2024-07-18 (ModArith Discussion)

- * start recording *
- Context: Modular arithmetic representing in HEIR
    - How do hardware accelerators and software backends with modular representation assumptions baked in - HW accelerators may also require montgomery representation from the start
    - Software world - standard lowerings to arith / LLVM that assume semantics about ops - what are the limitations here?
- Goal:
    - Understand coexisting requirements and document for later passes to adhere
    - How do we represent types and ops with specific modular representations? Right now we have arith_ext, can we design something cleaner?
- arith_ext problems:
    - Asserting and modeling the range that an integer type starts in (barrett reduction pass, HeaNN passes that also use sub if ge optimization)
- Ideas for how to model / assert range:
    - Could track integer range via analysis pass that uses info from reduction ops and attributes
        - Cons: Other passes may not preserve attributes
        - Pros: One time pass
        - Questions: add a hook to update analysis on values during IR transforms?
- arith problems:
    - Builtin passes that may rely on i32 wraparound semantics rather than modular-ly reduced
    - *Still need modular semantics on arithmetic*
- In hardware, what are the container types? Still power of two word sizes?
    - Not sure about optalysys photonics hw?
    - Alex: still a fixed number of bits
    - Maybe we have attributes with the modulus operand but still uses container integer type
    - Using integerlike types would still be fine, as long as we have custom dialect
    - integerlike types are also hard-coded in the polynomial type, conversion from poly to standard would be as easy as pulling the cmod modulus from ring attribute into a mod_arith op attribute
- No way to tell whether something is normal modular arithmetic or special (montgomery mod), and that changes the lowering to arith
- If the input is already modularly reduced, we have no way of telling it's range (barrett bound q^2 or other e.g. q or q/2)
- Do we need func attributes marking known range?
    - probably not
- Mod arith op attributes can be used in analysis pass or as instructions on how to lower:
    - re.g. range attribute indicates how: random.sample { range = [0, q] } is lowered
- What are the size of the inputs of mod_arith?
- representation forms:
    - mod_arith forms: barrett, montgomery<scalar>, reduced (0 to q)
- Modular arithmetic operation contracts:

    -
- Steps:
    - Rename arith_ext to mod_arith
    - Add representation attributes
    - Extend integer range analysis to work on tensors and understand mod arith attributes
    - Extend RNS

```
func.func(arg0 : int16 { mod_arith.reduced })
func.func(arg0 : int16 { mod_arith.UNKNOWN }):
        %reduced_arg0 : mod_arith.reduce arg0 {mod_arith.reduced} : int16
```