Leveraging and Expanding the Capabilities of the LLVM Compiler Infrastructure for Exascale Computing

Patrick McCormick

Extreme-scale Scientific Software Stack Forum (E4S Forum)
Albuquerque, NM
September 23, 2019

Managed by Triad National Security, LLC for the U.S. Department of Energy's NNSA

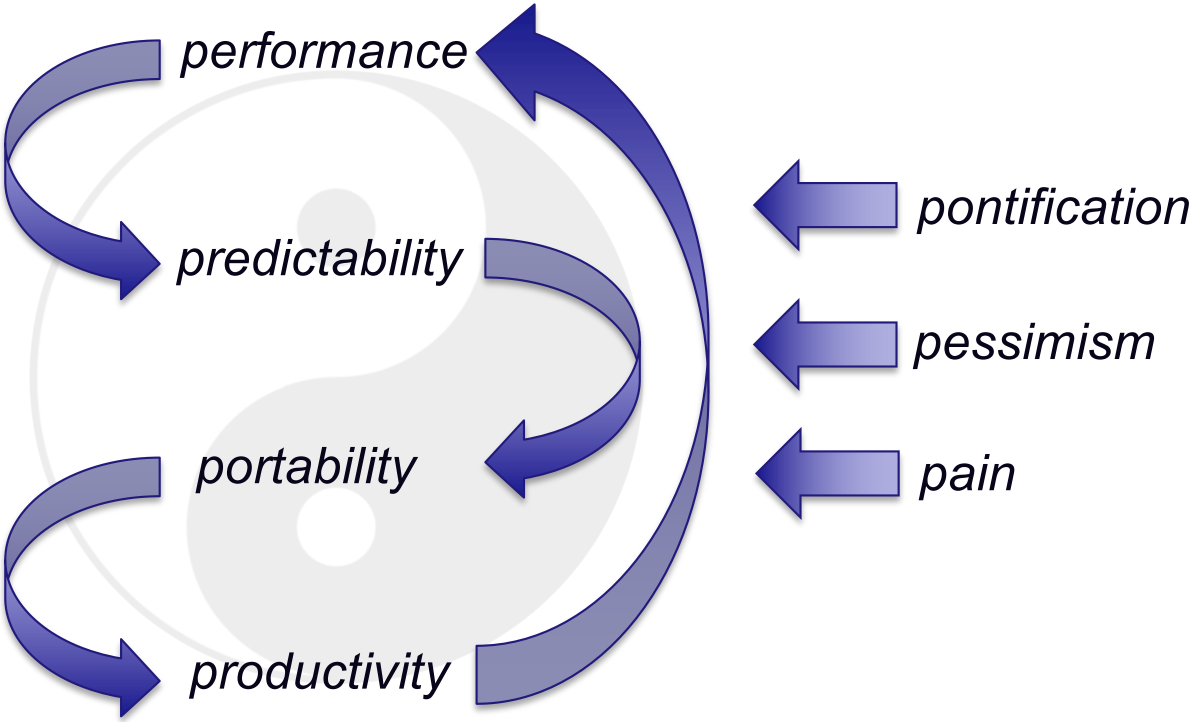
Why LLVM?

- *Modular, well designed, mature compiler infrastructure*
  - *Provides foundation for many commercial products*
- *Very strong community across industry, academia, and the labs*
  - *Great vehicle for collaborations*
- *Well defined path for experimentation, testing, adoption, and deployment*
  - *Caveats: Like any other broad community effort there are lots of external drivers and priorities for successfully adoption*

ECP's LLVM-based Projects

- *Many… More than I have time to present in detail today…*
- *Focus areas:*
  - *Auto-tuning, OpenMP, optimizations for exascale architectures, extensions and additions to LLVM, Fortran ("flang"), tooling…*
- *Rest of the talk looks at just a subset*
  - *Primary focus around intermediate representations and augmenting LLVM's infrastructure.*

The 3… No, 4… No, 7 P's



performance

predictability

portability

productivity

pontification

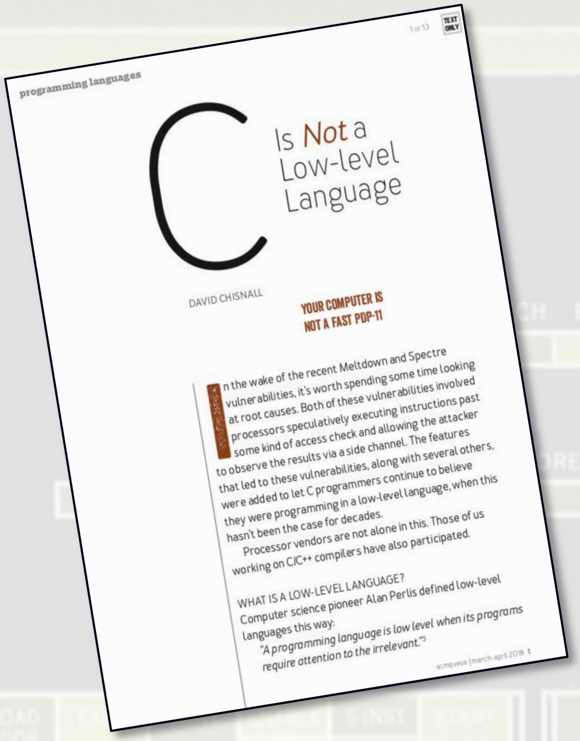
pessimism

pain

What model do we use for programming?

Model:

- A "fast PDP-11" *(Sans switches)*
  - A facade brought to you by:
    - *Hardware complexity*
    - *A complex compiler* : thousands of lines of code in Clang+LLVM related to hiding details
    - Longevity boosted by Moore's Law & Dennard Scaling
- *A parallel, latency-hiding processor that is obscured, but not necessarily hidden, by an abstract, sequential machine model and the compiler.*



programming languages

1 OF 13

TEXT ONLY

C Is *Not* a Low-level Language

DAVID CHISNALL

YOUR COMPUTER IS NOT A FAST PDP-11

In the wake of the recent Meltdown and Spectre vulnerabilities, it's worth spending some time looking at root causes. Both of these vulnerabilities involved processors speculatively executing instructions past some kind of access check and allowing the attacker to observe the results via a side channel. The features that led to these vulnerabilities, along with several others, were added to let C programmers continue to believe they were programming in a low-level language, when this hasn't been the case for decades.

Processor vendors are not alone in this. Those of us working on C/C++ compilers have also participated.

WHAT IS A LOW-LEVEL LANGUAGE?

Computer science pioneer Alan Perlis defined low-level languages this way:

"A programming language is low level when its programs require attention to the irrelevant."[5]

acmqueue | march-april 2018 1

David Chisnall: ***C Is Not a Low-level Language***

Foundation: We’re anything but fickle…

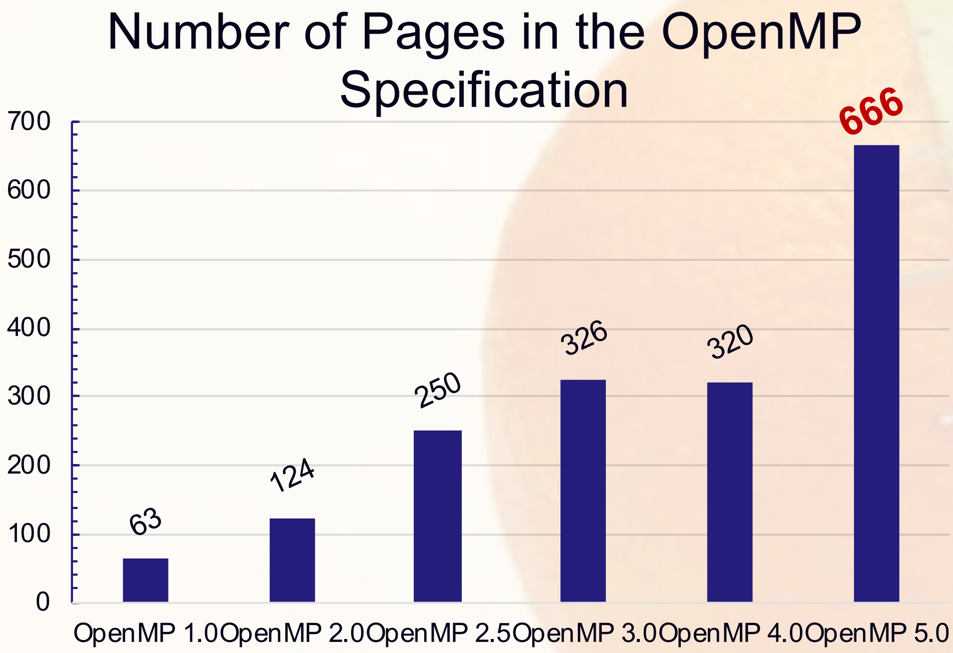
Mindset:

- HPC programmers have long sacrificed convenience, portability and ease of programming for (in hope of) getting good performance.
- *“Worse is better”* always wins over *“Get it right”* … (see Matson talk for a programming-centric view)
  - Quickly gain a large user base, with pressure eventually improve it such that it becomes “good enough”…

> *“HPC programming is about extending the lifespan of languages over many decades.”*
>
> *Nicole Hemsoth*
> *The Next Platform*
> *March 26, 2018*

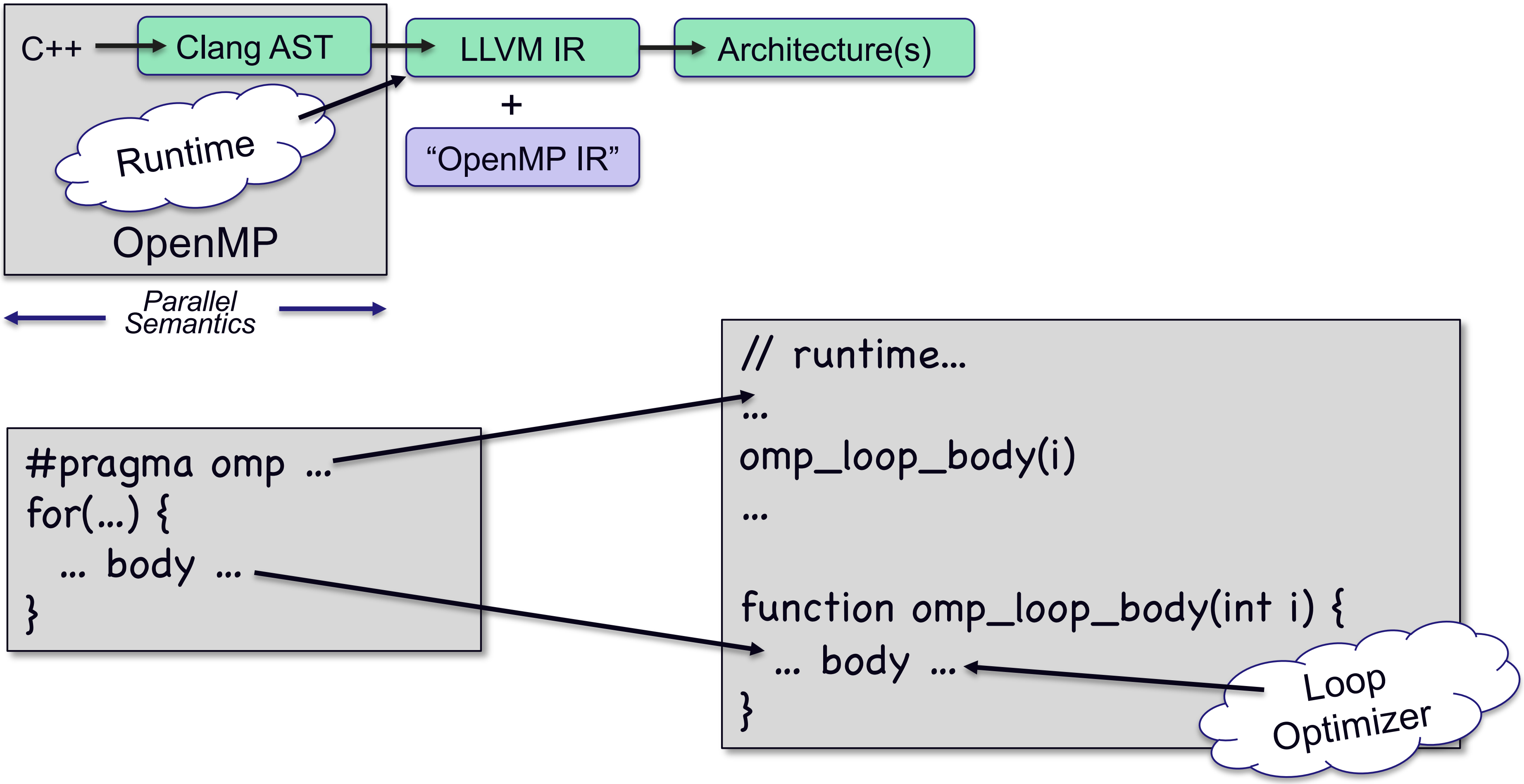
Tim Matson: **HIPS 2016 Keynote**,
2016 IEEE International Parallel and Distributed Processing Symposium Workshops

Increasing Complexity is Readily Apparent


Number of Pages in the OpenMP Specification

- Significant and continued growth of OpenMP feature set…
  - What once were ”simple” directives now have “language-like” features (e.g. precedence)
  - *“Worse is better”*
    - A cautionary note when complexity starts to get the best of the situation – level of risk increases…

Clang+LLVM Compiler Design…

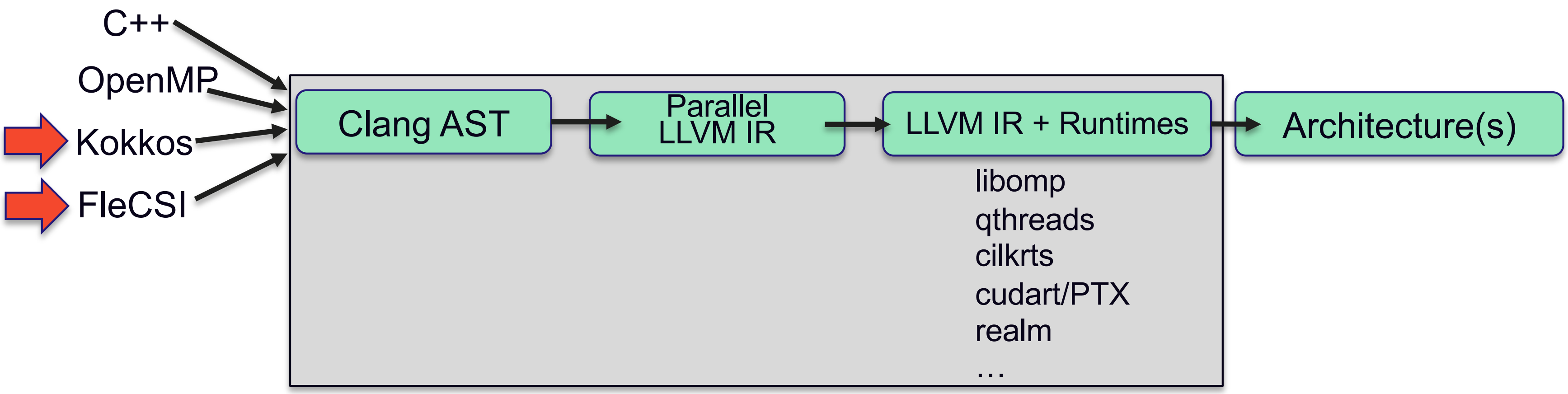


```
#pragma omp …
for(…) {
  … body …
}
```

```
// runtime…
…
omp_loop_body(i)
…

function omp_loop_body(int i) {
  … body …
}
```

Parallel-Aware Design…

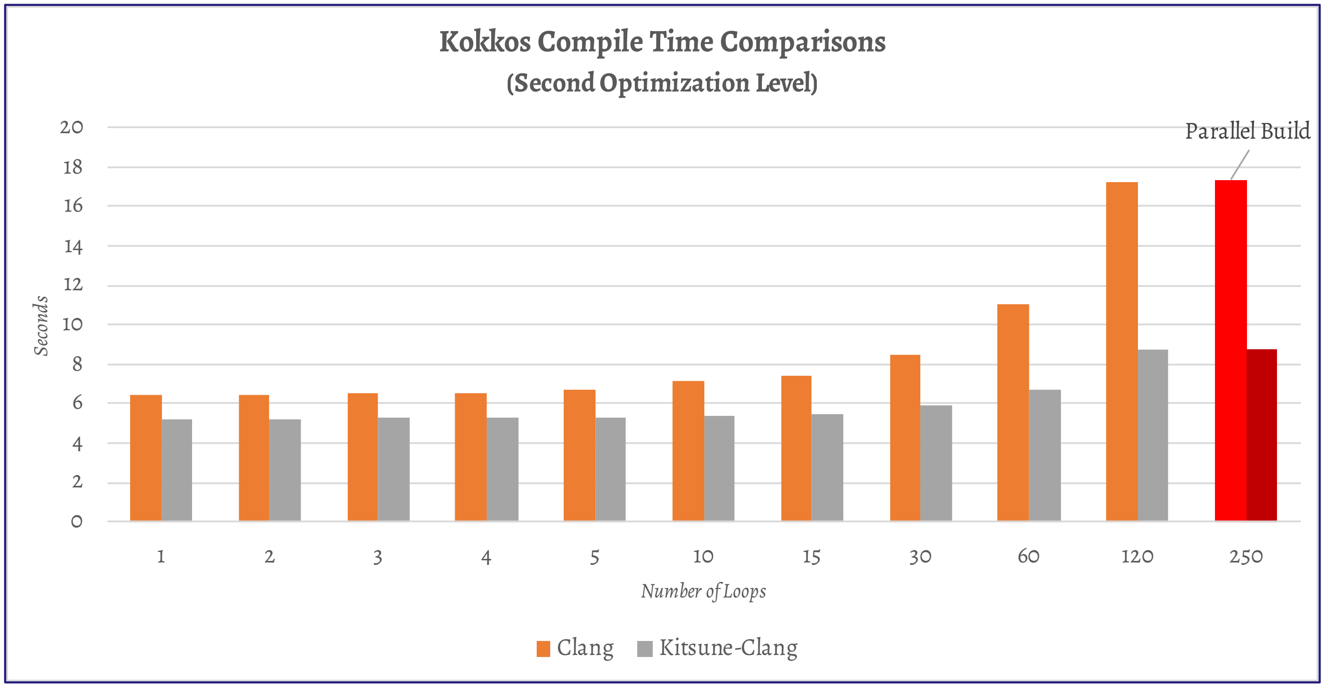


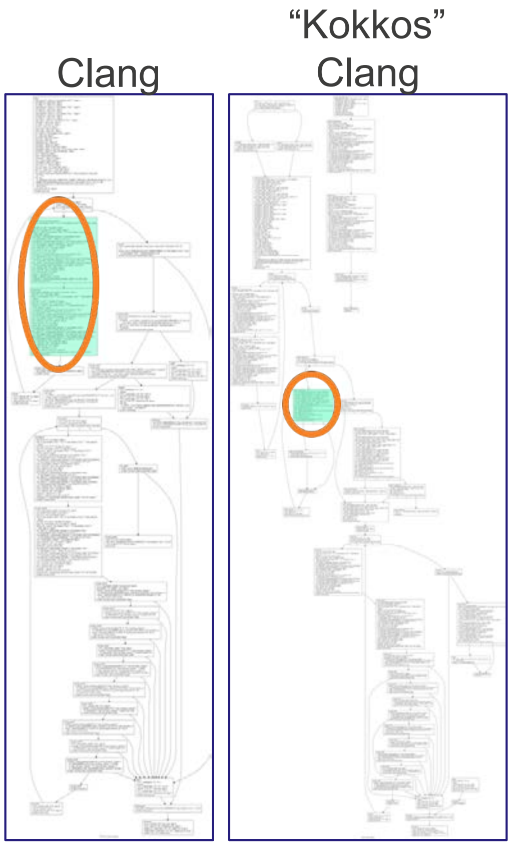
- As we work our way through we're finding modifications to LLVM that can actually enable better code optimization
- Exposes some portability features/potential (frontend to unsupported runtime – e.g., OpenMP to qthreads)
- Open LLVM community discussions, working group. Working implementations key for longer term adoption discussions…

Tapir: Embedding Fork-Join Parallelism into LLVM's Intermediate Representation
by Tao B. Schardl, William S. Moses, and Charles E. Leiserson
Symposium on Principles and Practice of Parallel Programming,
Pages: 249–265, February, 2017

Improved C++ Compile Times

- Motivated by compile time overheads – lower the parallel-IR directly from semantics -- bypassing template expansion…





Comparison of IR-level representations

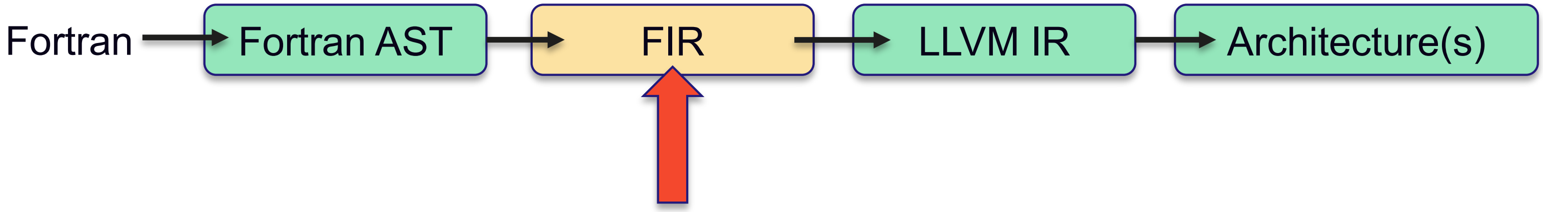
Flang: Fortran Frontend for LLVM

- This is actually a ***brand new*** ECP project
  - Previous investments via NNSA devoted to funding NVIDIA to open source a Fortran front-end for LLVM
  - Officially adopted into LLVM as “flang”
  - Still transitioning into the LLVM community. Building an active community.
  - To this point we’ve pushed well over 2.5M lines of tests and application codes through the parsing and semantics analysis implementations

- Our project builds on this foundation & focuses on ECP and associated Fortran-centric needs
  - Target architectures, optimizations, multi-dimensional array support in LLVM, extensive testing…

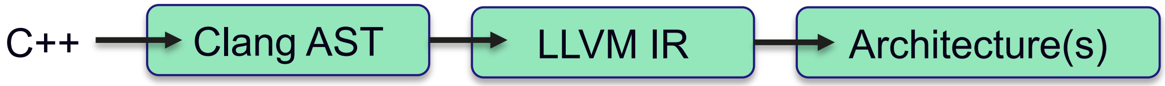
The LLVM Compiler Infrastructure

Flang: A Modern Design for an Important Language



"FIR" talk at LLVM Devs' meeting October 22-23.

"Fortran IR" – dialect of MLIR from Google…
Design concepts shared with Swift, Rust, Julia, and others.
But not Clang… 🤦 (see talk below)



Multi-Level Intermediate Representation Compiler Infrastructure
Tatiana Shpeisman & Chris Lattner (Google)
2019 European LLVM Developers Meeting

ML + Parallel Capabilities

- Many open questions on how to best leverage MLIR+parallel IR components.

Broad team acknowledgements and thanks:

George Stelle, Alexis Perry-Holby, EJ Park, Danny Shevitz, Hal Finkle, Johannes Doerfert, Brian Friesen, David Bernholdt, Doug Miles, Steve Scalpone, Gary Klimowicz, Peter Klausler, Eric Schweitz, Rob Neely, Kim Mish, Mike Heroux

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.