

Lawrence Berkeley National Laboratory

LBL Publications

Title

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

Permalink

<https://escholarship.org/uc/item/1zf6h82v>

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 Fiats¹ and coarray communication in the Intermediate Complexity Atmospheric Research (ICAR) model², 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>



BERKELEY LAB

Bringing Science Solutions to the World



Office of Science

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:

01

Motivation

02

Background:
Parallelism in
Fortran 2023

03

User Experience:
Fun with Compilers

04

Discussion of Results

05

Conclusions and Future Work



BERKELEY LAB

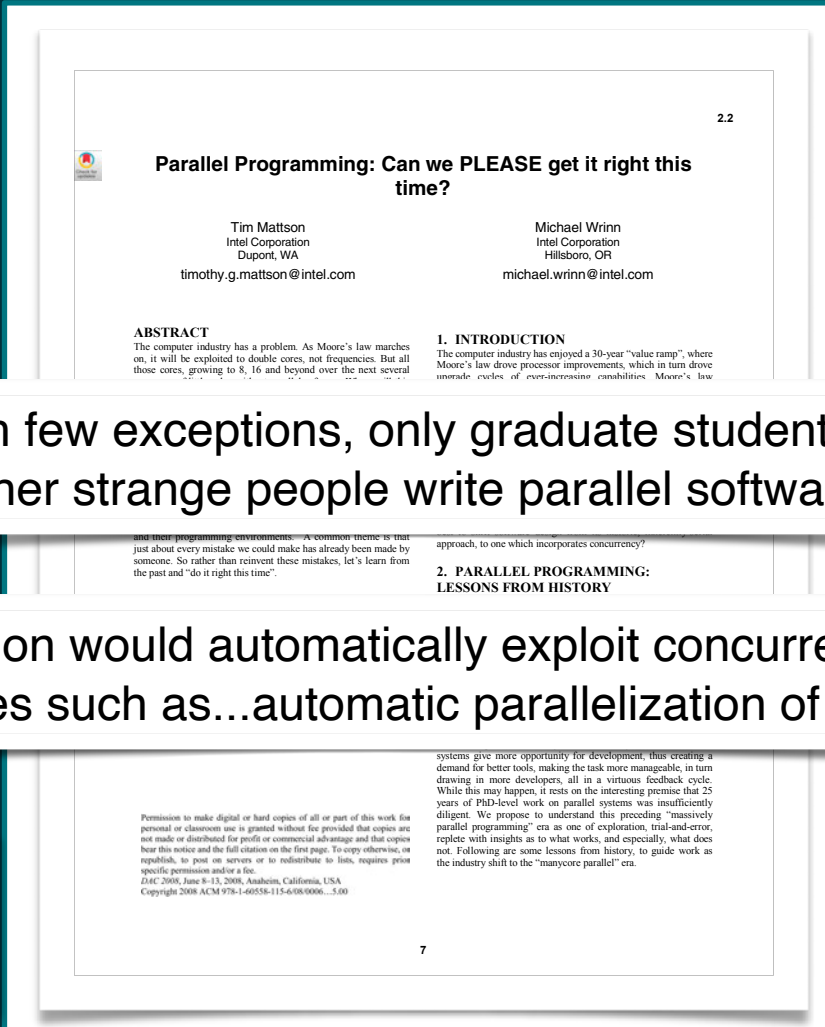
Bringing Science Solutions to the World

Background



U.S. DEPARTMENT OF
ENERGY

Office of Science



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

1. INTRODUCTION

The computer industry has enjoyed a 30-year "value ramp", where Moore's law drove processor improvements, which in turn drove monadic cycles of ever-increasing capabilities. Moore's law

“With few exceptions, only graduate students and other strange people write parallel software.”

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".

approach, to one which incorporates concurrency?

2. PARALLEL PROGRAMMING: LESSONS FROM HISTORY

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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
EAC 2008, June 8-13, 2008, Anaheim, California, USA
Copyright 2008 ACM 978-1-60558-115-6/08/0006...5.00

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.

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

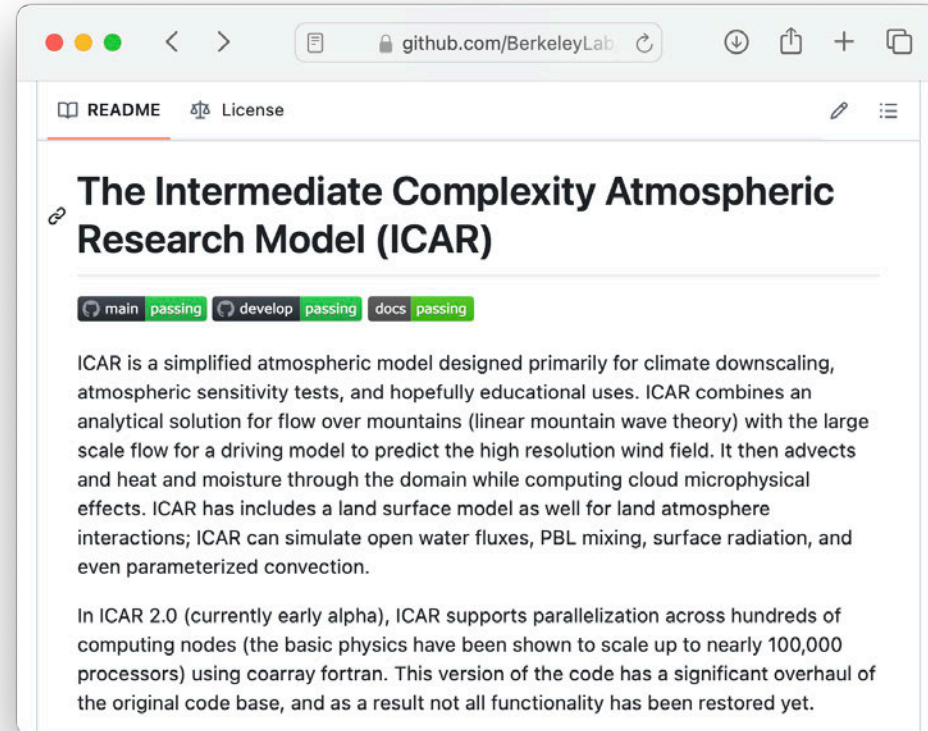
Fortran 2023 Parallelism

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. Hydrometeorol, doi:[10.1175/JHM-D-15-0155.1](https://doi.org/10.1175/JHM-D-15-0155.1).

Application:



The screenshot shows a web browser displaying the GitHub repository page for 'The Intermediate Complexity Atmospheric Research Model (ICAR)'. The page title is 'The Intermediate Complexity Atmospheric Research Model (ICAR)'. Below the title, there are three status bars: 'main passing', 'develop passing', and 'docs passing'. The main content area contains a paragraph describing the ICAR model: 'ICAR is a simplified atmospheric model designed primarily for climate downscaling, atmospheric sensitivity tests, and hopefully educational uses. ICAR combines an analytical solution for flow over mountains (linear mountain wave theory) with the large scale flow for a driving model to predict the high resolution wind field. It then advects and heat and moisture through the domain while computing cloud microphysical effects. ICAR has includes a land surface model as well for land atmosphere interactions; ICAR can simulate open water fluxes, PBL mixing, surface radiation, and even parameterized convection.' Below this paragraph, there is another paragraph: '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.'

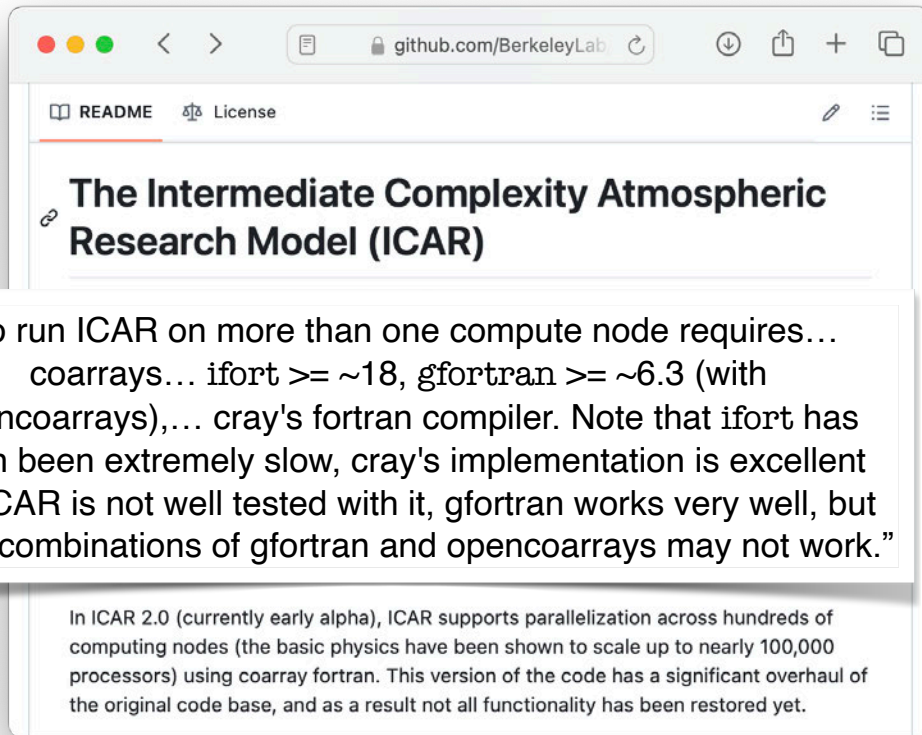
Fortran 2023 Parallelism

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).


Application:



The screenshot shows a browser window displaying the GitHub repository for 'The Intermediate Complexity Atmospheric Research Model (ICAR)'. The page title is 'The Intermediate Complexity Atmospheric Research Model (ICAR)'. Below the title, there is a quote: "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." Below the quote, there is a paragraph of text: "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."

Fortran 2023 Parallelism

Statement-/Construct Parallelism:

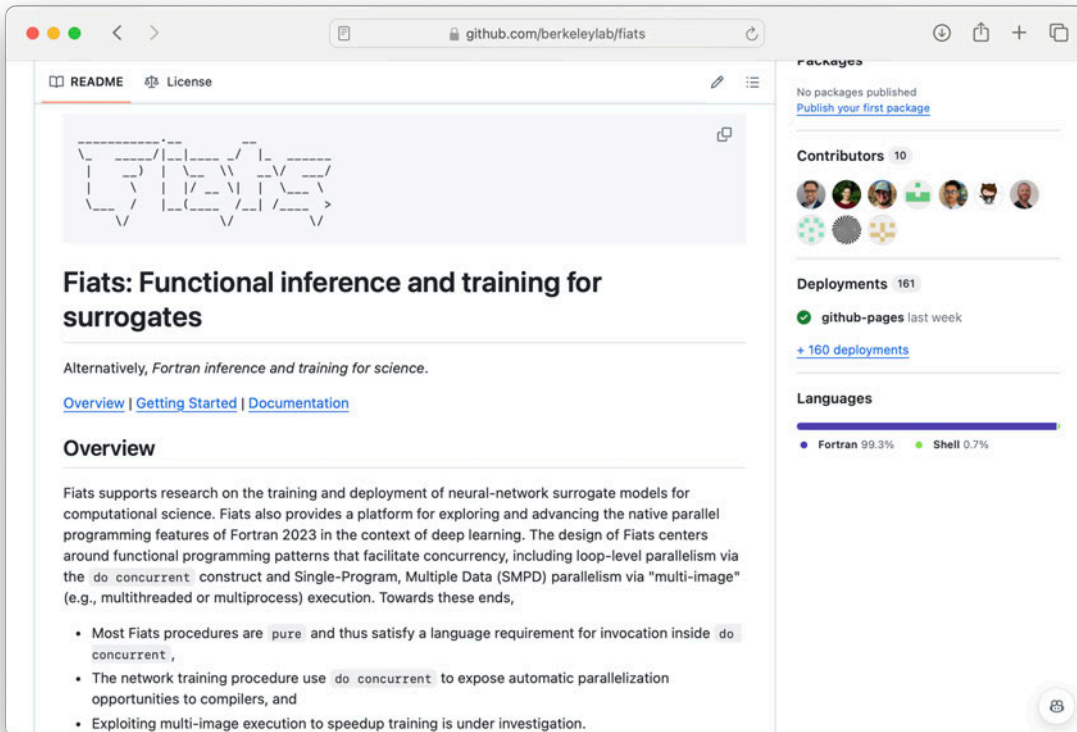
 do concurrent

- pure procedures

 Array statements

- elemental procedures
- Intrinsic functions:
matmul, pack, ...

Library:



The screenshot shows the GitHub repository page for 'berkeleylab/fiats'. The main content area displays the repository name, a README section with a diagram of a neural network, and an overview section. The overview section describes the library's purpose and features, including support for neural-network surrogate models and parallel programming features of Fortran 2023. A list of bullet points highlights key features and research directions.

Fiats: Functional inference and training for surrogates

Alternatively, *Fortran inference and training for science*.

[Overview](#) | [Getting Started](#) | [Documentation](#)

Overview

Fiats supports research on the training and deployment of neural-network surrogate models for computational science. Fiats also provides a platform for exploring and advancing the native parallel programming features of Fortran 2023 in the context of deep learning. The design of Fiats centers around functional programming patterns that facilitate concurrency, including loop-level parallelism via the `do concurrent` construct and Single-Program, Multiple Data (SMPD) parallelism via "multi-image" (e.g., multithreaded or multiprocess) execution. Towards these ends,

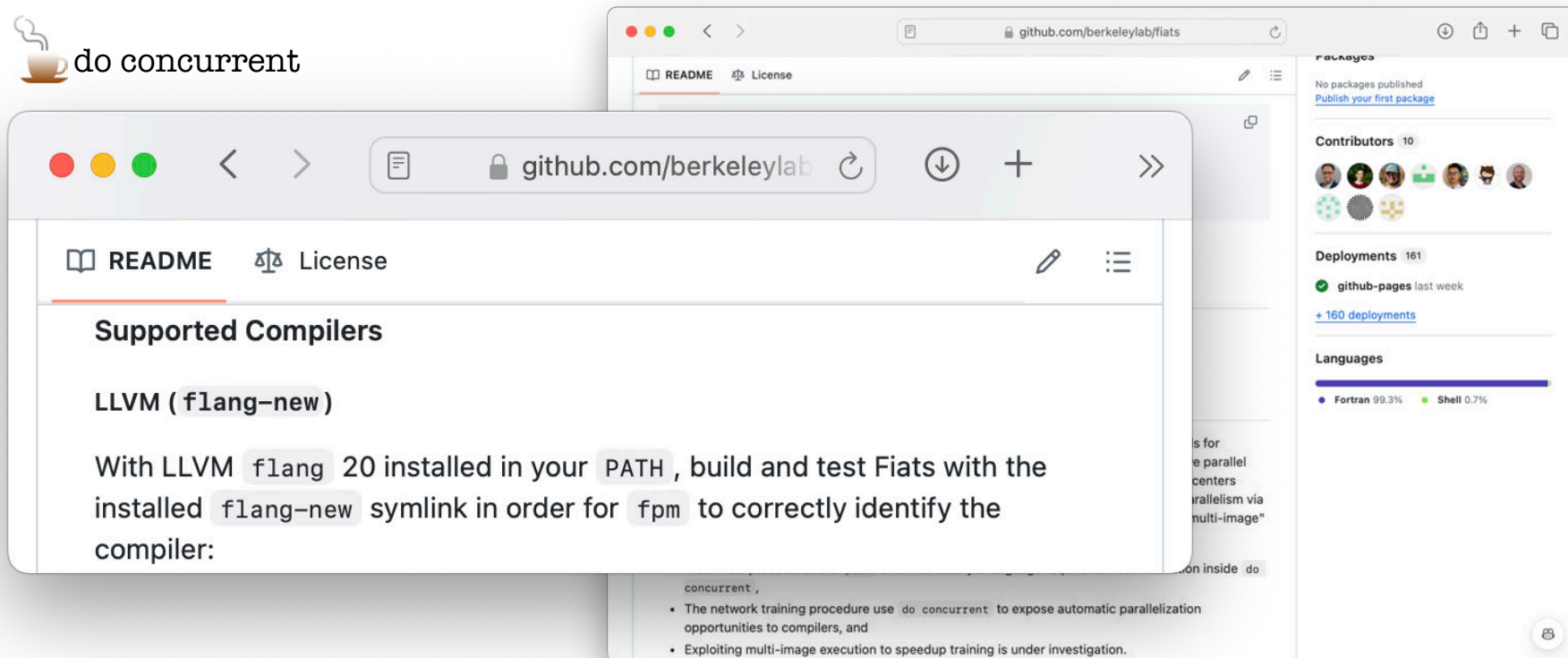
- Most Fiats procedures are `pure` and thus satisfy a language requirement for invocation inside `do concurrent`,
- The network training procedure use `do concurrent` to expose automatic parallelization opportunities to compilers, and
- Exploiting multi-image execution to speedup training is under investigation.

Fortran 2023 Parallelism

Statement-/Construct Parallelism:

 do concurrent

Library:



The screenshot shows a GitHub repository page for 'berkeleylab/fiats'. The main content is the README, which includes the following text:

Supported Compilers

LLVM (flang-new)


With LLVM flang 20 installed in your PATH , build and test Fiats with the installed flang-new symlink in order for fpm to correctly identify the compiler:

do concurrent ,

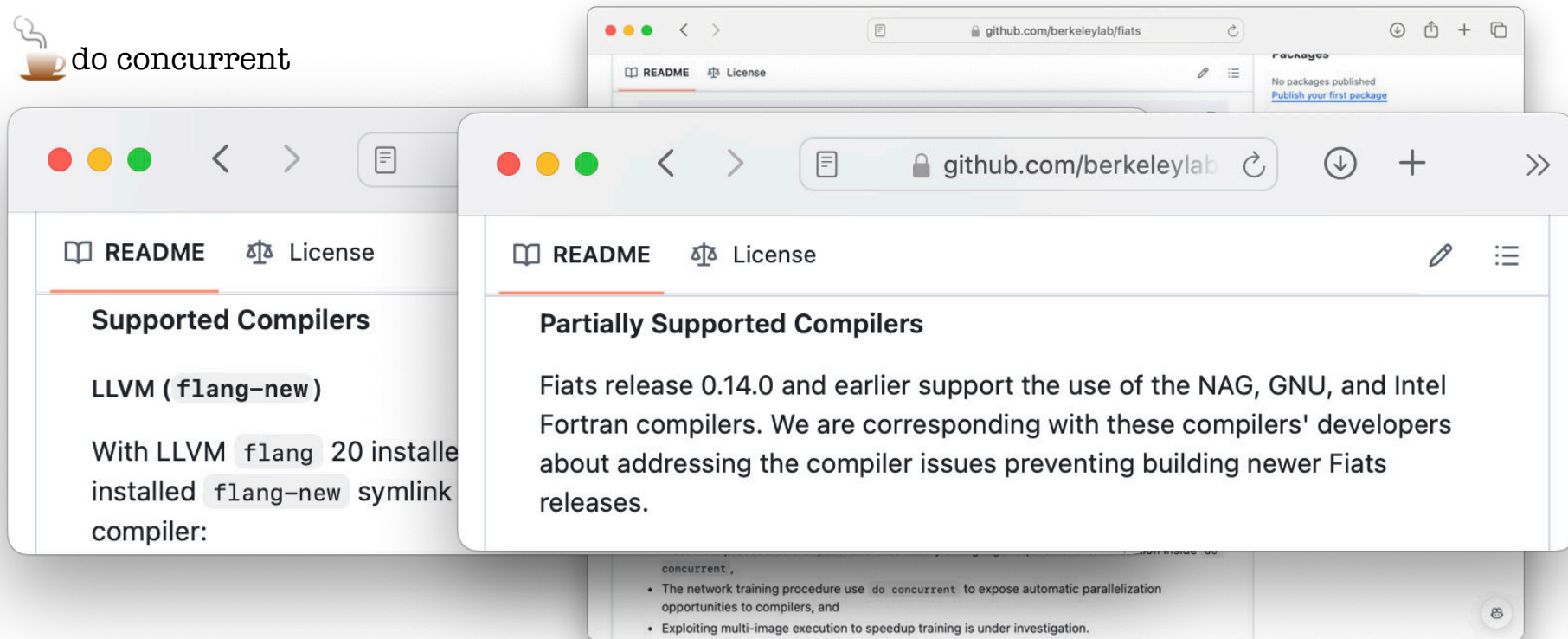
- The network training procedure use do concurrent to expose automatic parallelization opportunities to compilers, and
- Exploiting multi-image execution to speedup training is under investigation.

Fortran 2023 Parallelism

Statement-/Construct Parallelism:

 do concurrent

Library:



The image shows a screenshot of the GitHub repository for the 'fiats' library. The browser address bar shows 'github.com/berkeleylab/fiats'. The repository page is open to the README file. The 'Supported Compilers' section is visible, listing LLVM (flang-new) and providing instructions for installation. The 'Partially Supported Compilers' section is also visible, stating that Fiats release 0.14.0 and earlier support the use of the NAG, GNU, and Intel Fortran compilers, and that the team is working on addressing compiler issues.

Supported Compilers

LLVM (flang-new)

With LLVM flang 20 installed
installed flang-new symlink
compiler:

Partially Supported Compilers

Fiats release 0.14.0 and earlier support the use of the NAG, GNU, and Intel Fortran compilers. We are corresponding with these compilers' developers about addressing the compiler issues preventing building newer Fiats releases.

- The network training procedure use do concurrent to expose automatic parallelization opportunities to compilers, and
- Exploiting multi-image execution to speedup training is under investigation.

Compiler Status

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 — 65x5
50 do concurrent(i=1:lat, k=1:lev, j=1:lon)
51     outputs(i,k,j) = neural_network%infer(inputs(i,k,j))
52 end do
53
```

```
example — vim concurrent-inferences.f90 — 70x10
59 !$omp parallel do shared(inputs,outputs)
60 do j=1,lon
61     do k=1,lev
62         do i=1,lat
63             outputs(i,k,j) = neural_network%infer(inputs(i,k,j))
64         end do
65     end do
66 end do
67 !$omp end parallel do
```

```
example — vim concurrent-inferences.f90 — 50x5
73 !$omp workshare
74 outputs = neural_network%infer(inputs)
75 !$omp end workshare
76
```

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

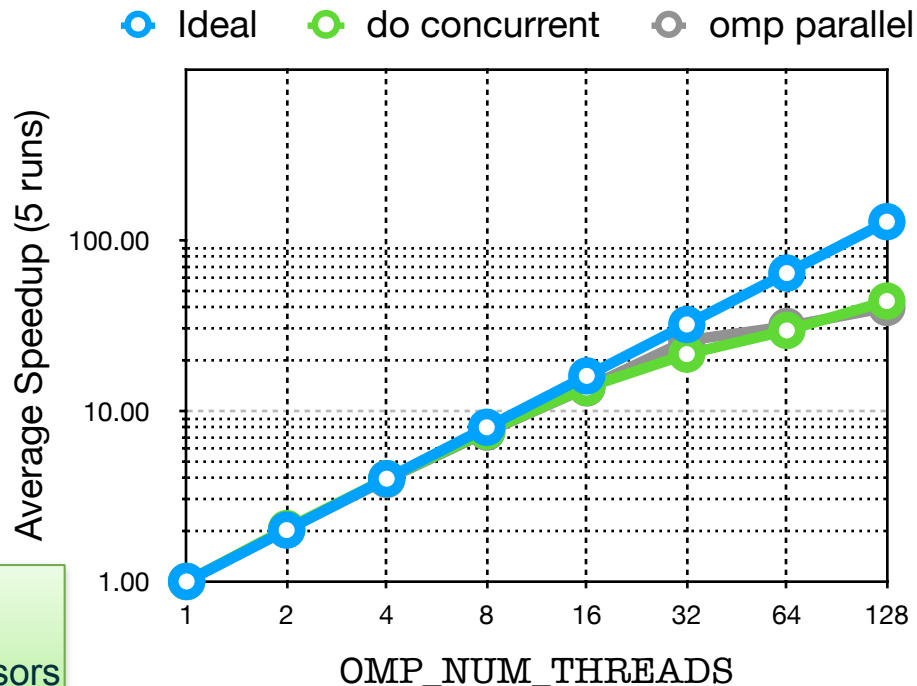
Platform:

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 — 101x36
907 #if F2023_LOCALITY
908     iterate_through_batch: &
909     do concurrent (pair = 1:mini_batch_size) local(a,z,delta) reduce(+: dcdb, dcdw)
910
911 #elif F2018_LOCALITY
912
913     reduce_gradients: &
914     block
915         real reduce_dcdb(size(dcdb,1),size(dcdb,2),mini_batch_size)
916         real reduce_dcdw(size(dcdw,1),size(dcdw,2),size(dcdw,3),mini_batch_size)
917         reduce_dcdb = 0.
918         reduce_dcdw = 0.
919
920     iterate_through_batch: &
921     do concurrent (pair = 1:mini_batch_size) local(a,z,delta)
922
923 #else
924
925     reduce_gradients: &
926     block
927         real reduce_dcdb(size(dcdb,1),size(dcdb,2),mini_batch_size)
928         real reduce_dcdw(size(dcdw,1),size(dcdw,2),size(dcdw,3),mini_batch_size)
929         reduce_dcdb = 0.
930         reduce_dcdw = 0.
931
932     iterate_through_batch: &
933     do concurrent (pair = 1:mini_batch_size)
934
935         iteration: &
936         block
937
938             real a(maxval(self%nodes_), input_layer:output_layer) ! Activations
939             real z(size(b,1),size(b,2)), delta(size(b,1),size(b,2))
940 #endif
941
```

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 Kadhemi, 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