caffeine** are both designed to rely solely upon a narrow serial subset of Fortran 2018 features (notably including `ISO_Fortran_binding.h`, assumed-type dummy arguments like `type(*)`, and assumed-rank dummy arguments like `array(..)`). The library implementation [22] is intended to be strictly standards-conformant, enabling **Caffeine** to interoperate with any Fortran compiler implementing these features in a standard-conforming manner. The **Caffeine** library currently supports compilation using either of two compiler families: LLVM Flang version 19 or later and GNU Fortran version 13 or later; support for additional compiler families is planned.

As suggested in Figure 1, **Caffeine** is implemented atop the GASNet-EX communication library [3, 4], which provides the communication services across a wide range of network hardware, including: Ethernet, InfiniBand, HPE Slingshot, Intel Omni-Path, Amazon EFA, and shared-memory workstations (no physical network).

Table 1 provides the updated status of **PRIF** feature support in **Caffeine**, which has made significant progress since previously reported in [5].

Table 1: Status of Caffeine’s support for the multi-image parallel features of Fortran (as of Caffeine v0.5.2)

Multi-image Fortran Feature	Status
Program startup and shutdown (incl. normal and error termination): STOP, ERROR STOP, END PROGRAM statements	yes
Collective subroutines: CO_{BROADCAST, SUM, MIN, MAX, REDUCE}	yes
Image queries: THIS_IMAGE, NUM_IMAGES, etc, intrinsic functions	yes
Synchronization: SYNC {ALL, IMAGES, MEMORY, TEAM} statements	yes
Storage management: Coarray allocation, deallocation and coarray aliases	yes
Coarray Queries: LCOBOUND, UCOBOUND, COSHAPE, etc.	yes
Contiguous and strided coarray access: Coarray puts and gets	yes
Teams: TEAM_TYPE intrinsic type and {FORM, CHANGE, END} TEAM statements	yes
Events: EVENT_TYPE intrinsic type, EVENT_QUERY subroutine and EVENT {POST, WAIT} statements	yes
Notifications: NOTIFY_TYPE intrinsic type and NOTIFY WAIT statement	yes
Atomics: ATOMIC_{INT, LOGICAL}_KIND kind parameters and ATOMIC_{DEFINE, REF, ...} subroutines	yes
Critical construct: CRITICAL and END CRITICAL	no
Locks: LOCK and UNLOCK statements	no
FAIL IMAGE statement	no

3 EVALUATION

3.1 Experimental Methodology

In order to assess the viability of our modifications to LLVM as well as the maturity of both **PRIF** and Caffeine, we opted to evaluate our approach using two benchmarks written by other members of the Fortran community:

- Caf-testsuite [20], a collection of correctness tests for CoArray Fortran (**CAF**) features. It checks whether compilers and runtimes conform to the Fortran standard’s coarray semantics by verifying the behavior of operations like coarray allocation, synchronization, image control, and remote memory access.
- Cafbench [9], a performance benchmarking suite designed to evaluate and compare the performance of **CAF** features across different compiler and runtime implementations. It focuses on measuring the efficiency of coarray operations such as communication (e.g. put, get, sync) and scalability in multi-image execution environments.

Because multi-image parallel feature performance may be highly sensitive to the environment, we opted to test our implementation in shared memory on both AArch64 Neoverse-V1 (AWS Graviton3e™, 64 cores @2.5 Ghz)² and X86-64 (Intel Xeon™ Sapphire Rapids, 80 cores @2.1Ghz). In both configurations, OS and kernel versions are the same (Linux RHEL 9, kernel v5.14). It should be noted that:

- Our prototype implementation of flang is available at [35] and we benchmarked at commit c17dd15683e3. The Caffeine library used was release version 0.5.2 (commit 36f0d0a73beb). Caffeine used GASNet-EX [23] library version 2024.5.0.

²Other AArch64 architecture were also tested: Neoverse-N1 and Neoverse-V2. Since they provide similar results, only Neoverse-V1 results are reported here.

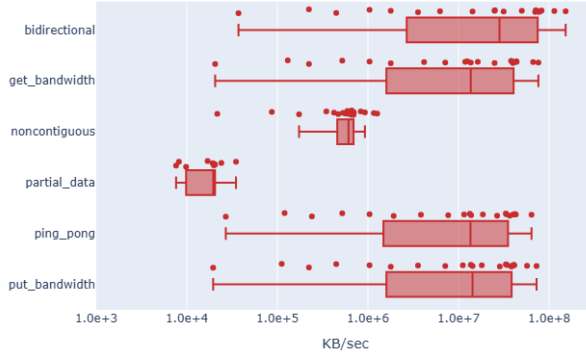
- In shared memory, we compared our results with GNU Fortran version 15.0.0 and OpenCoarrays [12] version 2.10.3, using OpenMPI version 5.0.8. OpenCoarrays was compiled using both the library’s default compilation flags and with -DSTRIDED in order to activate optimized strided-array transfer support, and the best results reported for each test.

3.2 Comparison with Existing Open-Source Implementation in Shared Memory Systems

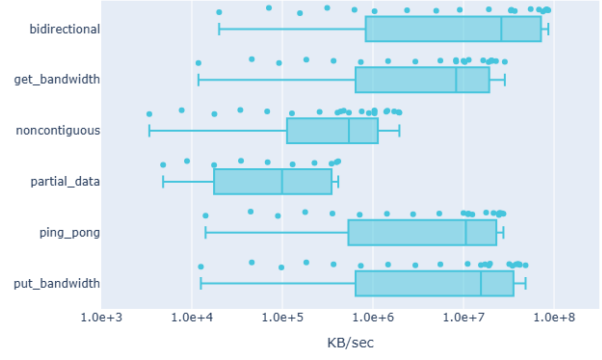
Figure 3 illustrates the comparison between flang/Caffeine and GNU-Fortran/OpenCoarrays on both AArch64 and X86-64 using the benchmarks provided in the Caf-testsuite [20]. We are unable to conduct measurements of GNU Fortran using Caffeine at this time, as only flang currently supports lowering to **PRIF**. This test suite measures two main indicators: latency and bandwidth using various GET and PUT coarray access operations. Following the benchmark’s default configuration, only two images were used. Nevertheless, it provides an effective evaluation of the point-to-point communication performance for the two different multi-image implementations.

Each benchmark test (e.g., bidirectional, get_bandwidth, noncontiguous, partial_data, ping_pong, and put_bandwidth) runs over a range of data transfer sizes up to 4 MiB to provide a comprehensive view of overall coarray access performance with respect to transfer size. Figure 3 represents these individual benchmarks by box-and-whisker plots. Each row represents the effective transfer bandwidth achieved by a given test across all the transfer sizes. For more insight, the average transfer bandwidth for each individual transfer size is additionally superimposed as a dot atop the box-and-whisker plot.

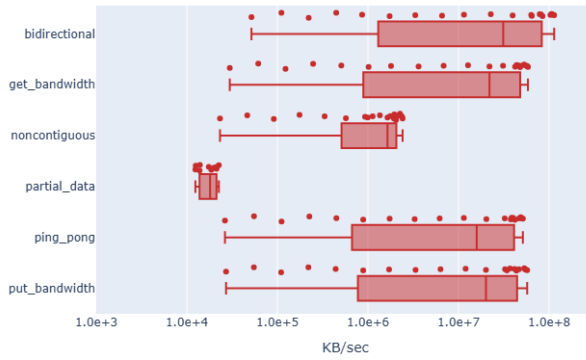
At first glance, it can be observed that our flang/Caffeine approach and GNU-Fortran/OpenCoarrays both provide comparable performance and qualitatively similar trends, regardless of the underlying architecture. This highlights the favorable efficiency of our prototype, demonstrating performance comparable to a mature implementation that’s been in production use for over 10 years. Two qualitative differences that can nonetheless be spotted:



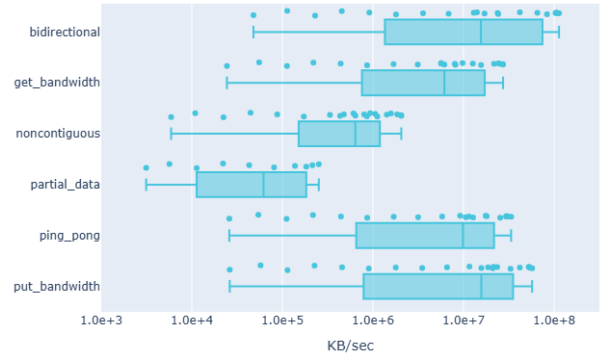
(a) flang and Caffeine on x86-64



(b) GNU Gfortran and OpenCoarrays on x86-64



(c) flang and Caffeine on AArch64



(d) GNU Gfortran and OpenCoarrays on AArch64

Figure 3: Overview of micro-benchmark results from Caf-testsuite on shared-memory systems

- For the `partial_data` test, flang/Caffeine bandwidth results are more tightly grouped across transfer sizes, whereas the GNU-Fortran/OpenCoarrays results have a wider variance across sizes but also a significantly higher mean bandwidth. This test measures the bandwidth of interleaved non-contiguous coarray put operations with 16KiB stride on both sides. The performance of this test in shared memory is likely very sensitive to details of memory layout and induced cache behavior, which is expected to differ between compilers. It's also likely to benefit from loop-interchange optimizations, which flang is not currently performing for coarray accesses.
- Figure 4 zooms into the results for the `noncontiguous` test, which measures the bandwidth obtained by a Fortran coindexed assignment statement of the form:
`arr(1,1:xfer_size)[2] = arr(1,1:xfer_size)`
 which induces repeated non-contiguous coarray put operations with 64-byte stride on both sides. The AArch64 results seem to uniformly favor our approach, while on x86-64 the relative performance of the two approaches is more mixed.

Table 2 summarizes the average latency of small contiguous coarray access operations PUT and GET (smaller is better). We find that our approach provides competitive latency performance. The differences likely arise due to differences in OpenCoarrays vs Caffeine,

Table 2: Latency for 4-byte coindexed access in shared memory, using flang (Caffeine) versus GNU Gfortran (OpenCoarrays)

Avg. Latency (nanoseconds)	Aarch64		X86-64	
	flang	Gfortran	flang	Gfortran
PUT	119	141	63	178
GET	129	136	63	180

and differences between the MPI and GASNet-EX communication backends. It could also be possible that our implementation, using `PRIF`, is more straightforward in the compiler and avoids some complexity in the handling of coarrays. Adding `PRIF` support to GNU Fortran could help to confirm this intuition, and this is left for future work.

3.3 Comparison in Distributed Memory

We also evaluated microbenchmark performance in distributed memory using two dedicated nodes on the CPU partition of the "Perlmutter" [29] Hewlett Packard Enterprise (HPE) Cray EX supercomputer at National Energy Research Scientific Computing Center (NERSC) [28]. Each Perlmutter node has 2×64 -core Advanced Micro Devices, Inc. (AMD) EPYC™ 7663 Milan processors @ 2.45GHz

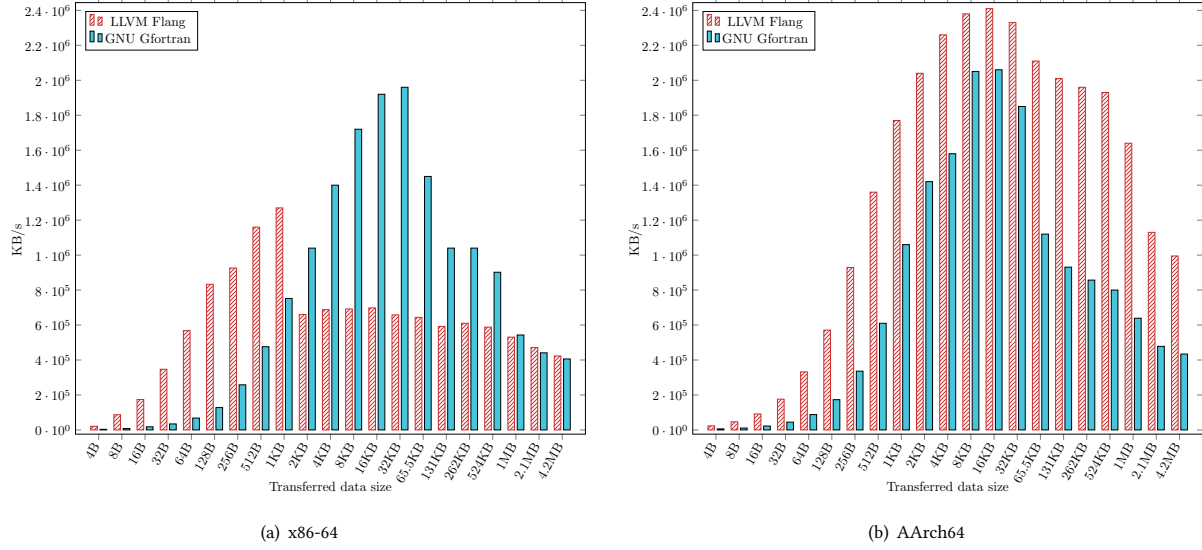


Figure 4: Results of Noncontiguous test from Caf-testsuite in shared memory

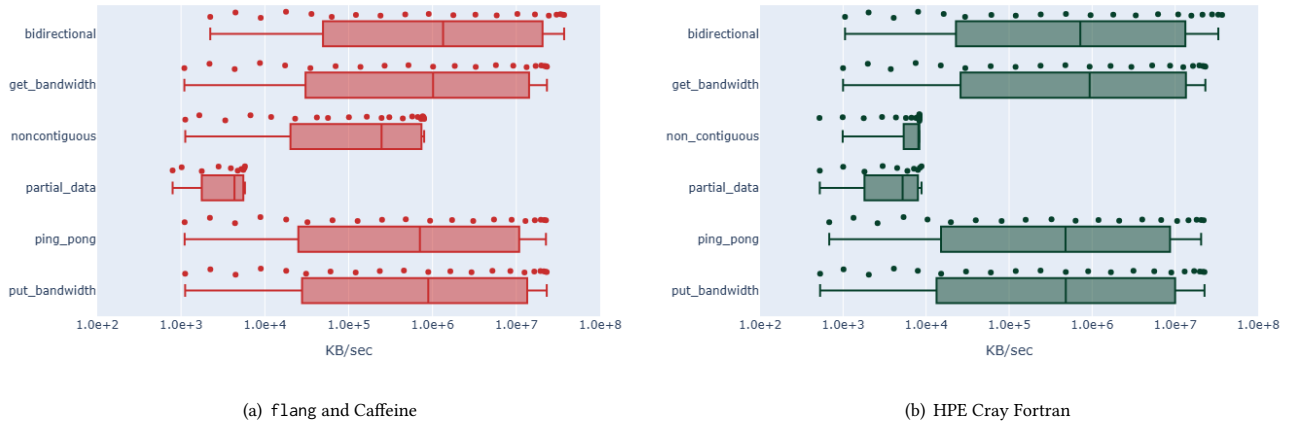
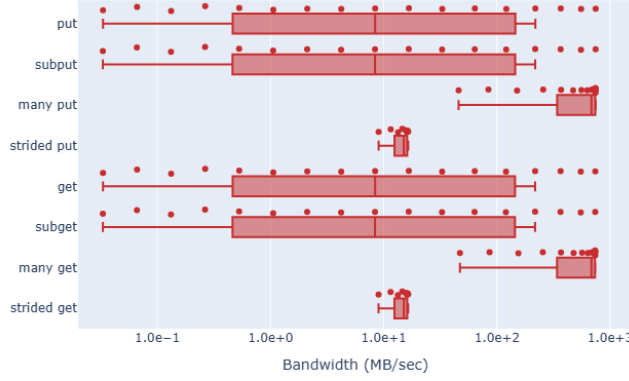


Figure 5: Results from running Caf-testsuite in distributed memory across two nodes of the NERSC Perlmutter supercomputer, connected over an HPE Slingshot 11 network

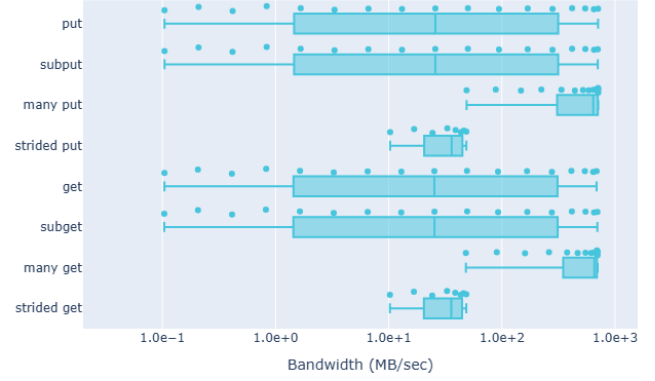
and one HPE Slingshot 11 200Gbps Cassini NIC connected in a dragonfly topology. We ran one image on each physical node, ensuring that all inter-image communication was routed over the Slingshot interconnect, using libfabric CXI provider version 1.20.1 and Slingshot Host Software version 11.0.2. For the flang/Caffeine results, we used all the same compiler/runtime software versions described in §3.1. For a performance baseline on this system, we used the vendor-provided **HPE CCE Fortran** compiler [18], version 18.0.0, with Cray Programming Environment version 2.7.32. While building Caf-testsuite, both compilers were provided with the `-O3` optimization option. For the **CCE Fortran** compiler, we additionally added the `-Onointerchange` option to disable loop interchange optimizations; without this option the compiler heuristics made a poor

choice to interchange inner loops in the benchmark’s timed regions, resulting in a slowdown exceeding several orders of magnitude for the contiguous transfer tests.

Figure 5 presents the results of a distributed-memory microbenchmark performance comparison on NERSC Perlmutter using Caf-testsuite. As with the shared-memory results in §3.2, the bandwidth performance delivered for the distributed-memory tests exercising contiguous transfers is qualitatively similar between flang/Caffeine and **CCE Fortran**. For the noncontiguous test, flang/Caffeine demonstrates wider bandwidth variance across transfer sizes, but also significantly higher mean and peak bandwidths. The two configurations are more closely matched on the `partial_data`



(a) flang and Caffeine on AArch64



(b) GNU Gfortran and OpenCoarrays on AArch64

Figure 6: Overview from a subset of micro-benchmark results from Cafbench

test, with CCE Fortran showing a slight advantage in mean and peak bandwidth.

Table 3 presents the average latency results for small contiguous coarray access operations PUT and GET in distributed memory (smaller is better). Again we find that our flang-based approach provides competitive latency performance. The minor differences likely arise due to the use of Caffeine and GASNet-EX for network communication, versus HPE's proprietary communication backend.

Overall, these results demonstrate that our prototype flang/Caffeine solution is already delivering multi-image Fortran performance that is comparable to a mature vendor product.

Table 3: Latency for 4-byte coindexed access across an HPE Singshot network, using flang (Caffeine) versus HPE CCE

Avg. Latency (microseconds)	flang / Caffeine	HPE CCE
PUT	3.64	7.17
GET	3.53	4.30

3.4 Experimental results using Cafbench

Similarly to Caf-testsuite, the Cafbench [9] project provides several microbenchmarks to measure the latency and bandwidth performance for coarray PUT and GET operations, with or without strides. It also includes benchmark tests for synchronization operations (sync_all, sync_images, ...); this is a major difference from Caf-testsuite that does not evaluate such operations. The second main difference is the use of coarray dummy arguments as procedure arguments. Supporting this kind of declaration is nontrivial and required some additional effort in our fork of LLVM. Indeed, matching LLVM's way of handling dummy arguments is not straightforward with the PRIF interface. In LLVM, dummy arguments are allocated inside of the callee procedure. Consequently, this is the best location to call the PRIF procedure prif_alias_create that creates a new coarray descriptor for the callee's dummy argument that is aliased to the coarray element data of the caller's descriptor. However, this

procedure requires a PRIF descriptor that is only available on the caller side in LLVM. This is the main reason why the flow depicted in §2.2 is split into two parts: using a GlobalOp operation helps to retrieve the descriptor in the callee procedure.

While our initial goal was to compare the Cafbench performance of flang/Caffeine against both GNU Fortran and HPE CCE, we were unable to successfully compile Cafbench using the HPE CCE Fortran compiler. The Cafbench code crashed the CCE version 18.0.0 compiler on Perlmutter, and also the latest CCE version 19.0.0 with Cray Programming Environment version 2.7.33 on a different HPE Cray EX supercomputer. This Cray Fortran compiler defect has been reported to HPE via NERSC as case number 5392228187. These limitations prevented a full performance comparison using Cafbench.

Figure 6 illustrates the comparison between flang/Caffeine and GNU-Fortran/OpenCoarrays on our shared-memory AArch64 test system using the benchmarks provided in Cafbench. In order to confirm results shown by Caf-testsuite, only two images were used.

Each benchmark test (e.g., put, get, subput, subget, strided_put, strided_get, many_put, and many_get) runs over a range of data transfer sizes up to 2 MiB. The trend that was previously demonstrated in the Section §3.2 is confirmed by Figure 6: both flang/Caffeine and GNU-Fortran/OpenCoarrays deliver similar results in terms of mean performance and distribution. This highlights the favorable performance of our prototype, even with experimental support for dummy arguments.

Although flang's multi-image support lacks the maturity of its colleagues, our prototype nevertheless successfully compiled a working multi-image executable for Cafbench. This demonstrates our preliminary support for coarray dummy arguments and exercised our use of procedures such as prif_alias_create and prif_alias_destroy. We plan to further investigate optimizing support for coarray dummy arguments, even lacking a comparison baseline with the Cray Fortran compiler.

4 FUTURE WORK

We have started to contribute our flang improvements for lowering multi-image features to **PRIF** into the upstream LLVM project repository. As of the time of writing, we've merged working lowering for the following features into the upstream llvm/main branch:

- (1) SPMD program initialization and -fcoarray option [36]
- (2) Basic image identification intrinsics (this_image, num_images) [37]
- (3) Image synchronization statements (sync all, sync images, sync memory) [39]
- (4) Collective subroutine intrinsics (co_broadcast, co_min, co_max, co_sum) [38]

Work continues on completing the multi-image parallel features for flang.

LLVM Flang performs lowering from its **HLFIR** and **FIR** dialects, which are mixed with a few standard dialects, directly to LLVM-IR [7]. This allows the compiler to benefit from the various operations, dialects, and **MLIR** transformations available in **MLIR**. We were thus able to leverage the features offered by **MLIR** to lower multi-image features into corresponding **PRIF** procedure calls. There is still significant remaining work in LLVM to support complete and stable lowering to **PRIF**. Improving support for coarray dummy arguments (although compiling and running Cabbench is a very good start), handling allocatable members of coarrays with derived type, and optimizing access operations for coarrays are the main areas for future development work.

The use of **MLIR** also opens up new possibilities for future work. We are currently working to propose the addition of a new **MLIR** dialect named Multi-Image Fortran (**MIF**), which will model operations of the multi-image features in flang [34]. In order to improve organization and code readability, we propose the handling of **MIF** operations and lowering to **PRIF** be moved into this specific dialect. Once the new dialect is ready for adoption, the approach described in §2.2 will be adjusted to directly use these new dialect operations during lowering, thus minimizing changes to existing lowering for statements and intrinsics. Finally, we'll implement a transformation pass at the **MLIR** level to convert **MIF** operations to LLVM-IR, replacing **MIF** dialect operations with the corresponding **PRIF** procedure calls.

An example of **IR** using such a dialect is depicted in Listing 3, which corresponds to a straightforward translation of the example given in Listing 1. A comparison with HLFIR translation presented in Listing 2 shows that the function calls fir.call have been replaced by the higher-level operations mif.this_image and mif.num_images. We have an initial prototype implementation for this dialect, covering all the aforementioned HLFIR operations that have been upstreamed. This prototype implementation will be refined to handle more multi-image features.

5 CONCLUSION

In this paper, we have provided an overview of the work accomplished with **PRIF** and the Caffeine parallel runtime library to support the multi-image parallel features of Fortran in LLVM Flang.

Section 3 showed that OpenCoarrays (with GNU Fortran) and Caffeine deliver comparable performance for basic multi-image Fortran features. In some cases our flang-based prototype stack

Listing 3: MIF Dialect output example

```
func.func @_Qmain() attributes {fir.bindc_name =
  "MAIN"} {
  ...
  %4 = mif.this_image -> i32
  %c1_i32 = arith.constant 1 : i32
  %5 = arith.cmpi eq, %4, %c1_i32 : i32
  fir.if %5 {
    %6 = mif.num_images -> i32
    hlfir.assign %6 to %2#0 : i32, !fir.ref<i32>
  }
  ...
}
```

even provides better performance than a mature implementation available for over 10 years. This demonstrates that the overhead associated with using a portable interface such as **PRIF** is negligible compared to what is offered by another open-source compiler. Apart from initializing the parallel environment and creating a coarray descriptor, **PRIF** is designed to provide a close semantic match to the multi-image operations specified by the Fortran standard.

The Caffeine implementation of **PRIF** has reached new levels of maturity, now implementing nearly all the multi-image features and procedures in **PRIF**. By adopting **PRIF** and Caffeine, our fork of flang can now support numerous multi-image operations without having to worry about too many functional constraints. In addition to Caffeine supporting LLVM Flang, Caffeine can offer the same capabilities to any compiler that targets **PRIF**. Moreover, such a compiler could seamlessly switch to any other parallel runtime library that supports **PRIF**. By contrast, the OpenCoarrays interface hard-wires GNU-Fortran-specific details such as variable descriptor types. We are aware that additional work must be finished in order to fully support **PRIF**, and to integrate forthcoming changes expected to be introduced in the upcoming Fortran 202Y standard. In the future, we expect that utilization of Caffeine will be a more advantageous option than OpenCoarrays in order to fully benefit from Fortran's multi-image features.

ACKNOWLEDGMENTS

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research.

This research was supported in part 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.

This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility located at Lawrence Berkeley National Laboratory, operated under Contract No. DE-AC02-05CH11231.

Selected portions of this paper are reproduced with permission from [5].

ACRONYMS

AMD	Advanced Micro Devices, Inc.
AST	Abstract Syntax Tree
CAF	CoArray Fortran
Caffeine	CoArray Fortran Framework of Efficient Interfaces to Network Environments
CCE	Cray Compiling Environment
FIR	Fortran Intermediate Representation
GCC	GNU Compiler Collection
HLFIR	High-level Fortran Intermediate Representation
HPC	High-Performance Computing
HPE	Hewlett Packard Enterprise
IR	Intermediate Representation
MIF	Multi-Image Fortran
MLIR	Multi-Level Intermediate Representation
NERSC	National Energy Research Scientific Computing Center
PGAS	Partitioned Global Address Space
PRIF	Parallel Runtime Interface for Fortran
SPMD	Single-Program, Multiple-Data

REFERENCES

- [1] B. Austin et al. 2020. *NERSC-10 Workload Analysis*. National Energy Research Scientific Computing Center. doi:10.25344/S4N30W
- [2] Robert T Biedron, Jan-René Carlson, Joseph M Derlaga, Peter A Gnoffo, Dana P Hammond, William T Jones, Bil Kleb, Elizabeth M Lee-Rausch, Eric J Nielsen, Michael A Park, et al. 2017. FUN3D Manual: 13.2. NASA TM 219661 (2017), 362 pages. https://fun3d.larc.nasa.gov/papers/FUN3D_Manual-13.2.pdf.
- [3] Dan Bonachea and Paul H. Hargrove. 2017. *GASNet Specification, v1.8.1*. Technical Report LBNL-2001064. Lawrence Berkeley National Laboratory. doi:10.2172/1398512
- [4] Dan Bonachea and Paul H. Hargrove. 2018. GASNet-EX: A High-Performance, Portable Communication Library for Exascale. In *Proceedings of Languages and Compilers for Parallel Computing (LPC'18) (LNCS, Vol. 11882)*. Springer, Salt Lake City, UT, 16 pages. doi:10.25344/S4QP4W
- [5] Dan Bonachea, Katherine Rasmussen, Brad Richardson, and Damian Rouson. 2024. Parallel Runtime Interface for Fortran (PRIF): A Multi-Image Solution for LLVM Flang. In *Tenth Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC2024)*. IEEE, Atlanta, GA, USA, 11 pages. doi:10.25344/S4N017
- [6] Dan Bonachea, Katherine Rasmussen, Brad Richardson, and Damian Rouson. 2025. Caffeine: A parallel runtime library for supporting modern Fortran compilers. *Journal of Open Source Software* 10, 107 (2025), 7895. doi:10.21105/joss.07895
- [7] Nick Brown. 2024. Fully integrating the Flang Fortran compiler with standard MLIR. In *Tenth Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC2024)*. IEEE, Atlanta, GA, USA, 939–949. doi:10.1109/SCW63240.2024.00133
- [8] Gokhan Danabasoglu, J-F Lamarque, J Bacmeister, DA Bailey, AK DuVivier, Jim Edwards, LK Emmons, et al. 2020. The Community Earth System Model version 2 (CESM2). *Journal of Advances in Modeling Earth Systems* 12, 2 (2020), 35 pages. doi:10.1029/2019MS001916
- [9] David Henty. 2013. *Fortran Coarray Micro-Benchmark Suite*. GitHub repository. <https://github.com/davidhenty/cafbench>.
- [10] Ming Ding, Xuhua Zhou, Haojie Zhang, Haozhi Bian, and Qiang Yan. 2021. A review of the development of nuclear fuel performance analysis and codes for PWRs. *Annals of Nuclear Energy* 163 (2021), 108542. doi:10.1016/j.anucene.2021.108542
- [11] Deepak Eachempati, Hyoung Joon Jun, and Barbara Chapman. 2010. An Open-source Compiler and Runtime Implementation for Coarray Fortran. In *Proceedings of the Fourth Conference on Partitioned Global Address Space Programming Models (PGAS'10)*. ACM, New York, NY, USA, Article 13, 8 pages. doi:10.1145/2020373.2020386
- [12] Alessandro Fanfarillo, Tobias Burnus, Valeria Cardellini, Salvatore Filippone, Dan Nagle, and Damian Rouson. 2014. OpenCoarrays: Open-source Transport Layers Supporting Coarray Fortran Compilers. In *Partitioned Global Address Space Programming Models (PGAS)*. ACM, Eugene, OR, USA, 11 pages. doi:10.1145/2676870.2676876
- [13] Fortran Standards Committee JTC1/SC22/WG5. 2010. *Information technology — Programming languages — Fortran, ISO/IEC 1539-1:2010*. International Organization for Standardization (ISO), Geneva, Switzerland. <https://www.iso.org/standard/50459.html>.
- [14] Fortran Standards Committee JTC1/SC22/WG5. 2018. *Information technology — Programming languages — Fortran, ISO/IEC 1539-1:2018*. International Organization for Standardization (ISO), Geneva, Switzerland. <https://www.iso.org/standard/72320.html>.
- [15] Fortran Standards Committee JTC1/SC22/WG5. 2023. *Information technology — Programming languages — Fortran, ISO/IEC 1539-1:2023*. International Organization for Standardization (ISO), Geneva, Switzerland. <https://www.iso.org/standard/82170.html>.
- [16] fpm contributors. 2025. *Fortran Package Manager (fpm)*. Web site. <https://fpm.fortran-lang.org>.
- [17] Shiyao Ge. May 2016. *Implementation and evaluation of additional parallel features of Coarray Fortran*. Master's thesis. University of Houston. <http://hdl.handle.net/10657/3268>.
- [18] Hewlett Packard Enterprise. 2025. *Cray Compiling Environment Fortran compiler*. Online documentation. https://cpe.ext.hpe.com/docs/latest/getting_started/CPE-CCE-Fortran.html.
- [19] HPCTools Group, University of Houston. 2013-2015. *OpenUH Compiler for Coarray Fortran*. GitHub repository. <https://github.com/uhhpctools/openuh>.
- [20] HPCTools Group, University of Houston. 2014. *CAF Validation and Performance Test Suites*. GitHub repository. <https://github.com/uhhpctools/caf-testsuite>.
- [21] Intel Corporation. 2025. *Intel Fortran Compiler Developer Guide and Reference*. Web site. <https://www.intel.com/content/www/us/en/docs/fortran-compiler/developer-guide-reference>.
- [22] Lawrence Berkeley National Laboratory. 2025. *Caffeine: CoArray Fortran Framework of Efficient Interfaces to Network Environments*. GitHub repository. <https://go.lbl.gov/caffeine>.
- [23] Lawrence Berkeley National Laboratory. 2025. *GASNet*. Web site. <http://gasnet.lbl.gov>.
- [24] LLVM Organization. 2025. *Flang Compiler Documentation*. Web site. <https://flang.llvm.org>.
- [25] LLVM Organization. 2025. *Flang Fortran Standards Support*. Web site. <https://flang.llvm.org/docs/FortranStandardsSupport.html>.
- [26] LLVM Organization. 2025. *LLVM Compiler Infrastructure Project*. GitHub repository. <https://github.com/llvm/llvm-project>.
- [27] Kevin B. McGrattan, Randall J. McDermott, Craig G. Weinschenk, and Glenn P. Forney. 2013. Fire dynamics simulator user's guide. *NIST special publication 1019* (2013), 104 pages. doi:10.6028/NIST.sp.1019
- [28] NERSC. 2025. *National Energy Research Scientific Computing Center (NERSC)*. Online documentation. <https://www.nersc.gov>.
- [29] NERSC. 2025. *Perlmuter System Details*. Online documentation. <https://docs.nersc.gov/systems/perlmuter/architecture/>.
- [30] Numerical Algorithms Group Ltd. 2025. *nAG Fortran Compiler*. Online documentation. <https://nag.com/fortran-compiler/>.
- [31] PRIF Committee. 2025. *Parallel Runtime Interface for Fortran (PRIF) Specification (Rev. 0.6)*. Technical Report LBNL-2001698. Lawrence Berkeley National Laboratory. doi:10.25344/S4M01X
- [32] Damian Rouson, Zhe Bai, Dan Bonachea, Kareem Ergawy, Ethan Gutmann, Michael Klemm, Katherine Rasmussen, Brad Richardson, Sameer Shende, David Torres, and Yunhao Zhang. 2025. Automatically parallelizing batch inference on deep neural networks using Fiats and Fortran 2023 "do concurrent". In *5th International Workshop on Computational Aspects of Deep Learning (CADL)*. eScholarship, Hamburg, Germany, 12 pages. doi:10.25344/S4VG6T
- [33] Damian Rouson and Dan Bonachea. November 2022. Caffeine: CoArray Fortran Framework of Efficient Interfaces to Network Environments. In *Proceedings of the Eighth Annual Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC2022)*. IEEE, Dallas, Texas, USA, 9 pages. doi:10.25344/S4459B
- [34] SiPearl, R&D. 2025. *Discussion on the Multi-Image Fortran dialect*. GitHub Pull Request. <https://github.com/SiPearl/llvm-project/pull/4>.
- [35] SiPearl, R&D. 2025. *SiPearl Fork of the LLVM Compiler Infrastructure project*. GitHub repository. <https://github.com/SiPearl/llvm-project/tree/prif>.
- [36] SiPearl, R&D. Aug 2025. *Add flang support for -fcoarray and initializing PRIF*. GitHub Pull Request. <https://github.com/llvm/llvm-project/pull/151675>.
- [37] SiPearl, R&D. Aug 2025. *Add flang support for THIS, IMAGE and NUM_IMAGES with PRIF*. GitHub Pull Request. <https://github.com/llvm/llvm-project/pull/154081>.
- [38] SiPearl, R&D. Sep 2025. *Add flang lowering for CO_{BROADCAST, MAX, MIN, SUM} to PRIF*. GitHub Pull Request. <https://github.com/llvm/llvm-project/pull/154770>.
- [39] SiPearl, R&D. Sep 2025. *Add flang lowering of SYNC ALL, SYNC MEMORY and SYNC IMAGES to PRIF*. GitHub Pull Request. <https://github.com/llvm/llvm-project/pull/154166>.
- [40] William C Skamarock, Joseph B Klemp, Jimmy Dudhia, David O Gill, et al. 2019. A description of the advanced research WRF model version 4. *National Center for Atmospheric Research: Boulder, CO, USA* 145 (2019), 145. doi:10.5065/1dfh-6p97
- [41] David Spickett. 2025. *LLVM Fortran Levels Up: Goodbye flang-new, Hello flang!* LLVM Project Blog. <https://blog.llvm.org/posts/2025-03-11-flang-new/>.