Function Signatures

A key job of the IREE compiler and runtime is capturing function call semantics from the originating system and providing mechanisms so that invocations can be performed in as similar way as possible in various target languages. In general, this requires additional metadata on top of the raw characteristics of a function. Where possible, this is done by attaching attributes to a function.

- `iree.abi` : JSON encoded description of the function's calling convention.

V1 ABI

This is the default ABI supported by the IREE VM invocations. It attempts to provide a default calling convention that can be used without further reflection metadata but which may be enhanced with it.

It natively allows monomorphic functions to be exported where arguments and results are composed of the following types:

Value Types:

- Byte aligned integer type (i8, i16, i32, i64)
- Floating point value (f16, f32, f64)

Reference Types:

- ND-Array buffers of Value Types:
    - Simple: Packed, C-layout
    - Strided: Arbitrary layout with strides (future)
- String (byte arrays)
- Opaque reference object

Sequence Types:

- Tuples: fixed length lists where each position has its own type bound
- Homogenous list: lists of arbitrary size where a single type bound applies to all elements

The intent with these low level types is that calling conventions can be synthesized to bind arbitrary high level, domain/language specific signatures to these types, possibly by way of additional reflection metadata.

Representations:

The above are all representable with native constructs in the VM:

- ValueType:
    - Runtime: `iree_vm_value`
    - Compile Time: primitive MLIR integer/floating point types

- Simple ND-Array Buffer:
  - Runtime: `iree_hal_buffer_view`
  - Compile Time: `tensor<>`
- String:
  - Runtime: `iree_vm_list` containing `i8`
  - Compile Time: `!util.list<i8>`
- Tuple:
  - Runtime: `iree_vm_list` of variant
  - Compile Time: `!util.list<?>`
  - Note that these are statically type erased at the boundary.
- TypedList (homogenous):
  - Runtime: `iree_vm_list` of `T`
  - Compile Time: `!util.list<T>`

Extended Type Calling Conventions

While the above features of the native ABI may be sufficient for direct use by various programs, many programs and callers will need to represent various higher level types, consistently mapping them to the above facilities. This section describes calling conventions for various higher level types which do not map 1:1 to the above. Not all source language types are representable, and extending these calling conventions (and the fundamental types above) is demand driven.

All of these calling conventions presume that the arity of the arguments/results of the raw function matches the user-level function, meaning that the calling convention is specified per argument/result. Higher-level whole function transformations may also exist for some domains but are outside of the scope of this specification.

Structure

A `Structure` is a common enough entity to have a dedicated calling convention. In C-like languages, this may just be a `struct`. In Python, it is typically a `dict` with an associated schema providing a name and type bound for each of its slots. In both, its slots are of fixed arity.

In this convention, such a structure is represented as a `Tuple` in the native calling convention (i.e. `!util.list` of variant type). The order of the elements of the tuple are the natural order of the structure, where that is either:

- For a C-like system where order is determinate, it is the order of declaration.
- For a name-based system (i.e. bind to `dict`) where no order is defined, the natural order will be the lexically sorted order of the keys.

String

Most languages interop between byte arrays (i.e. the native ABI `String` type) by way of applying an encoding. Such strings are just a sequence of bytes (i.e. `!util.list<i8>`).

Typed List

High level lists which all share the same type bound are represented as a `TypedList` in the native ABI.

NDArray of Reference Types

NDArrays of reference types are considered separately from those of value types. Internally, the code generated for them is completely different from what gets generated for numeric based arrays (i.e. has ref-counting, ownership semantics, non-POD, etc). These types are permitted for completeness, not necessarily performance: by nature they are already indirected and have overheads.

In the native ABI, these are represented as a composite tuple type (i.e. today a list since sugar for tuple is not yet defined): `!iree.tuple<!util.list<T>, !util.list<index>>`. The first element of the tuple is the list of values, packed with a C-Layout and the second element is the list of dimension sizes.

Reflection

Additional reflection metadata may be encoded in a custom JSON form, providing additional typing hints for arguments and results. If present, this will be a reflection attribute with key `d`, containing a serialized JSON object.

The JSON object contains:

- `a` (array): List of type records for each argument.
- `r` (array): List of type records for each argument.

Type records are one of:

- A string naming a primitive type:
    - `i[0-9]+`: Integer type with given bit width
    - `f[0-9]+`: IEEE floating point type with given bit width
    - `bf16`: BFloat16
- JSON `null`: A null reference value
- `"unknown"`: An unknown/unmapped type
- An array, interpreted as a tuple describing a compound type.

Compound type tuples

A compound type tuple has a type identifier as its first element, followed with type specific fields:

- `["named", "key", {slot_type}]`: Associates a name with a slot. This is used with the root argument list to denote named arguments that can be passed positionally or by keyword.
- `["ndarray", {element_type}, {rank}, {dim...}]`: For unknown rank, the `rank` will be `null` and there will be no dims. Any unknown dim will be `null`.

- `["slist", {slot_type...}]`: An anonymous structured list of fixed arity and slot specific types. If there are gaps in the list, empty slots will have a `null` type.
- `["stuple", {slot_type...}]`: Same as `slist` but some languages differentiate between sequences represented as lists and those represented as tuples (read-only lists).
- `["sdict", ["key", {slot_type}]...]`: An anonymous structure with named slots. Note that when passing these types, the keys are not passed to the function (only the slot values).
- `["py_homogeneous_list", {element_type}]`: A Python list of unknown size with elements sharing a common type bound given by `element_type`.