arm

Flang: The Fortran Frontend of LLVM

LLVM devroom, FOSDEM
1 Feb 2020

Kiran Chandramohan
Arm Ltd

Contents

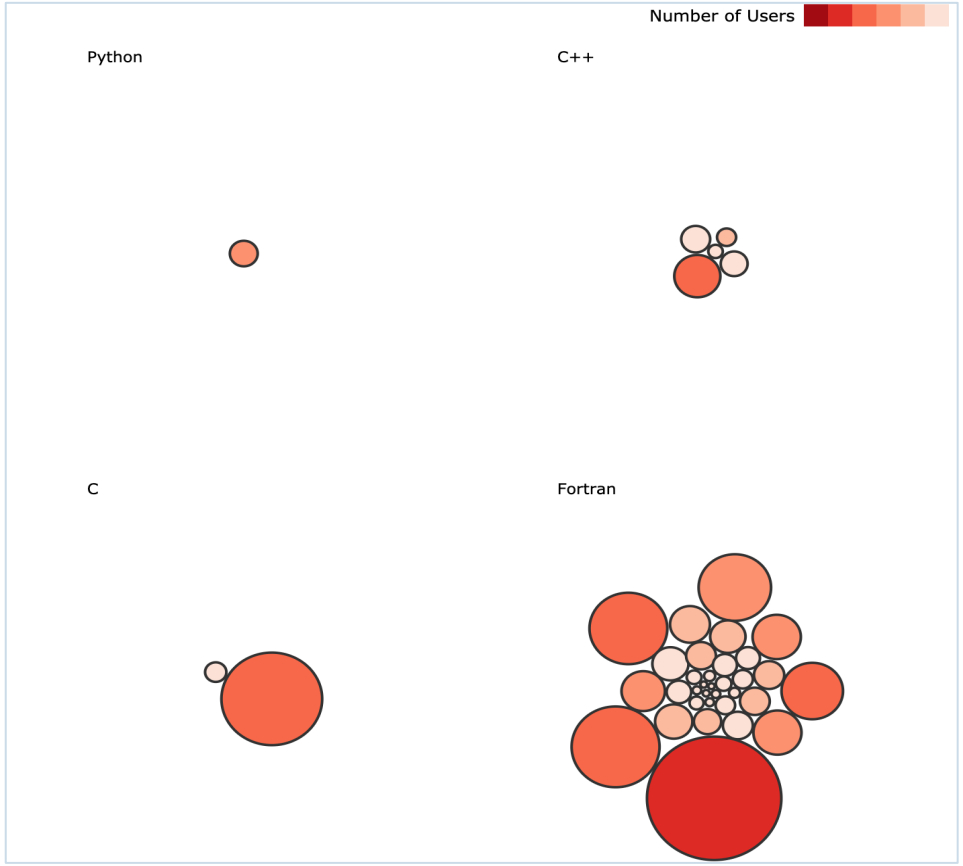
- A word about Fortran
- Old Flang
- New Flang/F18
  - Compiler stages
  - OpenMP
  - Driver
  - Submission to llvm-repo
  - How to contribute
  - Status
- Conclusion

A word about Fortran

- Fortran is a popular language in High Performance Computing
  - "Fortran remains the pre-eminent language in high-performance computing. It is a particularly outstanding language for number crunching, working with sizable floating-point data, or parallel processing. Its strengths in array operations--its wide variety of routines--make it attractive, and there is a huge library of freely available high-performance routines written over 40 years that still work together." – SteveLionel aka Dr. Fortran in CACM September 2017.
  - Is a modern language with support for Object Orientation, Modules, Parallelism etc
- Usage in the real world
  - Weather Forecasting (WRF, UM), Numerical simulation/modelling (VASP, CP2K) etc
  - Libraries : LAPACK, SCIPY
- Standardised
  - 2018 latest published standard
  - 202X and 202Y in the works

Fortran popularity on Archer supercomputer

- One bubble per application
- Size of the bubble represents amount of time used on Archer
- Color represents number of users
- https://www.archer.ac.uk/status/codes/



arm

Old Flang

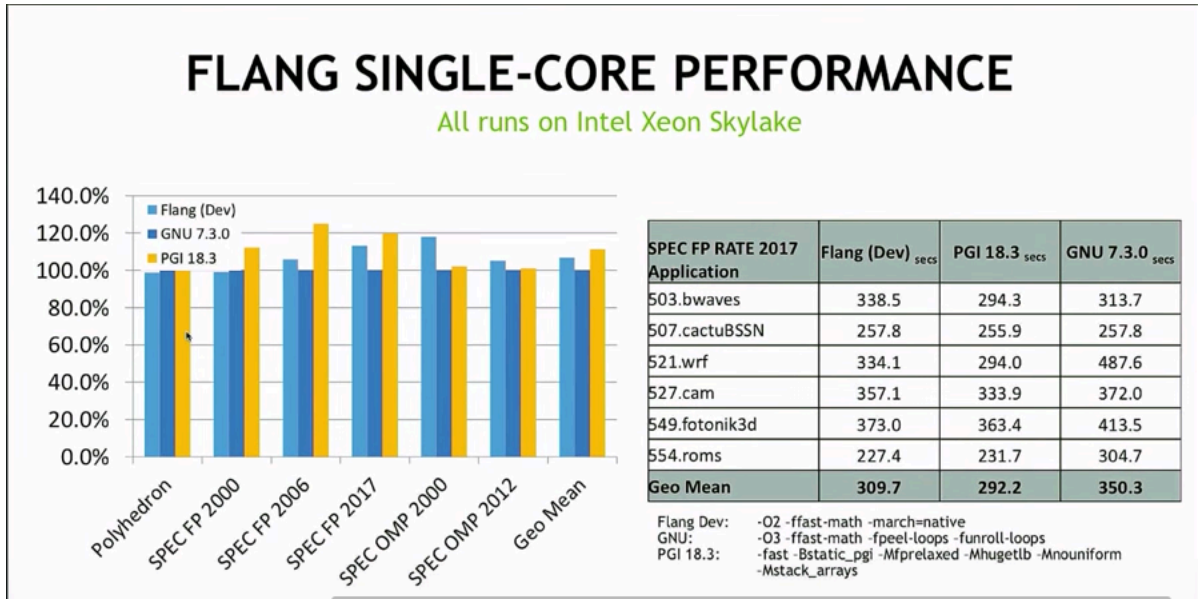
Flang is a Fortran frontend designed to work with the LLVM Compiler Infrastructure

- Sponsored by US DoE and its National Labs
- Open-sourced by Nvidia/PGI with an Apache-2 license
- Switched to LLVM License
- Available since May 2017. https://github.com/flang-compiler/flang
- Supports X86_64, Aarch64 and PowerPC
- Fills a key gap in LLVM for HPC

Common frontend for some commercial compilers

- PGI Compiler
- Arm Compiler for Linux
- AMD AOCC

Performance



SPEC FP RATE 2017 Application	Flang (Dev) secs	PGI 18.3 secs	GNU 7.3.0 secs
503.bwaves	338.5	294.3	313.7
507.cactuBSSN	257.8	255.9	257.8
521.wrf	334.1	294.0	487.6
527.cam	357.1	333.9	372.0
549.fotonik3d	373.0	363.4	413.5
554.roms	227.4	231.7	304.7
Geo Mean	**309.7**	**292.2**	**350.3**

Flang Dev: -O2 -ffast-math -march=native
GNU: -O3 -ffast-math -fpeel-loops -funroll-loops
PGI 18.3: -fast -Bstatic_pgi -Mfprelaxed -Mhugetlb -Mnouniform -Mstack_arrays

20 core Intel Skylake Gold processor @ 2.4GHz with 256 GB memory

Source : Flang Update by Steve Scalpone @ Euro LLVM, 2018

arm

Standards Conformance

Fortran 2003

Full Support

A few intrinsics are not supported in intialisation

Fortran 2008

Partial Support

Submodules, Block construct, contiguous attribute, intrinsics (Bessel, gamma, norm2 etc)

Do concurrent supported with serial execution

Coarrays, intrinsics (merging, masking etc)

No plan for Coarrays

Fortran 2018

No plan

Issues

Prolonged Pull Request processing

Previously due to dependency of Flang on PGI's commercial compiler

Currently blocked due to lack of CI

Code is old, difficult to maintain, entry barrier is high

Difficult to implement new features

Error messages do not give full information (e.g : no column)

Flang cannot be an LLVM project

Written in C

Cannot be used as a library or for building tools

Does not use the IRBuilder

Command line flags are not name based

Time for a new Flang?

New Flang/F18

New Fortran frontend developed as an Open source Project

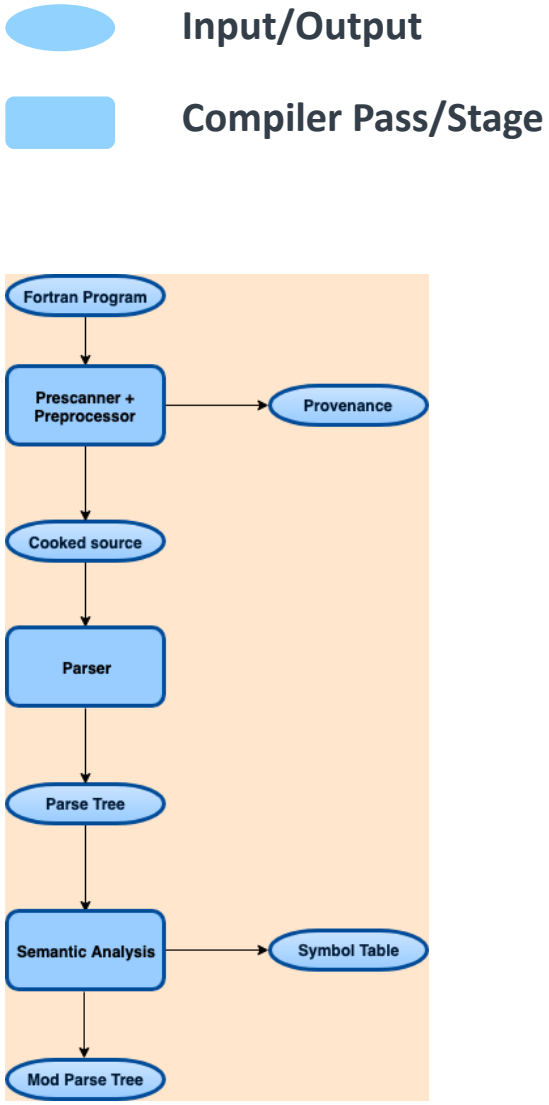
- Accepted as the LLVM Fortran frontend
- LLVM License. Apache with LLVM Exceptions
- No CLA required
- PGI/Nvidia is the lead developer
- Arm, AMD, US National Labs contributing

Features

- Uses 2018 standard as the reference for implementation
- Very standards friendly
- Written in modern C++ (C++17)
- AST as C++ classes
- AST lowered only after semantic checks
- High quality source locations
- Can be used for tooling
- Flangd already in the works

F18 Preprocessing

- Prescanner generates cooked character stream
  - Normalized source
  - Expanded macros, character case
  - Hides complexity from rest of compiler
- Provenance
  - Index into cooked character stream
  - Map from cooked character stream to sources maintained



arm

F18 Parsing

- Recursive Descent Parsing
- Grammar taken from standard and suitably modified
  - Left recursion removed
- Uses Parser combinators
  - Token parser
  - Operators & functions to combine parsers
- Parse tree closely follows specification in the standard

!Fortran source

```
integer::x=1
```

//2018 standards document
//R803 entity-decl ->
//object-name [(array-spec)] [lbracket coarray-spec rbracket]
// [* char-length] [initialization]

//lib/parser/Fortran-parsers.cpp

```
PARSER(construct<EntityDecl>(objectName,
maybe(arraySpec), maybe(coarraySpec),
maybe("*" >> charLength),
maybe(initialization)))
```

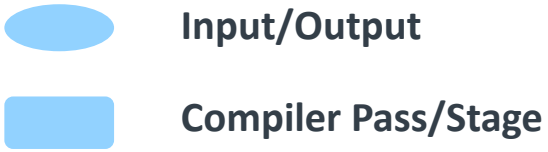
//Parse Tree Node (include/flang/parser/parse-tree.h)

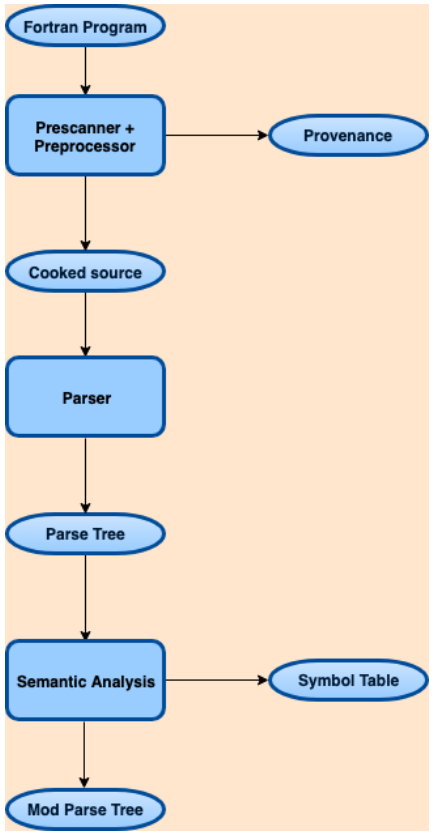
```
std::tuple<ObjectName,
std::optional<ArraySpec>,
std::optional<CoarraySpec>,
std::optional<CharLength>,
std::optional<Initialization>> t;
```


arm

F18 Semantic Analysis

- Checks the rules/constraints mentioned in the standard
- Label resolution
- Name resolution (Symbol Table)
- Modifies parse tree if ambiguous
- Constant Expression evaluation
- Expression and Statement Semantic Checks
- Emits Module files





arm

Module Format

- Modules will be stored as Fortran source
  - Module files will contain a header
    - Magic string, Version, Checksum
  - The body will contain declarations of all user visible entities
- Reading module files is fast
  - Fast parser, No pre-processing necessary

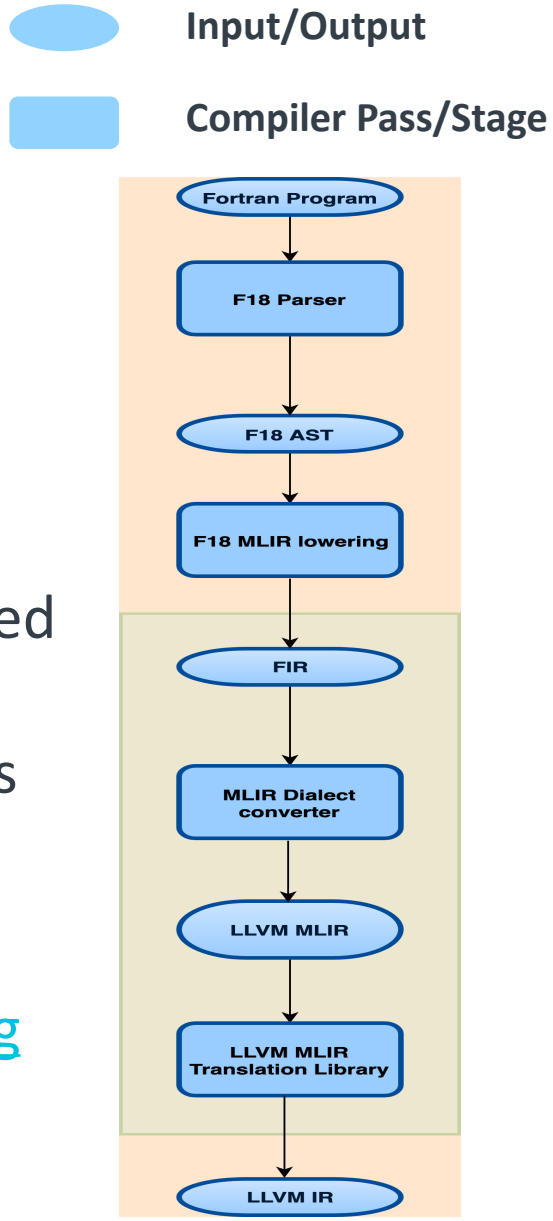
```
!mymod.f90
module vars
integer :: a
real :: b
contains
 subroutine add_val_a(x)
  integer :: x
  a = a + x
 end subroutine
end module
```

```
!vars.mod
!mod$ v1 sum:672b5185d5193446
module vars
integer(4)::a
real(4)::b
contains
subroutine add_val_a(x)
integer(4)::x
end
end
```


arm

Optimizer

- Uses MLIR for developing a high level IR
- MLIR is a framework for developing IRs
- FIR (Fortran IR) is the name of the dialect
- After several optimizations, the FIR dialect is converted to the LLVM dialect
  - Do optimizations which require Fortran semantics
- The LLVM dialect is then translated to LLVM IR
- Refer to llvm-dev talk for more details
  - https://www.youtube.com/watch?v=ff3ngdvUang



arm

OpenMP

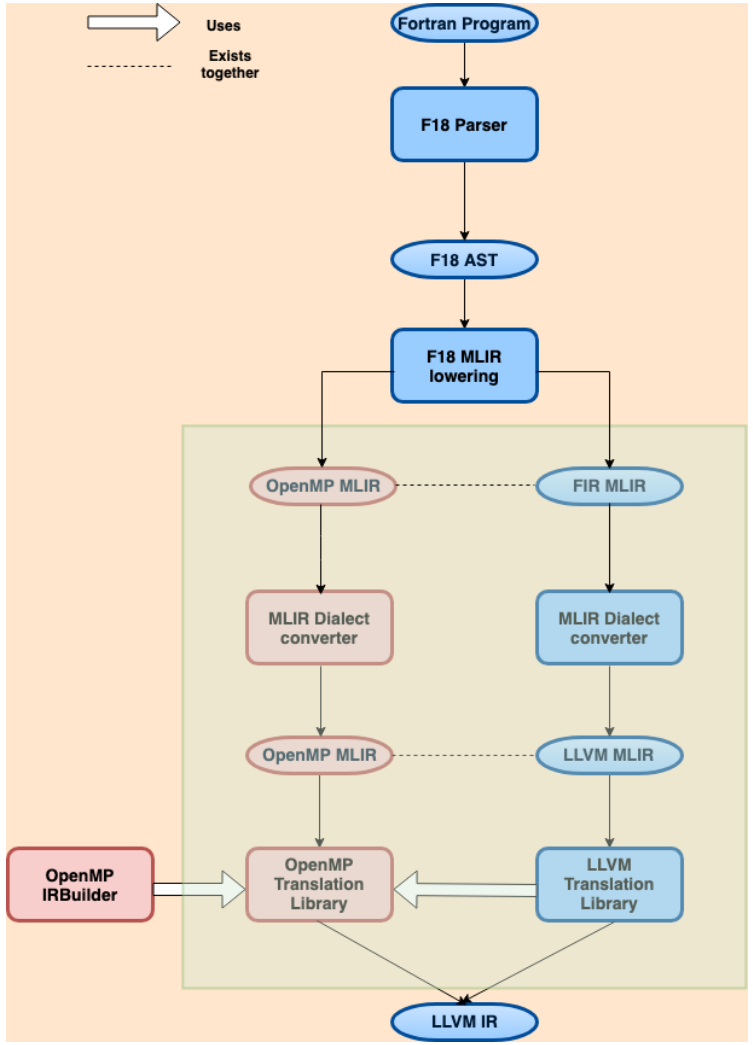
MLIR

- Flang compiler uses the MLIR based FIR dialect as its IR
- FIR models the Fortran language portion but does not have a representation for OpenMP constructs
- Add a dialect in MLIR for OpenMP
- MLIR provides common framework for representing OpenMP and Fortran constructs
- Take advantage of optimisations and avoid black boxes.

OpenMP IRBuilder

- Reusing codegen from Clang
- Refactor codegen for OpenMP constructs in Clang and move to the LLVM directory

High Level Design for OpenMP

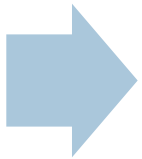


arm

Example : OpenMP Parallel

Fortran source with OpenMP

```
!Fortran code
!$omp parallel
  c = a + b
!$omp end parallel
!More Fortran code>
```



Flang parse tree

```
<Fortran parse tree>
| | ExecutionPartConstruct ->
ExecutableConstruct ->
OpenMPConstruct ->
OpenMPBlockConstruct
| | | OmpBlockDirective -> Directive =
Parallel
| | | OmpClauseList ->
| | | Block
| | | | ExecutionPartConstruct ->
ExecutableConstruct -> ActionStmt ->
AssignmentStmt
| | | | | Variable -> Designator ->
DataRef -> Name = 'c'
| | | | | Expr -> Add
| | | | | | Expr -> Designator -> DataRef
-> Name = 'a'
| | | | | | Expr -> Designator -> DataRef
-> Name = 'b'
| | | OmpEndBlockDirective ->
OmpBlockDirective -> Directive =
Parallel <More Fortran parse tree>
```

MLIR: FIR + OpenMP

```
Mlir.region(…) {
…
omp.parallel {
  %1 = addf %2, %3 : f32
}
%21 = <more fir> … }
```

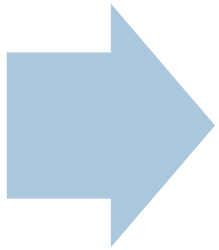

arm

Example : OpenMP Parallel

MLIR: LLVM + OpenMP dialect

```
Mlir.region(…)
{
…
omp.parallel {
  %1 = llvm.fadd %2, %3 : !llvm.float
}
%21 = <more llvm dialect>
…
}
```

Use OpenMP IRBuilder



LLVM IR

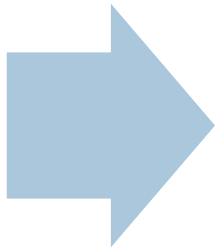
```
define @outlined_parallel_fn(...)
{
  …
  %1 = fadd float %2, %3
  ...
}
define @xyz(…)
{
  %1 = alloca float
   ....
   call
kmpc_fork_call(...,outlined_parallel_fn,...)
}
```


arm

Example : OpenMP Collapse

Fortran source with OpenMP

```
!$omp parallel  do private(j) collapse(2)
do i=lb1,ub1
  do j=lb2,ub2
    ...
    ...
  end do
end do
```



MLIR: FIR + OpenMP dialects

```
Mlir.region(…) {
omp.parallel {
  omp.do {collapse = 2} {
    fir.do %i = %lb1 to %ub1 : !fir.integer {
      fir.do %j = %lb2 to %ub2 : !fir.integer {
        ...
      }
    }
  }
 }
}
```


arm

Example : OpenMP Collapse

MLIR: FIR + OpenMP + loop

```
Mlir.region(…) {
omp.parallel {
  omp.do {collapse = 2} {
loop.for %i = %lb1 to %ub1 :
!integer {
     loop.for %j = %lb2 to
%ub2 : !integer {
       …
     }
    }
   }
  }
}
```

MLIR: FIR + OpenMP + loop

```
Mlir.region(…) {
omp.parallel {
  omp.do {
%ub3 = …
loop.for %i = 0 to %ub3 :
!integer {
 …
     }
   }
  }
 }
}
```

Loop Collapsed

MLIR: LLVM + OpenMP

```
Mlir.region(…) {
omp.parallel {
  %ub3 = …
  omp.do %i = 0 to %ub3 :
!integer {
  …
   }
  }
 }
}
```


arm

Driver

- Introduces a bin/flang binary
- Reuses libclangDriver and Options.td
- Sample invocation

  bin/flang -o foobar foobar.f90

  bin/flang -fc1 foobar.f90 -o /tmp/foobar_cafe1234.o
- HPC applications are mixed Fortran, C, C++
  - Important that frontend drivers are aware of each other
  - Can also be invoked as
    bin/clang –driver-mode=FORTRAN –o foobar foobar.f90
  - Without the driver mode will invoke gfortran (for now)
- See RFC for more details.
  http://lists.llvm.org/pipermail/cfe-dev/2019-June/062669.html

Submission to llvm-project

Initial submission discussion provided some feedback

Parser and Semantic analysis do not use LLVM API

IR (MLIR) uses

Currently addressing the issues pointed out by the community

Matching LLVM coding guidelines

Moving public headers to include folder

Renaming *.cc as *.cpp

Removing additional settings in clang-format

Using LLVM Infrastructure

Filesystem Handling

Using LLVM streams

Lit for testing

Using LLVM data-structures wherever applicable

Status

- Parser work is complete
  - Parses Fortran 2018, OpenMP 4.5
- Semantic Checks are almost complete
- Work in progress on MLIR based optimizer
- Work beginning on
  - Runtime
    - Rewriting some portions in C++ (I/O in progress)
    - Math library will continue to be pgmath
  - OpenMP
- Tentative Timeline

Moving to llvm-project repo	1 or 2 months
Serial codegen	Middle of this year
Parallel codegen (OpenMP 4.5)	Early next year
OpenMP 5.0 + Coarrays	End of 2021

Contributions

Project welcomes contributions

Code, Bug reports

Start with the documentation

https://github.com/flang-compiler/f18/tree/master/documentation

Start with C++style.md, FortranForCProgrammers.md, Overview.md

Projects page contains work items finished, in progress and not started.

Can pick up tasks from here or from issues tracker

Send a mail to flang-dev before starting to work

Code reviews in github

Read PullRequestChecklist.md before submitting

https://github.com/flang-compiler/f18/projects

NOTE: These links and process will change after submission to LLVM project

Code reviews will be in phabricator

arm

Conclusion

Old Flang demonstrated that an industry strength, performant LLVM based Fortran compiler is possible

New Flang/F18 addresses the deficiencies

New Flang will be the Fortran frontend of LLVM

Submission to LLVM expected to happen soon

Fills a gap for the LLVM HPC story

Written in modern C++

Uses MLIR

Shares code for OpenMP, Driver etc.

Aspires to be the compiler of choice for prototyping Fortran features for standardization

Adheres to 2018 standard

New Flang is under development

You can contribute

arm

arm

Thank You
Danke
Merci
谢谢
ありがとう
Gracias
Kiitos
감사합니다
धन्यवाद
شكرًا
תודה