Fortran alias in LLVM

An experiment

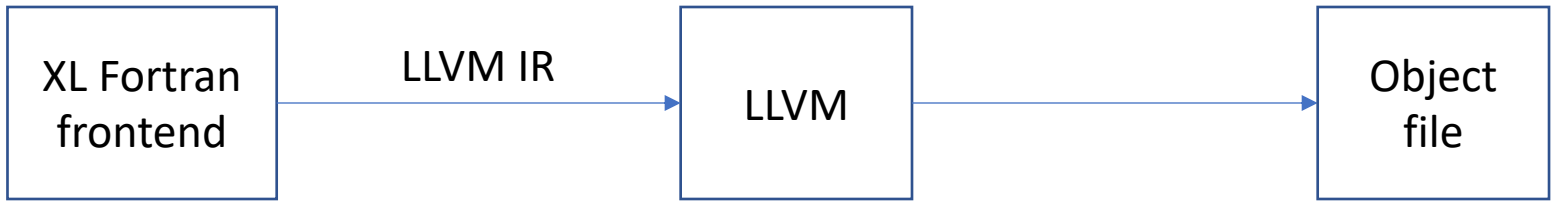
Kelvin Li (kli@ca.ibm.com)
Rafik Zurob (rzurob@ca.ibm.com)
Tarique Islam (tislam@ca.ibm.com)

Agenda

- Background
- Observations
- Next step

Background

- Problem: Representing Fortran alias information in LLVM IR
- Compilation:



- Representation of alias information in IR
  - Considered TBAA and ScopeAlias/NoAlias metadata (MD)
  - Used ScopeAlias/NoAlias MD, as Fortran rules result in more refined aliasing than the natural TBAA representation

Alias representation

- A memory instruction (load, store, call) is annotated with `alias.scope` and `noalias` MD
- `alias.scope` MD is a set of MD nodes where each element node represents a symbol that may alias the memory read from/written to
- `noalias` MD is a set of MD nodes where each element node represents a symbol that is definitely not aliased with the memory read from/written to

Alias representation: Example

```
subroutine sub(arg)
 integer :: arg
 integer, target :: tgt
 integer, pointer :: ptr
 ptr => tgt
 tgt = arg
 ptr = tgt + 1
end subroutine sub
```

```
Alias(arg)  : {arg}
Alias(tgt)  : {tgt, ptr}
Alias(ptr)  : {tgt, ptr}
Alias(d-ptr): {d-ptr, d-ptr%addr}
...
```

Alias representation: Example

```
subroutine sub(arg)
 integer :: arg
 integer, target :: tgt
 integer, pointer :: ptr
 ptr => tgt
 tgt = arg
 ptr = tgt + 1
end subroutine sub
```

```
Alias(arg)  : {arg}              NoAlias(arg): {d-ptr%addr, tgt, ptr, d-ptr}
Alias(tgt)  : {tgt, ptr}         NoAlias(tgt): {d-ptr%addr, d-ptr, arg}
Alias(ptr)  : {tgt, ptr}         NoAlias(ptr): {d-ptr%addr, d-ptr, arg}
Alias(d-ptr): {d-ptr, d-ptr%addr}
...
```

Alias representation: Example

```
subroutine sub(arg)
 integer :: arg
 integer, target :: tgt
 integer, pointer :: ptr
 ptr => tgt
 tgt = arg
 ptr = tgt + 1
end subroutine sub
```

```
Alias(arg)  : {arg}              NoAlias(arg): {d-ptr%addr, tgt, ptr, d-ptr}
Alias(tgt)  : {tgt, ptr}         NoAlias(tgt): {d-ptr%addr, d-ptr, arg}
Alias(ptr)  : {tgt, ptr}         NoAlias(ptr): {d-ptr%addr, d-ptr, arg}
Alias(d-ptr): {d-ptr, d-ptr%addr}
...
```

```
!2 = distinct !{!2, !"domain: sub"}
!4 = distinct !{!4, !2, !"d-ptr%addr"}
!6 = distinct !{!6, !2, !"tgt"}
!8 = distinct !{!8, !2, !"ptr"}
!11 = distinct !{!11, !2, !"arg”}
!23 = distinct !{!23, !2, !"d-ptr"}
!21 = !{!11}                    ; {arg}
!22 = !{!4, !6, !8, !23}        ; NoAlias(arg)
!24 = !{!6}                     ; {tgt}
!25 = !{!4, !23, !11}           ; NoAlias(tgt)
                                ; NoAlias(ptr)
!26 = !{!8}                     ; {ptr}
```

Alias representation: Example

```
subroutine sub(arg)
 integer :: arg
 integer, target :: tgt
 integer, pointer :: ptr
 ptr => tgt
 tgt = arg
 ptr = tgt + 1
end subroutine sub
```

```
Alias(arg)  : {arg}                NoAlias(arg): {d-ptr%addr, tgt, ptr, d-ptr}
Alias(tgt)  : {tgt, ptr}           NoAlias(tgt): {d-ptr%addr, d-ptr, arg}
Alias(ptr)  : {tgt, ptr}           NoAlias(ptr): {d-ptr%addr, d-ptr, arg}
Alias(d-ptr): {d-ptr, d-ptr%addr}
...
```

```
!2 = distinct !{!2, !"domain: sub"}
!4 = distinct !{!4, !2, !"d-ptr%addr"}
!6 = distinct !{!6, !2, !"tgt"}
!8 = distinct !{!8, !2, !"ptr"}
!11 = distinct !{!11, !2, !"arg”}
!23 = distinct !{!23, !2, !"d-ptr"}
!21 = !{!11}                    ; {arg}
!22 = !{!4, !6, !8, !23}        ; NoAlias(arg)
!24 = !{!6}                     ; {tgt}
!25 = !{!4, !23, !11}           ; NoAlias(tgt)
                                ; NoAlias(ptr)
!26 = !{!8}                     ; {ptr}
```

Alias representation: Example

```
subroutine sub(arg)
 integer :: arg
 integer, target :: tgt
 integer, pointer :: ptr
 ptr => tgt
 tgt = arg
 ptr = tgt + 1
end subroutine sub
```

```
!4 = distinct !{!4, !2, !"d-ptr%addr"}
!6 = distinct !{!6, !2, !"tgt"}
!8 = distinct !{!8, !2, !"ptr"}
!11 = distinct !{!11, !2, !"arg”}
!23 = distinct !{!23, !2, !"d-ptr"}
!21 = !{!11}                    ; {arg}
!22 = !{!4, !6, !8, !23}        ; NoAlias(arg)
!24 = !{!6}                     ; {tgt}
!25 = !{!4, !23, !11}           ; NoAlias(tgt)
                                ; NoAlias(ptr)
!26 = !{!8}                     ; {ptr}
```

```
%_val_arg_ = load i32, i32* %_val_.arg_, align 4,          !alias.scope !21, !noalias !22
store i32 %_val_arg_, i32* %tgt, align 4,                  !alias.scope !24, !noalias !25  ; tgt = arg
...
%_val_tgt_ = load i32, i32* %tgt, align 4,                 !alias.scope !24, !noalias !25
%_add_tmp = add nsw i32 %_val_tgt_, 1
store i32 %_add_tmp, i32* %"_val_d-ptr%addr_", align 4,    !alias.scope !26, !noalias !25  ; ptr = tgt + 1
```

Alias representation: Example

```
subroutine sub(arg)
 integer :: arg
 integer, target :: tgt
 integer, pointer :: ptr
 ptr => tgt
 tgt = arg
 ptr = tgt + 1
end subroutine sub
```

```
!4 = distinct !{!4, !2, !"d-ptr%addr"}
!6 = distinct !{!6, !2, !"tgt"}
!8 = distinct !{!8, !2, !"ptr"}
!11 = distinct !{!11, !2, !"arg”}
!23 = distinct !{!23, !2, !"d-ptr"}
!21 = !{!11}                    ; {arg}
!22 = !{!4, !6, !8, !23}        ; NoAlias(arg)
!24 = !{!6}                     ; {tgt}
!25 = !{!4, !23, !11}           ; NoAlias(tgt)
                                ; NoAlias(ptr)
!26 = !{!8}                     ; {ptr}
```

```
%_val_arg_ = load i32, i32* %_val_.arg_, align 4,          !alias.scope !21, !noalias !22
store i32 %_val_arg_, i32* %tgt, align 4,                  !alias.scope !24, !noalias !25  ; tgt = arg
...
%_val_tgt_ = load i32, i32* %tgt, align 4,                 !alias.scope !24, !noalias !25
%_add_tmp = add nsw i32 %_val_tgt_, 1
store i32 %_add_tmp, i32* %"_val_d-ptr%addr_", align 4,    !alias.scope !26, !noalias !25  ; ptr = tgt + 1
```

Alias representation: Example

```
subroutine sub(arg)
 integer :: arg
 integer, target :: tgt
 integer, pointer :: ptr
 ptr => tgt
 tgt = arg
 ptr = tgt + 1
end subroutine sub
```

```
!4 = distinct !{!4, !2, !"d-ptr%addr"}
!6 = distinct !{!6, !2, !"tgt"}
!8 = distinct !{!8, !2, !"ptr"}
!11 = distinct !{!11, !2, !"arg”}
!23 = distinct !{!23, !2, !"d-ptr"}
!21 = !{!11}                    ; {arg}
!22 = !{!4, !6, !8, !23}        ; NoAlias(arg)
!24 = !{!6}                     ; {tgt}
!25 = !{!4, !23, !11}           ; NoAlias(tgt)
                                ; NoAlias(ptr)
!26 = !{!8}                     ; {ptr}
```

```
%_val_arg_ = load i32, i32* %_val_.arg_, align 4,          !alias.scope !21, !noalias !22
store i32 %_val_arg_, i32* %tgt, align 4,                  !alias.scope !24, !noalias !25  ; tgt = arg
...
%_val_tgt_ = load i32, i32* %tgt, align 4,                 !alias.scope !24, !noalias !25
%_add_tmp = add nsw i32 %_val_tgt_, 1
store i32 %_add_tmp, i32* %"_val_d-ptr%addr_", align 4,    !alias.scope !26, !noalias !25  ; ptr = tgt + 1
```

Observations

- No performance degradation for the workloads that we ran.
- Achieved moderate to significant improvement in execution performance for several workloads (2% – 200%).
- Compile-time performance degraded significantly for some workloads (1x – >500x).
- The size of the IR increased significantly (1.4x – 10x).

	compile time increase	performance improvement	IR size increase
workload 1	1.8x	1x	1.4x
workload 2	4x	1.2x	3.2x
workload 3	15.5x	1x	2.5x
workload 4	6.6x	1.9x	8.7x

Observations

- Compile-time performance
  - Each alias query (`ScopedNoAliasAAResult::mayAliasInScopes`) involves partitioning an MD set based on the domains of the MD elements. Pre-partitioning the MD sets and maintaining the partitions on updates can help.
  - Intersection of `noalias` sets is $O(n^2)$ as MD elements do not have any ordering. Defining some order on the elements can help significantly.
  - Some optimizations do not scale well when the size of the working instruction set increases. Example: SCEV functions.
- Size of LLVM IR
  - The `noalias` MD requires a flattened set of MD nodes. A hierarchical representation can reduce memory footprint.

Next steps

- Goal: An LLVM representation that is a natural fit for Fortran aliasing and that scales for large workloads.
  - In terms of expressive power, ScopedAlias/NoAlias seems more suitable for Fortran than TBAA.
  - ScopedAlias/NoAlias in its current state does not scale. Are there any suggestions to enhance the metadata implementation in LLVM to achieve more efficient alias operations (e.g. query, intersection) and a smaller memory footprint?

- Call for collaboration
  - We are interested in learning about other Fortran aliasing experiments and observations.
  - We would like to further explore either enhancing or replacing the LLVM representation to make it work better for Fortran. We would love to work closely with the community on this.

Back Up