AMD

DEBUG-INFO PR'S

ALOK KUMAR SHARMA
SOURABH SINGH TOMAR

29TH JULY 2020

DEBUG INFO PR UNDER REVIEW

Owner: Alok

PR’S:

- #863: Support for -gdwarf-5 in flang2
- #89: Support for -gdwarf-5 option in Flang driver
- #90: Flang does not choose correct dwarf version when multiple -g/-gdwarfN mentioned
- #91: Default dwarf version should be 4 for -g with flang
- #877: flang generated executable shows wrong value for character type in debugger
- #883: Flang generated executable does not show result variable of function
- #888: flang gen-exec does not show routine variables with parameter attribute
- #901: flang gen-exec doesnt show assumed shape array in debugger
- #902: flang gen-exec doesnt print allocatable array in debugger
- #913: flang gen-exec does not generate DW_AT_associated or DW_AT_allocated

#863: Support for -gdwarf-5 in flang2

Status: Approved

PR Description: To support –gdwarf-4 and –gdwarf-5, xflags (0x1000000, 0x2000000 respectively) are reserved in Flang compiler (0x200 and 0x4000 are already reserved for 2, 3). When –x 120 0x1000000 is passed to flang2 it sets DWARF version to 4, and “-x 120 0x2000000” sets DWARF version to 5 in LLVM IR file. Flang driver can pass appropriate –x 120 flag for –gdwarf-4 and –gdwarf-5 (separate PR #89).

 # flang2 –x 120 0x1000000

!0 = !{i32 2, !"Dwarf Version", i32 4}

 # flang2 –x 120 0x2000000

!0 = !{i32 2, !"Dwarf Version", i32 5}

#89: Support for -gdwarf-5 option in Flang driver.

Status: Approved

PR Description: To support –gdwarf-4 and –gdwarf-5, The Flang driver now passes “-x 120 0x1000000” option for –gdwarf-4 and “-x 120 0x2000000” option for –gdwarf-5 (PR #863 in Flang compiler will honor it).

 # flang –gdwarf-4 …

 # flang2 –x 120 0x1000000 …

 # flang –gdwarf-5 …

 # flang2 –x 120 0x2000000 …

#90:Flang does not choose correct dwarf version when multiple -g/-gdwarfN mentioned

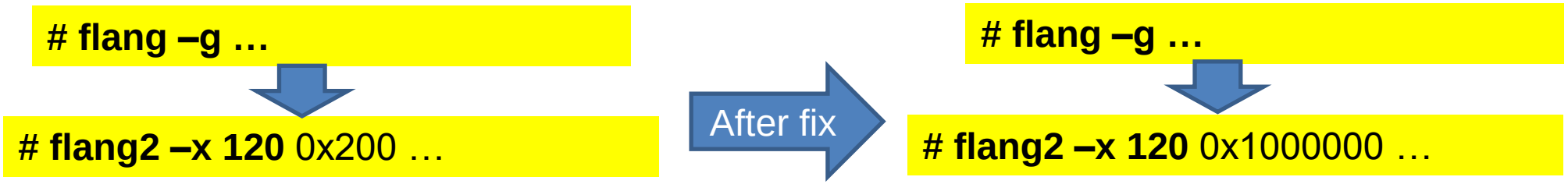
Status: Approved

PR Description: The Flang driver does not choose the correct dwarf version when multiple –gdwarf-N options are given. Clang/gfortran choose the last option, Currently Flang chooses the least option, It is fixed to behave same as Clang/gfortran.

#91: Default dwarf version should be 4 for -g with flang

Status: Approved

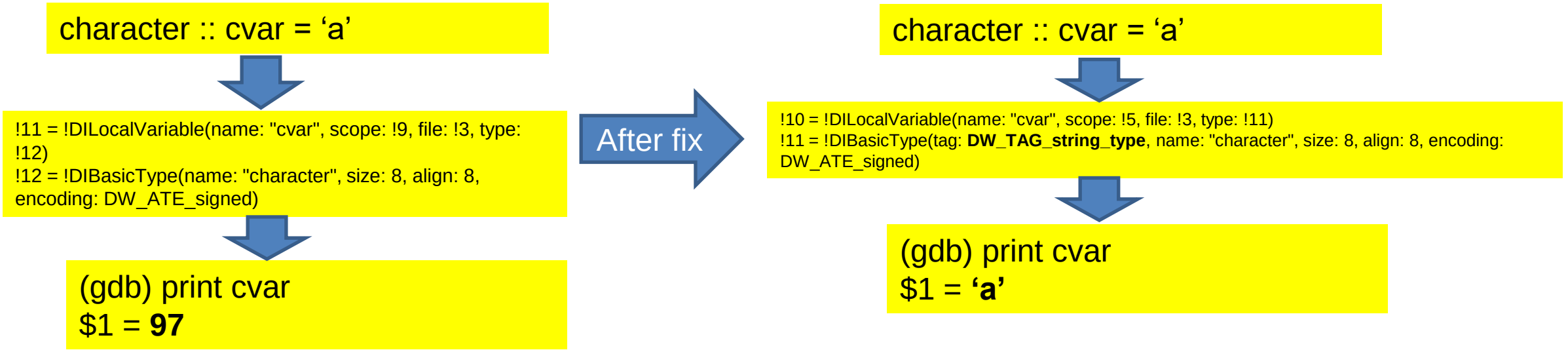
PR Description: Currently Flang dumps dwarf version 2 for -g and 4 for absence of -g.



#877: flang generated executable shows wrong value for character type in debugger

Status: Approved

PR Description: For fortran character type DW_TAG_string_type should be generated, in absence of this, debugger doesnt differentiate between character(kind=1) and integer(kind=1). Due to this debugger displays value as an integer.



#883: Flang generated executable does not show result variable of function.

Status: Approved

PR Description: The Flang compiler marks result variable as artificial variable due to this debuggers don’t show it.

function func(arg) result(rvar)

call void @llvm.dbg.declare(metadata i32* %rvar_299, metadata !14, metadata !DIExpression()), !dbg !11
!14 = !DILocalVariable(scope: !12, file: !3, type: !9, flags: DIFlagArtificial)

(gdb) print rvar
No symbol “rvar" in current context.

After fix

function func(arg) result(rvar)

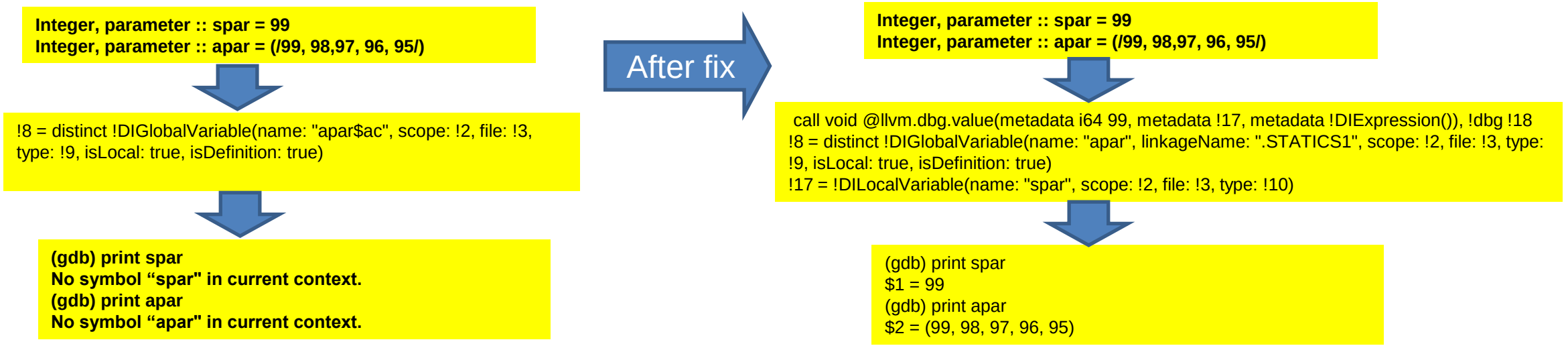
call void @llvm.dbg.declare(metadata i32* %rvar_339, metadata !13, metadata !DIExpression()), !dbg !11
!13 = !DILocalVariable(name: "rvar", scope: !5, file: !3, type: !9)

(gdb) print rvar
$1 = **5**

#888: flang gen-exec does not show routine variables with parameter attribute

Status: Approved

PR Description: The Flang compiler optimizes away the variables with parameter attribute in front-end itself making it unavailable for debuggers. Enums are also handled as parameters so those are also unavailable. It is fixed by inserting dbg.value instructions.



#901: flang gen-exec does not show assumed shape array in debugger

Status: Under review, LLVM11 dependency

PR Description: The Flang compiler generates 0 sized DISubrange for assumed shape arrays. After upgrade of DISubrange in LLVM11, now proper IR is generated.

```
Subroutine sub(assume, n)
  Integer :: assume(n)
End subroutine sub
```

```
!8 = !DICompositeType(tag: DW_TAG_array_type, baseType: !9,
size: 32, align: 32, elements: !10)
!10 = !{!12}
!12 = !DISubrange(count: 0, lowerBound: 1)
!13 = !DILocalVariable(name: "assume", arg: 1, scope: !5, file: !3,
type: !8)
```

After fix

```
Subroutine sub(assume, n)
  Integer :: assume(n)
End subroutine sub
```

```
!8 = !DICompositeType(tag: DW_TAG_array_type, baseType: !9, size: 32, align: 32, elements:
!10)
!10 = !{!12}
!12 = !DISubrange(lowerBound: 1, upperBound: !13)
!13 = distinct !DILocalVariable(scope: !5, file: !3, type: !14, flags: DIFlagArtificial)
```

#902: flang gen-exec doesnt print allocatable array in debugger

Status: Under review, LLVM11 dependency

PR Description: This patch enables debugging of allocatable arrays using DW_OP_push_object_address, DISubrange and DW_AT_data_location/dataLocation.

integer(kind=4), allocatable :: arr(:)

!10 = !DILocalVariable(name: "arr", scope: !9, file: !3, type: !11)
!11 = !DICompositeType(tag: DW_TAG_array_type, baseType: !12, size: 32, align: 32, elements: !13)
!13 = !{!14}
!14 = !DISubrange(count: 0, lowerBound: 1)

After fix

integer(kind=4), allocatable :: arr(:)

!15 = distinct !DILocalVariable(scope: !5, file: !3, type: !10, flags: DIFlagArtificial)
!22 = !DILocalVariable(name: "arr", scope: !5, file: !3, type: !23)
!23 = !DICompositeType(tag: DW_TAG_array_type, baseType: !11, size: 32, align: 32, elements: !24, **dataLocation: !15**)
!24 = !{!25}
!25 = **!DISubrange(lowerBound: !DIExpression(DW_OP_push_object_address, DW_OP_plus_uconst, 80, DW_OP_deref), upperBound: !DIExpression(DW_OP_push_object_address, DW_OP_plus_uconst, 120, DW_OP_deref), stride: !DIExpression(DW_OP_push_object_address, DW_OP_plus_uconst, 112, DW_OP_deref, DW_OP_constu, 4, DW_OP_mul))**

#913: flang gen-exec does not generate DW_AT_associated or DW_AT_allocated

Status: Under review, LLVM11 dependency

PR Description: With this patch, debugging of allocatable arrays when not allocated or deallocated becomes better. Same is true for pointer arrays.

integer(kind=4), allocatable :: arr(:)

```
!10 = !DILocalVariable(name: "arr", scope: !9, file: !3, type: !11)
!11 = !DICompositeType(tag: DW_TAG_array_type, baseType: !12,
size: 32, align: 32, elements: !13)
!13 = !{!14}
!14 = !DISubrange(count: 0, lowerBound: 1)
```

```
(gdb) pt arr
type = integer (140737345375288:140737354129776)
(gdb) p arr
value requires 35017956 bytes, which is more than max-value-size
```

After fix

integer(kind=4), allocatable :: arr(:)

```
!15 = distinct !DILocalVariable(scope: !5, file: !3, type: !10, flags: DIFlagArtificial)
!22 = !DILocalVariable(name: "arr", scope: !5, file: !3, type: !23)
!23 = !DICompositeType(tag: DW_TAG_array_type, baseType: !11, size: 32, align: 32, elements:
!24, dataLocation: !15, allocated: !16)
!24 = !{!25}
!25 = !DISubrange(lowerBound: !DIExpression(DW_OP_push_object_address,
DW_OP_plus_uconst, 80, DW_OP_deref), upperBound:
!DIExpression(DW_OP_push_object_address, DW_OP_plus_uconst, 120, DW_OP_deref),
stride: !DIExpression(DW_OP_push_object_address, DW_OP_plus_uconst, 112,
DW_OP_deref, DW_OP_constu, 4, DW_OP_mul))
```

```
(gdb) pt arr
type = integer, allocatable (:)
(gdb) p arr
$1 = <not allocated>
```

DEBUG INFO PR UNDER REVIEW

Owner: Sourabh

PR’S:

- #895: [DebugInfo]Fix for redundant lexical block in debug info
- #908: [DebugInfo] Remove "_V_" from parameter names in case of byvalue parameters
- #907: [DebugInfo] flang produces dbg.declare and dbg.value for the same variable
- #898: [DebugInfo] Corrected debuginfo for modules
- #878: [DebugInfo]: Fix for missing DISPFlagOptimized in debug metadata
- #865: [DebugInfo]: Corrected emission of isOptimized flag.

PR895: Fix for redundant lexical block…

Status: **Approved**

PR Description: For a typical FORTRAN program flang creates a redundant Lexical block just below the subprogram. This causes debugger to assume a Lexical scope present in source program.

```
SUBROUTINE sub(foo_arg)
      integer,value :: foo_arg
      print*, foo_arg
END SUBROUTINE
```

→

```
!5 = distinct
!DISubprogram(name:
"sub"
!DILexicalBlock(scope: !5
```

→

```
DW_TAG_subprogram
  DW_TAG_lexical_block
    DW_TAG_formal_parameter
```

↓

```
(gdb) b sub
(gdb) info args
No arguments.
```

PR908:**Remove "_V_" from parameter names…**

Status: **Under review**

PR Description: For parameters passed byvalue to a subroutine flang prefixes **“_V_”** to argument name this caused debugger to display this prefixed name and not the actual name.

```
SUBROUTINE sub(foo_arg)
    integer,value :: foo_arg
    print*, foo_arg
END SUBROUTINE
```

→ **!DILocalVariable(name: "_V_foo_arg"**

→ **DW_TAG_formal_parameter**
DW_AT_name ("_V_foo_arg")

↓

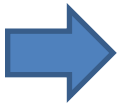
```
(gdb) b sub
(gdb) info args
_V_foo_arg = 55
```

PR907: flang produces dbg.declare and dbg.value …

Status: **Under review**

PR Description: flang produces dbg.declare and dbg.value at (-O0) for the same variable this is incorrect WRT to dbg.value intrinsic semantics. This may also cause spurious location list as program is compiled at higher optimization levels.

```
SUBROUTINE sub(foo_arg)
      integer,value :: foo_arg
      print*, foo_arg
END SUBROUTINE
```



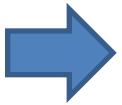
```
call void @llvm.dbg.declare(metadata i32* %_V_foo_arg.addr,
metadata !9, metadata !DIExpression()), !dbg !11
call void @llvm.dbg.value(metadata i32 %0, metadata !9, metadata
!DIExpression()), !dbg !11
!9 = !DILocalVariable(name: "_V_foo_arg", scope: !10, file: !3, type:
!8)
```

PR898: **Corrected debug info for modules**

Status: **Under review,** Dependency: LLVM11

PR Description: flang produced debug info wasn't enough for GDB to display the contents of FORTRAN Module to user. **Line no.** and **File** are needed to cater this. But **DIModule** metadata itself was missing these fields. So **DIModule** is extended to contain these fields and subsequently flang is also modified to emit these fields.

```
module dummy
      integer :: foo
end module dummy
```



```
!2 = !DIModule(scope: !3, name: "dummy")
```

```
DW_TAG_module
      DW_AT_name      ("dummy")
```

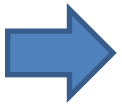
```
(gdb) info modules
All defined modules:
(gdb)
```

PR878: missing DISPFlagOptimized in debug…

Status: Approved

PR Description: flang is not generating “DISPFlagOptimized" flag for optimized builds "-O2 -g or -O3 -g“. This flag is need for lower level processing of debug metadata.

```
SUBROUTINE sub(foo_arg)
    integer,value :: foo_arg
    print*, foo_arg
END SUBROUTINE
```



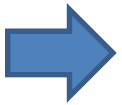
```
$flang -O3 -g -S -emit-llvm test.f90 -o- | awk /DISPFlagOptimized/
$
```

PR865: Corrected emission of isOptimized flag

Status: Approved

PR Description: flang is generating "isOptimized" flag as false in LLVM IR. Regardless of the whether it's an optimized build "-O2 -g or -O3 -g" or Un-optimized build "-g" or "-O0 -g"

```
SUBROUTINE sub(foo_arg)
    integer,value :: foo_arg
    print*, foo_arg
END SUBROUTINE
```



```
$flang -O3 -g -S -emit-llvm test.f90 -o- | awk /isOptimized/
$ isOptimized: false
```