# New Flang: The Modern Fortran Frontend of LLVM

BCS Fortran Meeting

24 Sep 2019

Kiran Chandramohan
Arm Ltd

# Contents

- Flang
  - Performance
  - Standards Conformance
  - Why new Flang
- New Flang/F18
  - Intro
  - Internals
  - Status
- Conclusion

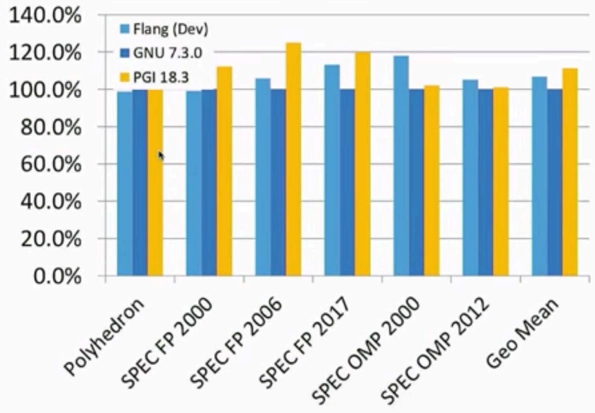
# Introduction

- 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
  - 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

## FLANG SINGLE-CORE PERFORMANCE

All runs on Intel Xeon Skylake




| 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

# Standards Conformance

- Fortran 2003
  - Full Support
  - A few intrinsics are not supported in intialisation
- Fortran 2008
  - Partial Support
  - Submodules, contiguous attribute, intrinsics (Bessel, gamma, norm2 etc)
  - Do concurrent supported with serial execution
  - Coarrays, Block construct, intrinsics (merging, masking etc)
  - Work underway for Block construct
  - No plan for Coarrays
    - No customer has specifically asked for this
    - Open Coarrays a bit tied to Gfortran
- Fortran 2018
  - No plan

# Issues

- Code contribution requires a CLA
- Prolonged Pull Request processing due to dependency of flang on PGI's commercial compiler
- 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

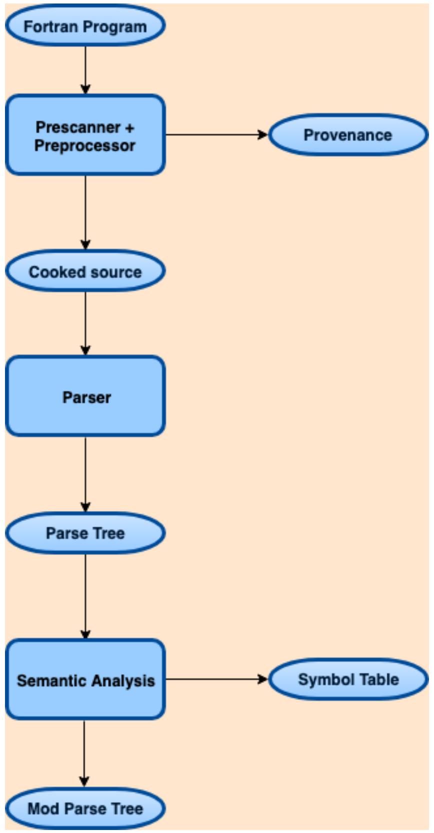
- New Fortran frontend developed as an Open source Project
  - Apache-2 License. Will change to match LLVM
  - No CLA required
  - PGI is lead developer
  - Arm is contributing
- Features
  - Uses 2018 standard as the reference for implementation
  - Very standards friendly
  - Written in modern C++ (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

Input/Output

Compiler Pass/Stage

- 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



# 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/grammar.h
```
PARSER(construct<EntityDecl>(objectName,
maybe(arraySpec), maybe(coarraySpec),
maybe("*" >> charLength),
maybe(initialization)))
```

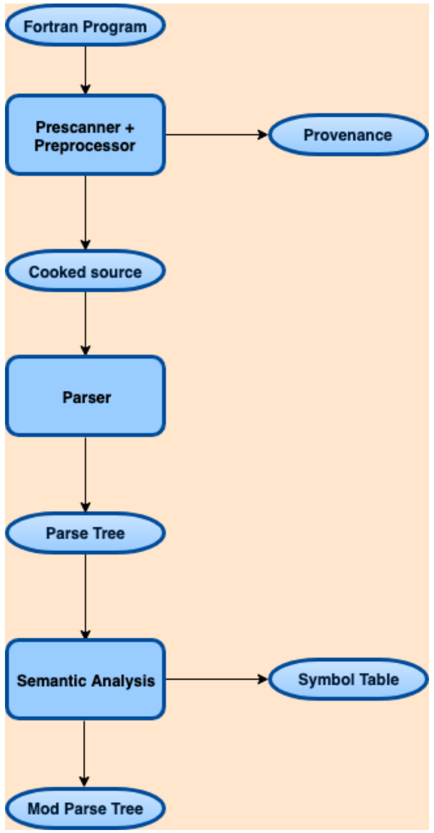
//Parse Tree Node (lib/parser/parse-tree.h)
```
std::tuple<ObjectName,
std::optional<ArraySpec>,
std::optional<CoarraySpec>,
std::optional<CharLength>,
std::optional<Initialization>> t;
```

# F18 Semantic Analysis

Input/Output

Compiler Pass/Stage

- Checks the rules/constraints mentioned in the standard
- Modifies parse tree if ambiguous
- Creates Symbol table
- Constant Expression evaluation
- Emits Module files



# Descriptor

**/* 18.5.3 generic data descriptor */**

```
typedef struct CFI_cdesc_t {
  /* These three members must appear first,
  in exactly this order. */
  void *base_addr;
  size_t elem_len; /* element size in bytes */
  int version; /* == CFI_VERSION */
  CFI_rank_t rank; /* [0 .. CFI_MAX_RANK] */
  CFI_type_t type;
  CFI_attribute_t attribute;
  unsigned char f18Addendum;
  CFI_dim_t dim[]; /* must appear last */
} CFI_cdesc_t;
```

**//Addendum**

```
const DerivedType
*derivedType_{nullptr};
std::uint64_t flags_{0};
TypeParameterValue len_[1];
```

# Module Format

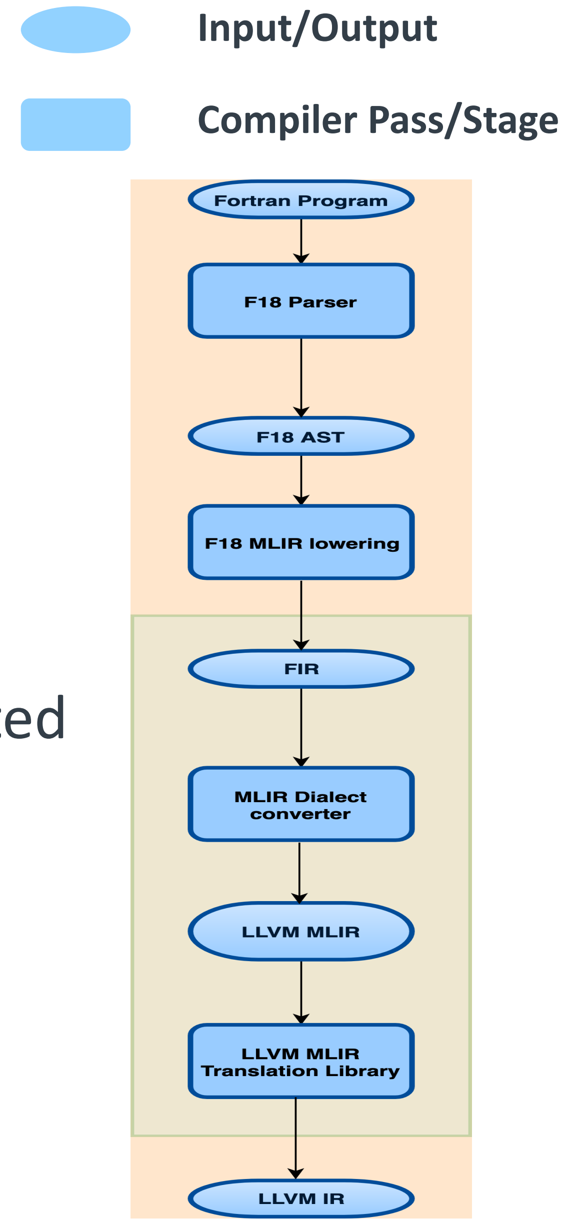
- Modules will be stored as Fortran source
  - Module files will contain a header
    - Magic string, Version, Checksum
  - The body will contain declarations of all user visible entities
- Reading module files is fast
  - Fast parser, No pre-processing necessary

```
!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
```

# 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.
- The LLVM dialect is then translated to LLVM IR

# Status

- Parser work is complete
  - Parses Fortran 2018
  - OpenMP 4.5
- Semantic Checks are nearing completion
  - Switched ON by default if run as flang
- Work in progress on MLIR based optimizer
- Work beginning on runtime
  - Rewriting some portions in C++
  - Will retain I/O library functions of Old Flang
  - Math library will continue to be pgmath
- Tentative Timeline
  - Serial codegen by end of this year
  - Parallel codegen (OpenMP 4.5) by end of next year
  - OpenMP 5.0 + Coarrays  by end of 2021

# Conclusion

- Old Flang demonstrated that an industry strength, performant LLVM based Fortran compiler is possible
- New Flang/F18 addresses the deficiencies and will be the Fortran frontend of LLVM
- New Flang makes creation of compiler tools possible
- Aspires to be the compiler of choice for prototyping features for standardization
  - Adheres to 2018 standard
- New Flang is under development
  - You can contribute
  - https://github.com/flang-compiler/f18/projects
  - https://github.com/flang-compiler/f18/tree/master/documentation

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