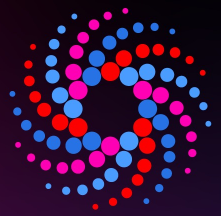


SC22
Dallas, TX | hpc accelerates.

Caffeine: Co-Array Fortran Framework of Efficient Interfaces to Network Environments

Damian Rouson and Dan Bonachea

Computer Languages and Systems Software (CLaSS) Group

LLVM-HPC2022, 13 November 2022

https://go.lbl.gov/caffeine

doi:10.25344/S4459B

BERKELEY LAB
Bringing Science Solutions to the World

U.S. DEPARTMENT OF ENERGY
Office of Science

Caffeine: Co-Array Fortran Framework of Efficient Interfaces to Network Environments

Damian Rouson and Dan Bonachea

Computer Languages and Systems Software (CLaSS) Group

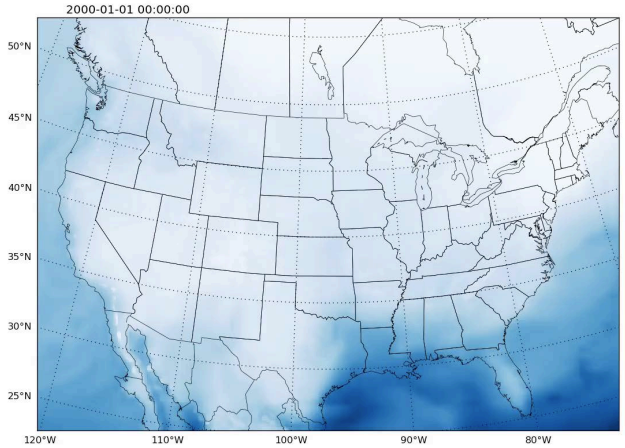
LLVM-HPC2022, 13 November 2022

https://go.lbl.gov/caffeine **doi:10.25344/S4459B**

Outline

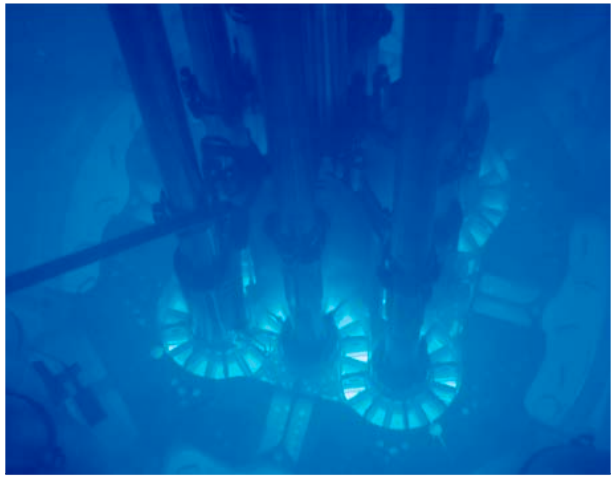
- Introduction
  - Why Fortran matters
  - Motivation and objectives
  - Contributions
- Methodology
- Discussion of Results
- Future Work
- Conclusions

Why Fortran Matters



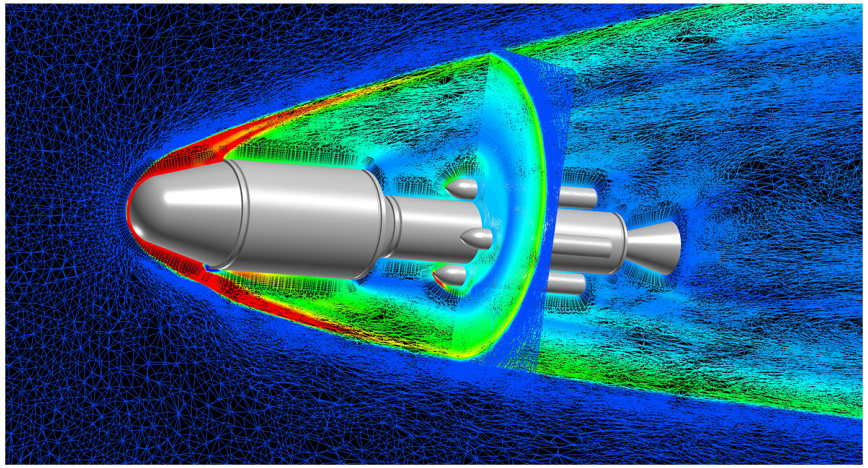
Intermediate Complexity Atmospheric Research (ICAR) Model
Courtesy of Ethan Gutmann, NCAR

Weather & Climate



NRC File Photo

Nuclear Energy

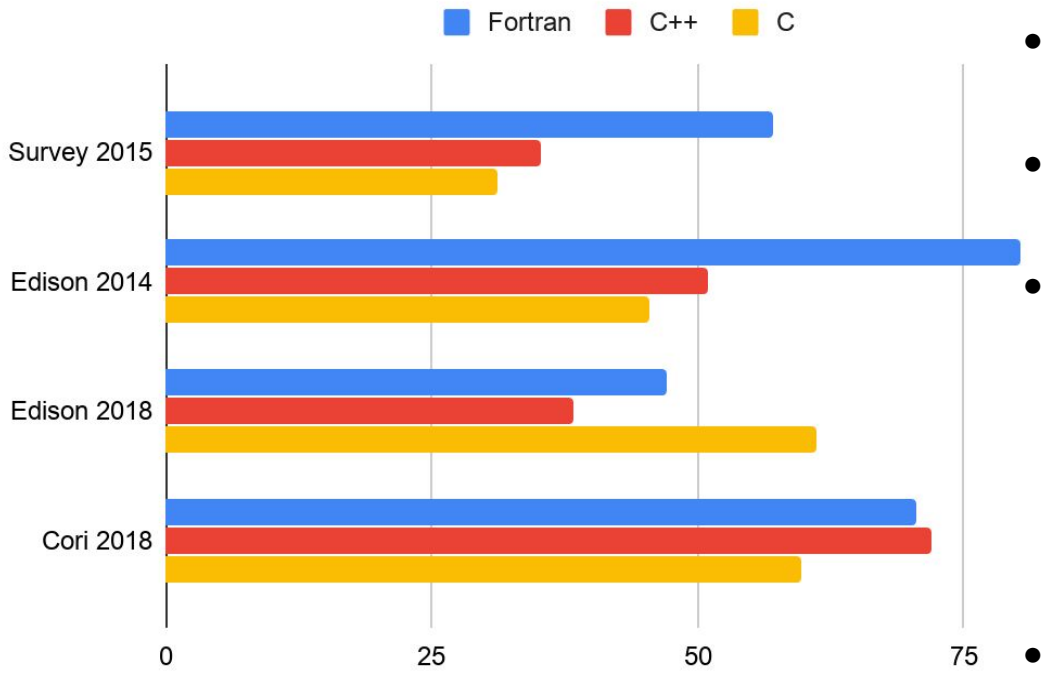


FUN3D Mesh Adaptation for Mars Ascent Vehicle, Courtesy of
Eric Nielsen & Ashley Korzun, NASA Langley

Aerospace

Why Fortran Matters

Compiled languages used at NERSC



Totals exceed 100% because some users rely on multiple languages.

- Fortran remains a common language for scientific computation.
- Noteworthy increases in C++ and multi-language
- Language use inferred from runtime libraries recorded by ALTD. (previous analysis used survey data)
  - ALTD-based results are mostly in line with survey data.
  - No change in language ranking
  - Survey underrepresented Fortran use.
- Nearly ¼ of jobs use Python.

Motivations

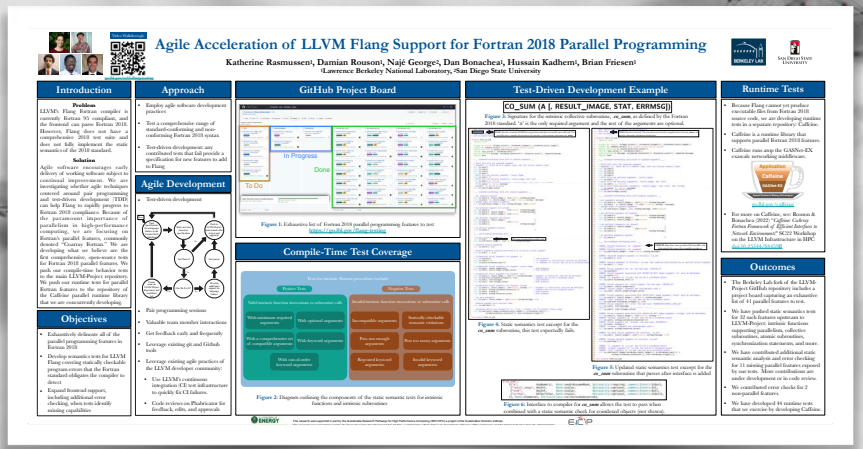
- Parallelism is of paramount importance in HPC.
- Fortran added parallel features in the 2008 standard, whereas Flang currently compiles Fortran 95 and parses Fortran 2018.
- Several competing compilers support Fortran's parallel features:
  - HPE Cray
  - Intel
  - GCC/OpenCoarrays
  - NAG

Objectives

To accelerate LLVM Flang's adoption of parallel features through test-driven development:

- Runtime unit tests that drive the development of Caffeine and
- Compile-time semantics tests that drive our contributions to the LLVM Flang frontend*.

* See our SC22 research poster: doi: 10.25344/S4CP4S:



Fortran Standard Terms

Term	Definition
assumed -type	unlimited polymorphic data object declared with `type(*)`, described informally here as "type-agnostic"
assumed -rank	data-object dummy argument that assumes the rank of its effective argument, described here as "rank-agnostic"
coarray	data structure partitioned across a team's images and accessible by each image in the corresponding team
effective argument	entity that is argument-associated with a dummy argument in a procedure call
image	instance of a Fortran program
intrinsic	entity or operation defined in the Fortran standard and accessible without further definition or specification
rank	number of array dimensions of a data entity (zero for a scalar entity)
team	ordered image set created by executing a `form team` statement, or the initial ordered set of all images
unlimited polymorphic	able to have any dynamic type during program execution

Contributions

Key Insights & Innovations:

- A subset of the Fortran 2018 *non-parallel* features suffice for writing a runtime library, mostly in Fortran, to support the Fortran 2018 *parallel* features
- Using Fortran 2018 type-agnostic procedure arguments facilitates porting across compilers:
  - no explicit reference to compiler-specific data descriptors
- Using Fortran 2018 type- and rank-agnostic arguments greatly simplifies the parallel runtime library:
  - almost no branching on type or rank

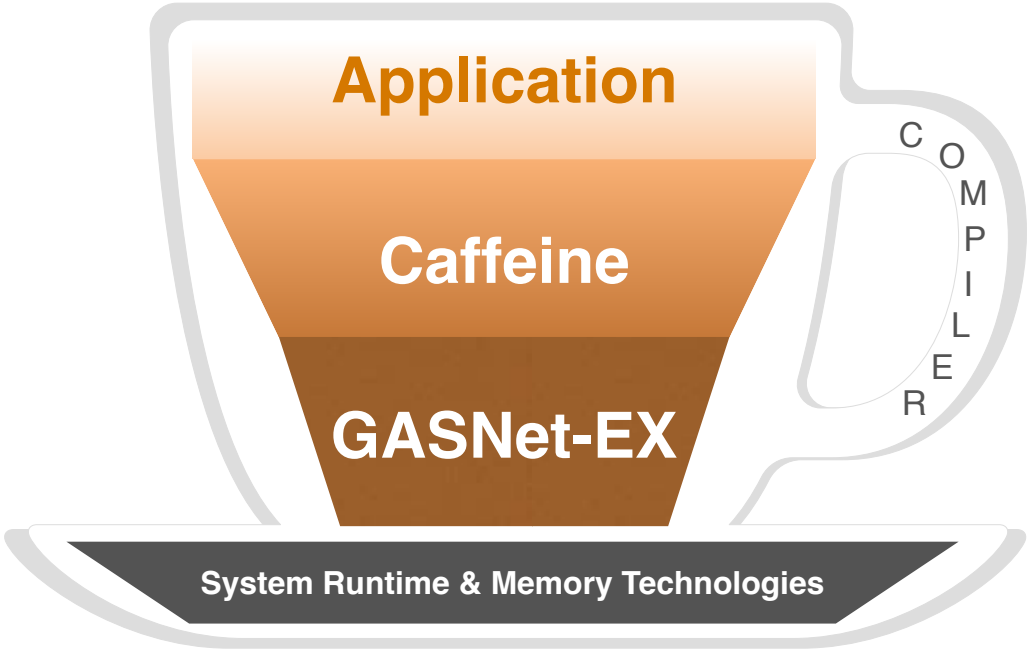
Bonus:
Writing in the language of the users improves sustainability by lowering a barrier to community maintenance.

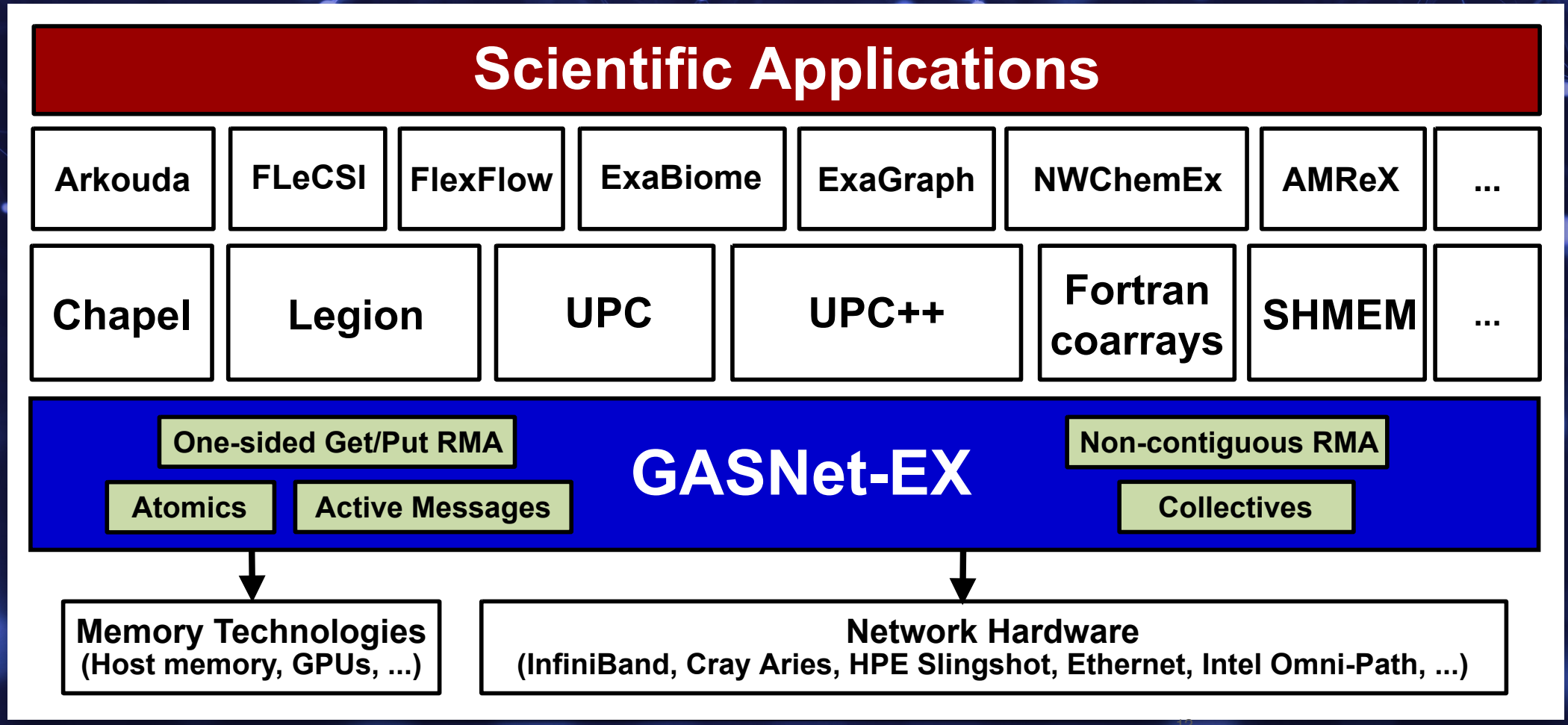
9

Outline

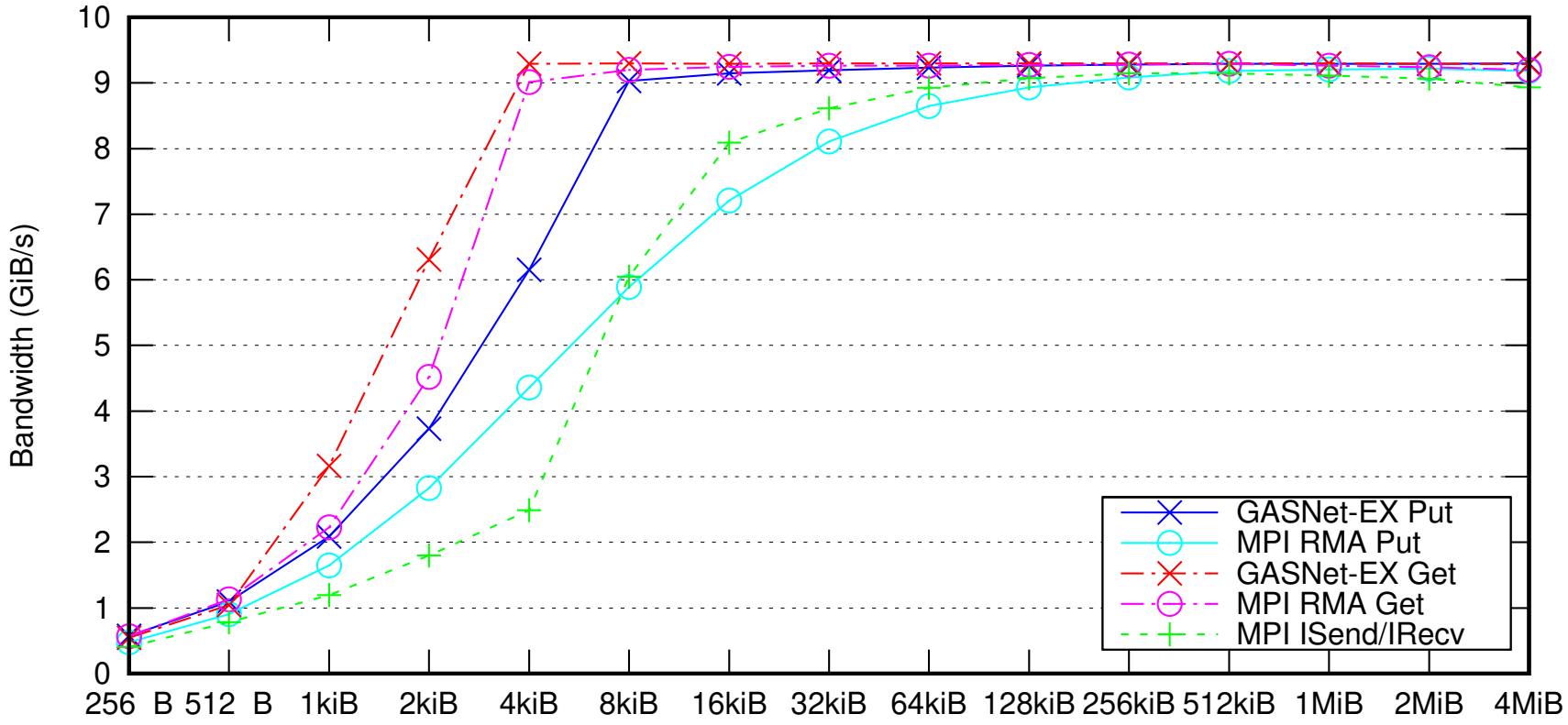
- Introduction
- Methodology
  - Software stack
  - GASNet-EX
  - Caffeinating LLVM Flang
- Discussion of Results
- Future Work
- Conclusions

Software Stack




Scientific Applications
Arkouda
FLeCSI
FlexFlow
ExaBiome
ExaGraph
NWChemEx
AMReX
...
Chapel
Legion
UPC
UPC++
Fortran coarrays
SHMEM
...
One-sided Get/Put RMA
Atomics
Active Messages
GASNet-EX
Non-contiguous RMA
Collectives
Memory Technologies
(Host memory, GPUs, ...)
Network Hardware
(InfiniBand, Cray Aries, HPE Slingshot, Ethernet, Intel Omni-Path, ...)

GASNet-EX Performance

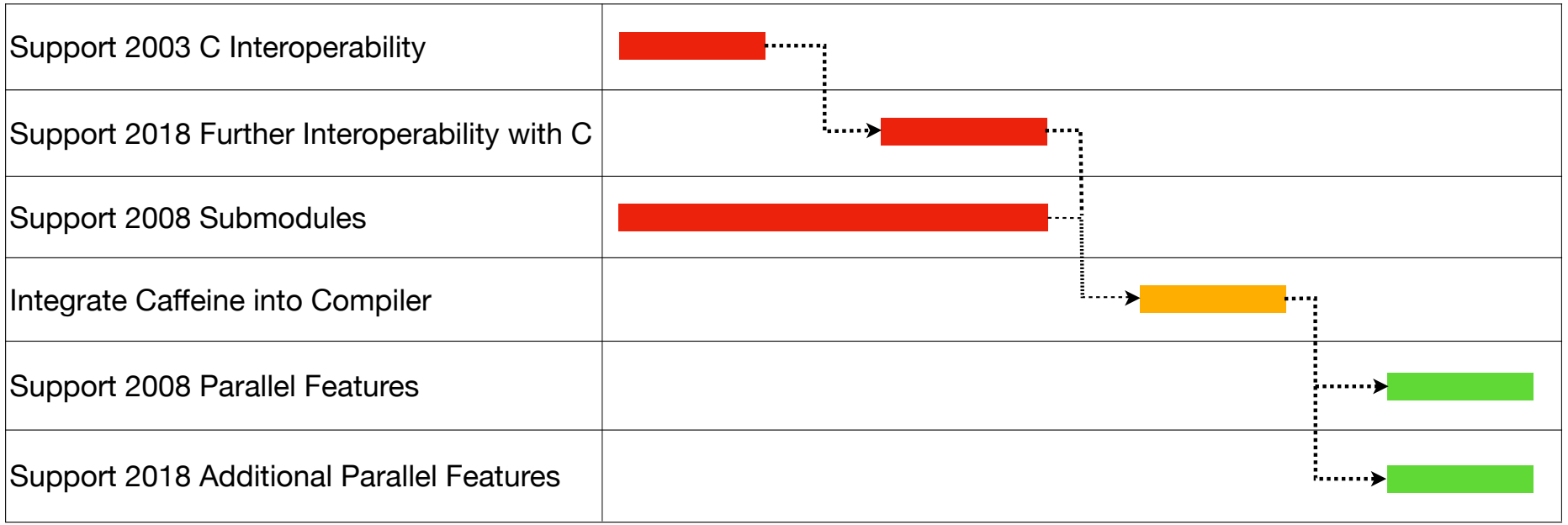


Cori-I:
Haswell
Aries
Cray MPI

Microbenchmark performance comparison of GASNet-EX RMA on NERSC Cori, versus Cray MPI's RMA and message-passing.

D. Bonachea and P. H. Hargrove, "GASNet-EX: A High- Performance, Portable Communication Library for Exascale," in *Proceedings of Languages and Compilers for Parallel Computing (LCPC'18)*, ser. LNCS, vol. 11882. Springer, October 2018, doi:10.25344/S4QP4W.

Integration Timeline



Implement compiler feature prerequisites for building Caffeine

Integrate Caffeine into compiler

Parallel feature availability

Outline

- Introduction
- Methodology
- Discussion of Results
  - Status of parallel feature support
  - Compiler-facing interface
  - Unit tests
- Future Work
- Conclusions

Status

TABLE 2
STATUS OF CAFFEINE'S SUPPORT FOR THE PARALLEL FEATURES OF FORTRAN 2008 AND 2018.

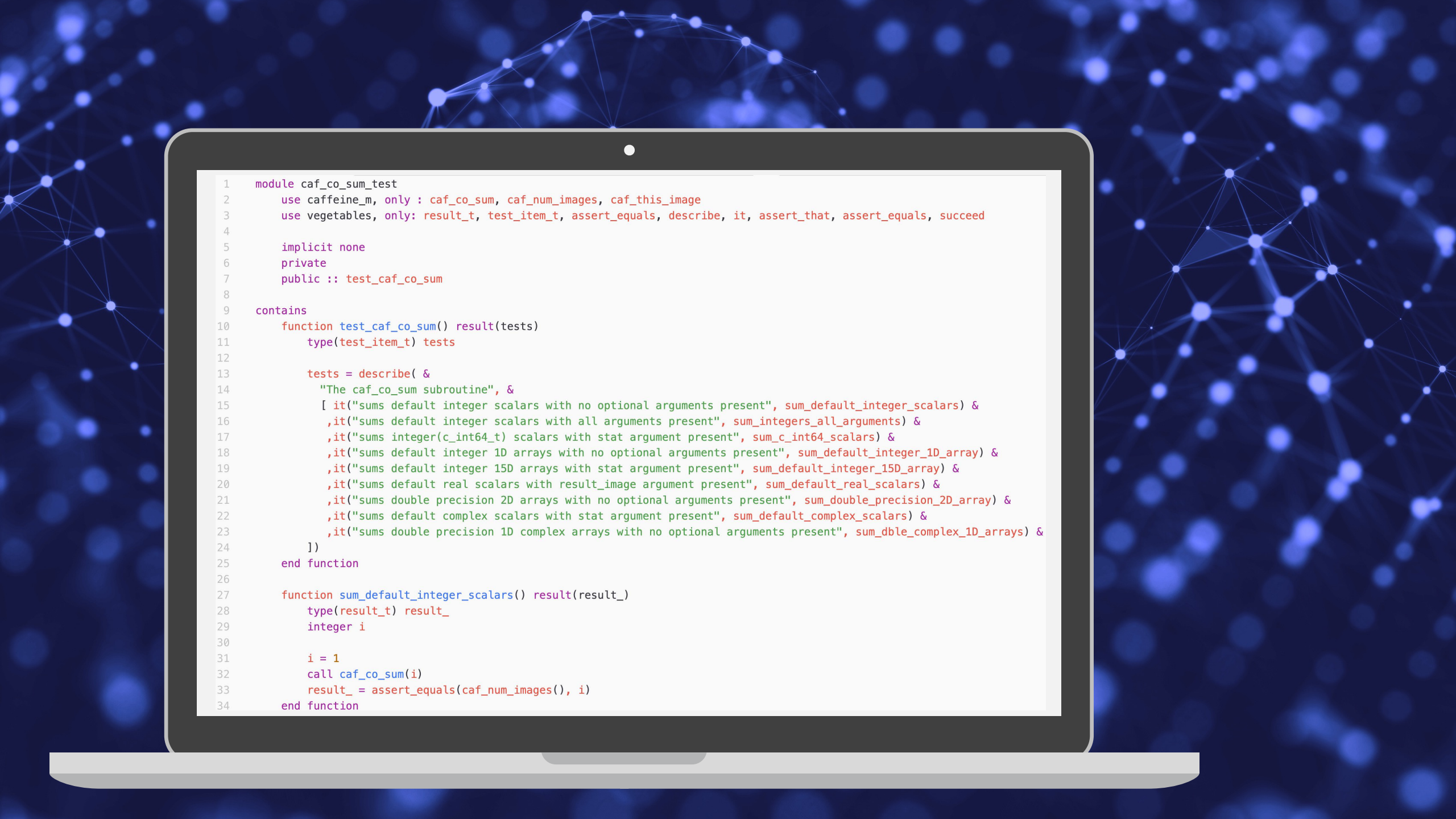
Standard	Feature	Status
2008	Program launch	yes
2008	Normal termination: `stop` and `end program` statements	yes
2008	Error termination: `error stop` statement	yes
2008	Image enumeration: `this_image` and `num_images` intrinsic functions	partial
2008	Synchronization: `sync {all,images,memory,team}` statements	partial
2008	Coarrays: declaration, access, (de)allocation, inquiry functions	WIP
2008	Critical construct: `critical` and `end critical`	WIP
2008	Atomics: `atomic_{int,logical}_kind` kind parameters and `atomic_{define,ref,...}` subroutines	WIP
2008	Locks: `lock` and `unlock` constructs	WIP
2018	Collective subroutines: `co_{broadcast,sum,min,max,reduce}`	yes
2018	Events: `event_type` intrinsic type, `event_query` subroutine and `event {post,wait}` statements	WIP
2018	Teams: `team_type` intrinsic type and `{form,change,end} team` statements	WIP
2018	Failed/stopped images: `fail image` statement, `{failed,stopped}_image` intrinsic functions, related constants	WIP

```
module subroutine caf_co_sum(a, result_image, stat, errmsg)
  implicit none
  type(*), intent(inout), contiguous, target :: a(..)
  integer, intent(in), target, optional :: result_image
  integer, intent(out), target, optional :: stat
  character(len=*), intent(inout), target, optional :: errmsg
end subroutine
```

type-/rank-agnostic argument

`caf_co_sum` calls a `bind(C)` function that receives a 2018-standard C struct `CFI_cdesc_t`

17



```
module caf_co_sum_test
    use caffeine_m, only : caf_co_sum, caf_num_images, caf_this_image
    use vegetables, only: result_t, test_item_t, assert_equals, describe, it, assert_that, assert_equals, succeed

    implicit none
    private
    public :: test_caf_co_sum

contains
    function test_caf_co_sum() result(tests)
        type(test_item_t) tests

        tests = describe( &
          "The caf_co_sum subroutine", &
          [ it("sums default integer scalars with no optional arguments present", sum_default_integer_scalars) &
           ,it("sums default integer scalars with all arguments present", sum_integers_all_arguments) &
           ,it("sums integer(c_int64_t) scalars with stat argument present", sum_c_int64_scalars) &
           ,it("sums default integer 1D arrays with no optional arguments present", sum_default_integer_1D_array) &
           ,it("sums default integer 15D arrays with stat argument present", sum_default_integer_15D_array) &
           ,it("sums default real scalars with result_image argument present", sum_default_real_scalars) &
           ,it("sums double precision 2D arrays with no optional arguments present", sum_double_precision_2D_array) &
           ,it("sums default complex scalars with stat argument present", sum_default_complex_scalars) &
           ,it("sums double precision 1D complex arrays with no optional arguments present", sum_dble_complex_1D_arrays) &
        ])
    end function

    function sum_default_integer_scalars() result(result_)
        type(result_t) result_
        integer i

        i = 1
        call caf_co_sum(i)
        result_ = assert_equals(caf_num_images(), i)
    end function
```

Outline

- Introduction
- Methodology
- Discussion of Results
- Future Work
- Conclusions

Future Work

- Coarray allocation and access:

```
real, allocatable :: coarray(:,:)[:,:,:]
allocate(coarray(10,10)[2,1,*])
if (this_image()==2) coarray = coarray(:,:)[1,1,1]
```

  - Follow OpenUH Coarray Fortran compiler design, which used GASNet-1
  - Use GASNet-EX RMA to for coarray access as a lightweight pass-through
- Teams of images: `team_type`, `form team`, `change team`, `end team`
- Atomics: `atomic_{int, logical}_kind` and `atomic_*` subroutines
- Events: `event_type`, `event post` and `event wait` statements, and `event_query` subroutine
- Critical blocks: `critical` and `end critical`
- Synchronization subsets: `sync images` and `sync team`

Conclusions

https://go.lbl.gov/caffeine

- Three key insights inspired Caffeine:
  1. A subset of *non-parallel* Fortran 2018 features provide a compelling capability for writing a mostly-Fortran runtime library to upport the *parallel* features.
  2. Using Fortran 2018 assumed-type dummy arguments obviates the need for passing compiler-specific data descriptors to the communication substrate.
  3. Using Fortran 2018 assumed-type, assumed-rank dummy arguments reduces line counts for some modules an order of magnitude relative to using the Fortran 2003 C-interoperability features.
- Writing a library in the language of the users improves sustainability by lowering barriers to community maintenance.
- We propose a workflow that leapfrogs standards to support parallel 2018 features before completing 2003 support.
- GASNet-EX offers a fully capable, high-performance communication substrate that supports a broad ecosystem of libraries and languages and the applications that use them.