Title

Just Write Fortran: Experiences with a Language-Based Alternative to MPI+X

Authors

Rouson, Damian

Dibba, Baboucarr

Rasmussen, Katherine

et al.

Publication Date

2024-11-17

DOI

10.25344/S4H88D

Peer reviewed

Just Write Fortran: Experiences with a Language-Based Alternative to MPI+X

Baboucarr Dibba, Katherine Rasmussen, Brad Richardson, Damian Rouson, David Torres, and Yunhao Zhang
Computer Languages and Systems Software (CLaSS) Group,
Lawrence Berkeley National Laboratory, Berkeley, California, USA
{bdibba,krasmussen,brad.richardson,rouson,davytorres,yzhang22}@lbl.gov

Ethan Gutmann
Research Applications Laboratory
National Center for Atmospheric Research, Boulder, Colorado, USA
gutmann@ucar.edu

Kareem Ergawy and Michael Klemm
Advanced Microdevices, Inc.
Munich, Germany
{michael.klemm,kareem.ergawy}@amd.com

Sameer Shende
Performance Research Laboratory, OACISS
University of Oregon, Eugene, Oregon
sameer@cs.uoregon.edu

***Index Terms*—Coarray Fortran, parallel programming, deep learning, high-performance computing, climate modeling.**

In a 2008 paper entitled "Parallel programming: can we PLEASE get it right this time?", Mattson et al. [1] wrote, "With few exceptions, only graduate students and other strange people write parallel software." Parallel programming had already started becoming more widespread in the research community with the 1995 publication on the first distributed-memory Beowulf clusters comprised of networked commodity personal computers [2]. Shared-memory parallelism proliferated in the mid-2000s when the multicore processors first proposed a decade prior [3] reached commodity status contemporaneously with the advent of general-purpose computation on graphics processing units (GPGPUs) [4]. With these hardware trends democratizing parallel computing, the timeliness of the 1996 Message Passing Interface (MPI) specification [5] and the 1997 OpenMP specification explain the widespread use of programming models defined outside of programming languages. But it no longer has to be this way!

Mattson et al. called for a simpler parallelization paradigm: "An ideal solution would automatically exploit concurrency through techniques such as... automatic parallelization of loops." Fortran 2008 [6] answered this call with `do concurrent` and also supported distributed-memory parallelism by incorporating aspects of the Co-Array Fortran language developed in 1996 by Numrich and Reid [7], who stated, "The underlying philosophy of our design is to make the smallest number of changes to the language required to obtain a robust and efficient parallel language without requiring the programmer to learn very many new rules."

Fortran 2023 greatly expands the parallel feature set. The Cray®, Intel®, LFortran, LLVM®, and NVIDIA® compilers automatically parallelize `do concurrent`. The Cray, Intel, GNU, and NAG compilers support coarrays. Thus, language-based parallelism is emerging as a portable alternative to extra-language programming models.

This talk will present experiences with the automatic parallelization of `do concurrent` in the Fortran 2023 deep learning library Fiats[1] and coarray communication in the Intermediate Complexity Atmospheric Research (ICAR) model[2], respectively.

References

[1] T. Mattson and M. Wrinn, "Parallel programming: can we please get it right this time?" in *Proceedings of the 45th annual design automation conference*, 2008, pp. 7–11.

[2] D. J. Becker, T. Sterling, D. Savarese, J. E. Dorband, U. A. Ranawak, and C. V. Packer, "Beowulf: A parallel workstation for scientific computation," in *Proceedings, international conference on parallel processing*, vol. 95, 1995, pp. 11–14.

[3] K. Olukotun, B. A. Nayfeh, L. Hammond, K. Wilson, and K. Chang, "The case for a single-chip multiprocessor," *ACM Sigplan Notices*, vol. 31, no. 9, pp. 2–11, 1996.

[4] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell, "A survey of general-purpose computation on graphics hardware," in *Computer graphics forum*, vol. 26, no. 1. Wiley Online Library, 2007, pp. 80–113.

[5] D. W. Walker and J. J. Dongarra, "Mpi: a standard message passing interface," *Supercomputer*, vol. 12, pp. 56–68, 1996.

[6] Fortran Standards Committee JTC1/SC22/WG5, *Information technology — Programming languages — Fortran, ISO/IEC 1539-1:2010*. International Organization for Standardization (ISO), Oct 2010, https://www.iso.org/standard/50459.html.

[7] R. W. Numrich and J. Reid, "Co-array fortran for parallel programming," in *ACM Sigplan Fortran Forum*, vol. 17, no. 2. ACM New York, NY, USA, 1998, pp. 1–31.

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

[1] https://go.lbl.gov/fiats
[2] https://github.com/berkeleylab/icar

Just Write Fortran:

Experiences with a Language-Based Alternative to MPI+X

Damian Rouson, Baboucarr Dibba, Katherine Rasmussen, Brad Richardson, David Torres, Yunhao Zhang
Berkeley Lab

Ethan Gutmann
NCAR

Kareem Ergawy, Michael Klemm
Advanced Micro Devices, Inc.

Sameer Shende
University of Oregon

Parallel Applications Workshop — Alternatives to MPI+X (PAW-ATM), 17 November 2024

Overview

Just Write Fortran:

Background

2.2

Parallel Programming: Can we PLEASE get it right this time?

Tim Mattson
Intel Corporation
Dupont, WA
timothy.g.mattson@intel.com

Michael Wrinn
Intel Corporation
Hillsboro, OR
michael.wrinn@intel.com

ABSTRACT
The computer industry has a problem. As Moore's law marches on, it will be exploited to double cores, not frequencies. But all those cores, growing to 8, 16 and beyond over the next several

and their programming environments. A common theme is that just about every mistake we could make has already been made by someone. So rather than reinvent these mistakes, let's learn from the past and "do it right this time".

1. INTRODUCTION
The computer industry has enjoyed a 30-year "value ramp", where Moore's law drove processor improvements, which in turn drove

approach, to one which incorporates concurrency?

2. PARALLEL PROGRAMMING: LESSONS FROM HISTORY

systems give more opportunity for development, thus creating a demand for better tools, making the task more manageable, in turn drawing in more developers, all in a virtuous feedback cycle. While this may happen, it rests on the interesting premise that 25 years of PhD-level work on parallel systems was insufficiently diligent. We propose to understand this preceding "massively parallel programming" era as one of exploration, trial-and-error, replete with insights as to what works, and especially, what does not. Following are some lessons from history, to guide work as the industry shift to the "manycore parallel" era.

> "With few exceptions, only graduate students and other strange people write parallel software."

> "An ideal solution would automatically exploit concurrency through techniques such as...automatic parallelization of loops."

T. Mattson and M. Wrinn (2008)
Proceedings of the 45th annual design automation conference, pp. 7–11.

Fortran 2023 Parallelism

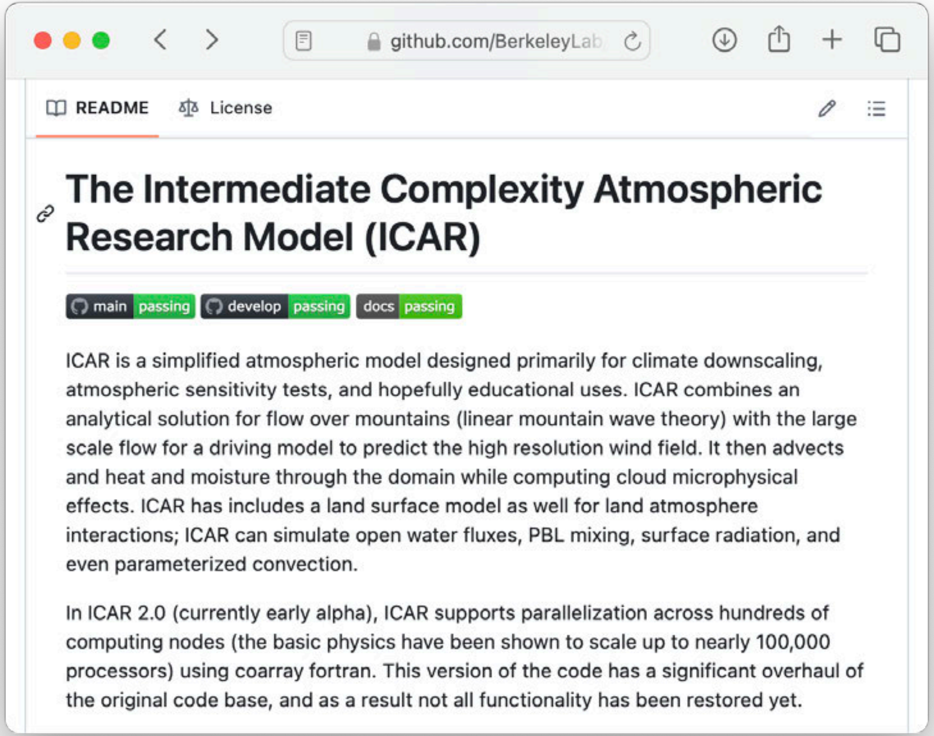
BERKELEY LAB
Bringing Science Solutions to the World

Multi-Image Execution ("Coarray Fortran"):

- Coarrays
- Synchronization
- Events
- Notifications
- Locks
- Failed images
- Teams
- Critical sections
- Intrinsic functions
- Collective subroutines
- Atomic subroutines
- Types
- Values
- Atomic kind type parameters

Gutmann, E. D., I. Barstad, M. P. Clark, J. R. Arnold, and R. M. Rasmussen (2016), *The Intermediate Complexity Atmospheric Research Model*, J. Hydrometeor, doi:10.1175/JHM-D-15-0155.1.

Application:



Berkeley Lab fork (neural-net branch): go.lbl.gov/icar

Fortran 2023 Parallelism

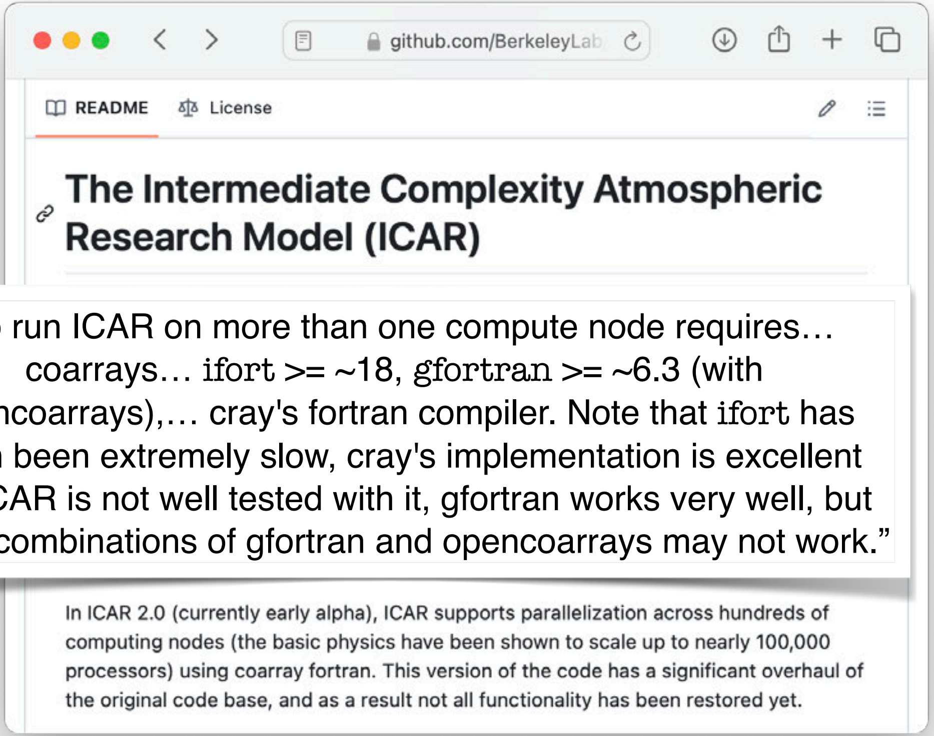
Multi-Image Execution ("Coarray Fortran"):

- Coarrays
- Synchronization
- Events
- Notifications
- Locks
- Failed images
- Teams
- Critical sections
- Intrinsic functions
- Collective subroutines
- Atomic subroutines
- Types
- Values
- Atomic kind type parameters

Gutmann, E. D., I. Barstad, M. P. Clark, J. R. Arnold, and R. M. Rasmussen (2016), *The Intermediate Complexity Atmospheric Research Model*, J. Hydrometeor, doi:[10.1175/JHM-D-15-0155.1](https://doi.org/10.1175/JHM-D-15-0155.1).

BERKELEY LAB
Bringing Science Solutions to the World

Application:


github.com/BerkeleyLab

README License

The Intermediate Complexity Atmospheric Research Model (ICAR)

In ICAR 2.0 (currently early alpha), ICAR supports parallelization across hundreds of computing nodes (the basic physics have been shown to scale up to nearly 100,000 processors) using coarray fortran. This version of the code has a significant overhaul of the original code base, and as a result not all functionality has been restored yet.

"To run ICAR on more than one compute node requires… coarrays… `ifort` >= ~18, `gfortran` >= ~6.3 (with opencoarrays),… cray's fortran compiler. Note that `ifort` has often been extremely slow, cray's implementation is excellent but ICAR is not well tested with it, gfortran works very well, but some combinations of gfortran and opencoarrays may not work."

Berkeley Lab fork (neural-net branch): go.lbl.gov/icar

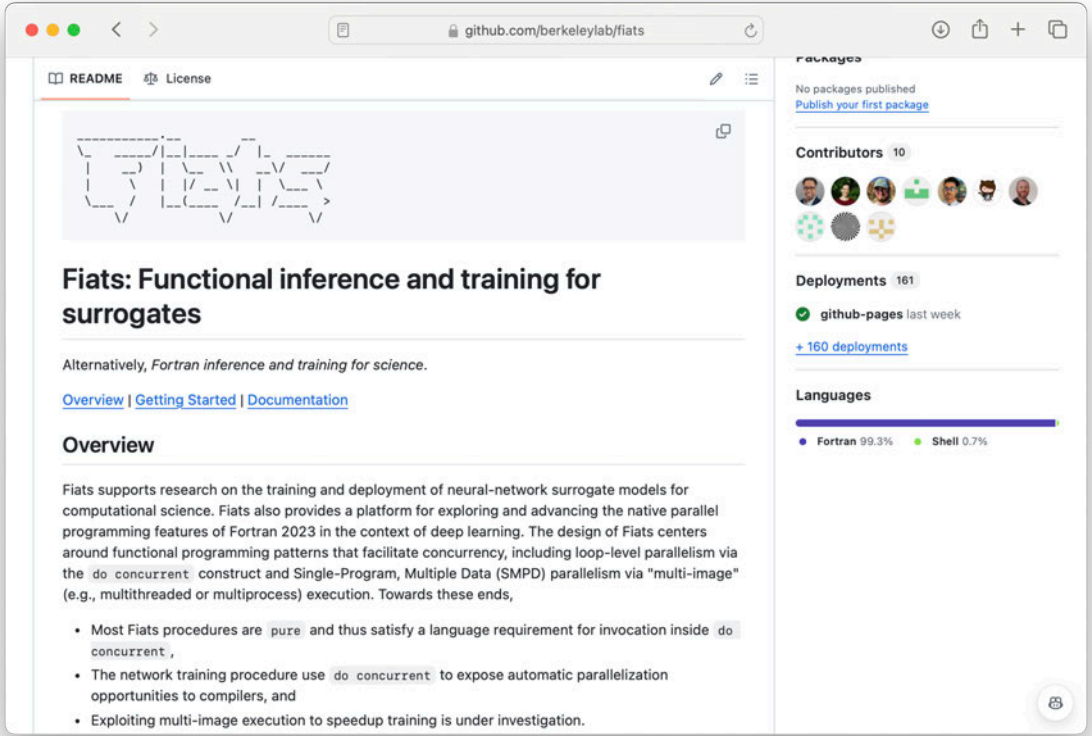
Fortran 2023 Parallelism

BERKELEY LAB
Bringing Science Solutions to the World

Statement-/Construct Parallelism:

- do concurrent
  - pure procedures
- Array statements
  - elemental procedures
  - Intrinsic functions: matmul, pack, …

Library:



go.lbl.gov/fiats

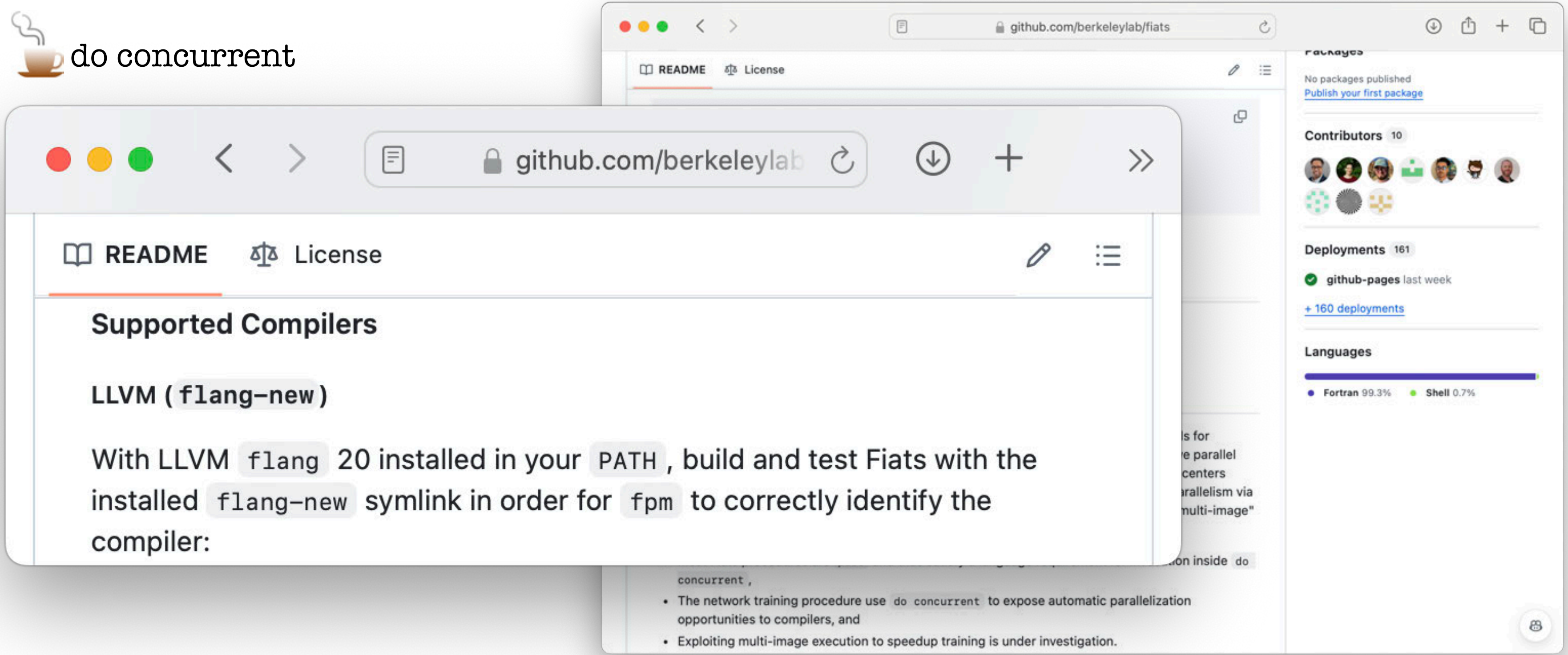
Fortran 2023 Parallelism

BERKELEY LAB
Bringing Science Solutions to the World

Statement-/Construct Parallelism:

Library:

do concurrent



go.lbl.gov/fiats

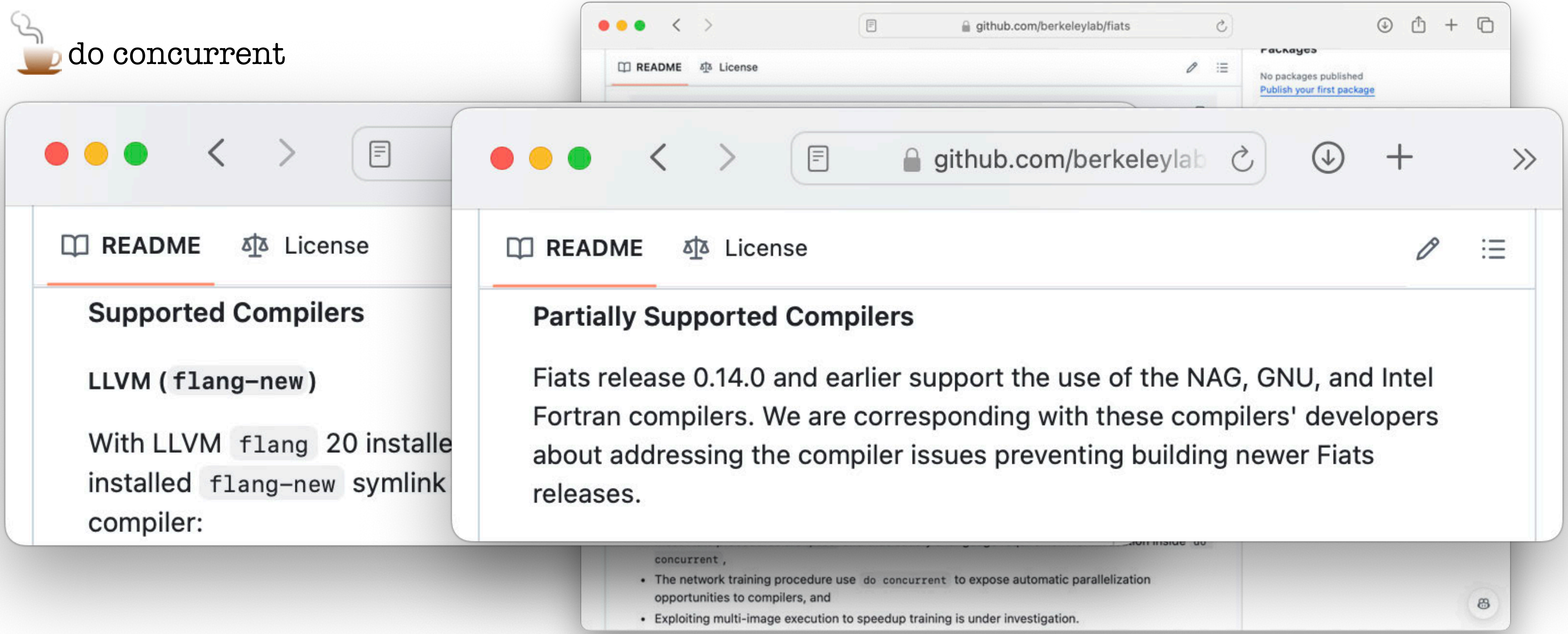
Fortran 2023 Parallelism

BERKELEY LAB
Bringing Science Solutions to the World

Statement-/Construct Parallelism:

☕ do concurrent

Library:



go.lbl.gov/fiats

Compiler Status

BERKELEY LAB
Bringing Science Solutions to the World

Multi-Image Execution:

- Cray
- Intel
- GNU + OpenCoarrays
- NAG
- LLVM Flang:
  - **Complete:** Parses parallel syntax
  - **Recently launched:** Lowering to PRIF calls
  - **In review:** PRIF 0.4 Design Document
  - **Under development:** Caffeine parallel runtime library

Automatic Parallelization of `do concurrent`:

- NVIDIA: CPU, GPU
- Intel: CPU, GPU
- Cray: CPU, GPU
- LFortran: CPU
- LLVM Flang: CPU (GPU under development)

The World’s Shortest Bug Reproducer

```
end
```

Fiats: Inference

example — vim concurrent-inferences.f90 — 65×5

```fortran
      do concurrent(i=1:lat, k=1:lev, j=1:lon)
        outputs(i,k,j) = neural_network%infer(inputs(i,k,j))
      end do

```

example — vim concurrent-inferences.f90 — 70×10

```fortran
      !$omp parallel do shared(inputs,outputs)
      do j=1,lon
        do k=1,lev
          do i=1,lat
            outputs(i,k,j) = neural_network%infer(inputs(i,k,j))
          end do
        end do
      end do
      !$omp end parallel do
```

example — vim concurrent-inferences.f90 — 50×5

```fortran
      !$omp workshare
      outputs = neural_network%infer(inputs)
      !$omp end workshare

                                        73,1          64%
```

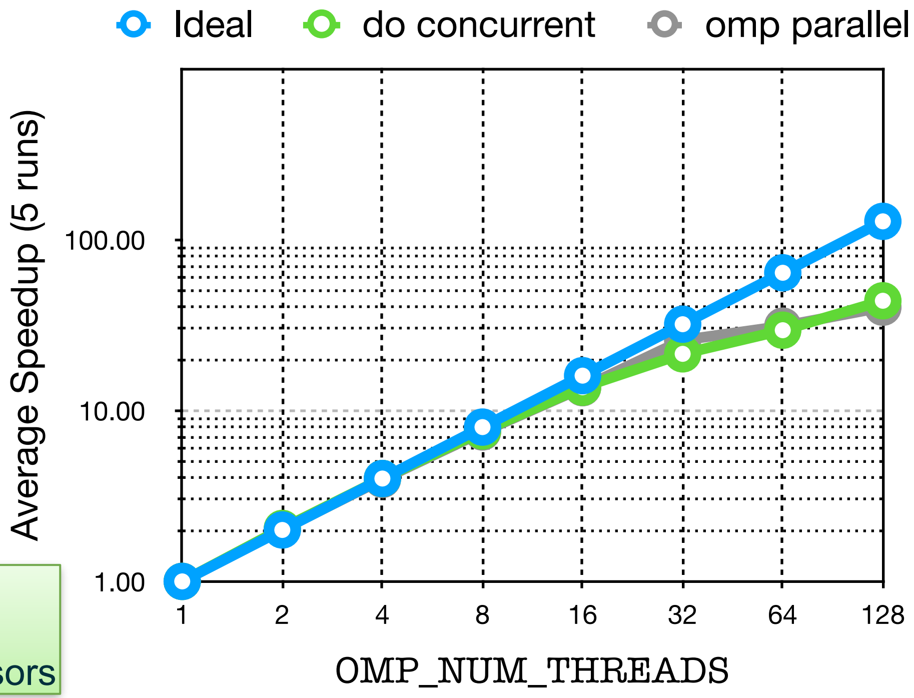
CPU Parallelism on Perlmutter

Compiler:
Berkeley Lab [llvm-project fork]
git tag paw-atm24-fiats
Commits pulled from ROCm fork

Neural network:
- Activation function: GELU
- Numbers of inputs: 80
- Number of outputs: 31
- Nodes per hidden layer: 256, 384, 256

Source: Z. Bai

Platform:
Dedicated interactive node
2x 64-core AMD EPYC 7663 processors



Example Command:
```
OMP_NUM_THREADS = 128  fpm run --example concurrent-inferences \
   --runner "srun --cpu_bind=cores -c 128 -n 1" -- --network model.json
```

Fiats: Training

fiats — vim neural_network_s.F90 — 101×36

```
#if F2023_LOCALITY
       iterate_through_batch: &
       do concurrent (pair = 1:mini_batch_size) local(a,z,delta) reduce(+: dcdb, dcdw)

#elif F2018_LOCALITY

       reduce_gradients: &
       block
         real reduce_dcdb(size(dcdb,1),size(dcdb,2),mini_batch_size)
         real reduce_dcdw(size(dcdw,1),size(dcdw,2),size(dcdw,3),mini_batch_size)
         reduce_dcdb = 0.
         reduce_dcdw = 0.

         iterate_through_batch: &
         do concurrent (pair = 1:mini_batch_size) local(a,z,delta)

#else

       reduce_gradients: &
       block
         real reduce_dcdb(size(dcdb,1),size(dcdb,2),mini_batch_size)
         real reduce_dcdw(size(dcdw,1),size(dcdw,2),size(dcdw,3),mini_batch_size)
         reduce_dcdb = 0.
         reduce_dcdw = 0.

         iterate_through_batch: &
         do concurrent (pair = 1:mini_batch_size)

           iteration: &
           block

             real a(maxval(self%nodes_), input_layer:output_layer) ! Activations
             real z(size(b,1),size(b,2)), delta(size(b,1),size(b,2))
#endif

```

910,0-1 89%

Conclusions and Future Work

Conclusions

☕ Fortran 2023 provides a language-based alternative to MPI+x in the form of multi-image execution + statement/construct-level parallelism.

☕ Non-overlapping sets of compilers support either or both forms of parallelism.

☕ Compiler implementations still vary in maturity and robustness, but the LLVM Flang CPU parallelization results are encouraging.

Future Work

☕ Inference and training on GPUs: offloading vs embedded

☕ Multi-image training

☕ Ongoing development of AMD Next-Gen Fortran compiler: blog.

Acknowledgements

The Berkeley Lab Fortran Team

Dan Bonachea, Hugh Kadhem, Brad Richardson, Kate Rasmussen

Collaborators

Fiats: Zhe Bai, Jeremy Bailey, David Torres, Kareem Jabbar Weaver, Jordan Welsman, Yunhao Zhang

LLVM Flang: Jeff Hammond, Jean-Didier Paillex, Etienne Renault

OpenCoarrays: Izaak Beekman, Tobias Burnus, Alessandro Fanfarillo, Andre Vehreschild

ICAR: Ethan Gutmann

TAU: Sameer Shende