Compilers & Hardware Accelerators: MLIR Dialects

HomomorphicEncryption.org Working Group 2

Date: 2023-06-27
Scribe: Jeremy Kun

Video recording:
https://drive.google.com/file/d/1S893iJdeL08BAonLenmkUGNC34PX1qpg/view?usp=sharing

Notes

- If a dialect can lower to two different semantics, it's buggy.
- Goal: ONNX of FHE (or rather StableHLO). Not to build a full compiler
- Should we do it upstream in MLIR?
  - Others can fix build issues as they come up
  - Lenient policy once we get the initial dialect shell in
  - Helps with buy-in
- Give ourselves a month or two to agree on what we want to upstream
  - Keep HEIR repo as maybe holding specific lowerings, extra batteries, and a testing ground for experimental things before they can get upstreamed
  - Some more private IP involved in lowerings, crypto innovations
  - Everyone has their own hardware and so their lowerings disagree
  - Some strange dependencies get pulled into lowerings/optimizations (yosys/abc, lwe estimator, etc.)
  - Maybe in a year or two we could unify once the details are sorted out.
- Dialects discussion
  - Polynomial ring arithmetic dialect
  - HEaaN: poly operations like NTT, sampling, pre-defined ring parameters
  - HECO: arith for FHE (signless integer type constraint)
  - Concrete: similar eint arith dialect, and lower level implementation-tracking versions
  - Approximateness and error tolerance
    - Three types of error:
      - FHE encryption noise: analysis
      - Encoding/decoding noise: analysis
      - Program approximation as an arithmetic circuit / Lookup table precision
    - Types are expensive to update/handle
    - Something like multiplication is handled very differently in the different schemes
      - Bit-wise version
      - Vector operation and some heavy operations to get it back into normal form

            - Univariate lookup table trick
            - Specify what accuracy is acceptable on an “fhe.mult” in order to allow the lowerings to do their magic.
            - If we say +/- 1 bit tolerance no an the.mult, does the theory allow us to easily hold this
        - Quantum computing related noise tracking ideas?
        - Noise analysis is not sophisticated in the MLIR ecosystem, FHE might be a nice motivation to do it better, and find a bridge to QC.
    - Take high level dialect with approximately (fhe arith) and compare that with a more radical approach to “fix arith”
        - Arith is left-over historical artifact from “standard dialect” days. Semantics are poorly defined, and OK in some applications but not in crypto where you need to know what happens when you have an overflow, or multiple additions/mults
        - secret<X> that works with arith
    - Scheme specific dialects: we can start with more quickly
        - Schemes:
            - BGV/BFV
            - CGGI/tfhe-rs
            - CKKS
        - Could follow similar approach to CIRCT, where things started out with individual dialects (e.g. trivial dialect targeting specific libraries) and then start factoring out common parts, and slowly transition to universal dialect
    - Easier and more general purpose to upstream:
        - Polynomial arithmetic
        - ‘High level’ FHE arith (“secret”)
    - Module level attribute for crypto parameters vs parameters on the type
    - IR should be self-contained so you can recover everything you need
    - Region-level parameters
    - Poly
        - Focus on our use case with an eye to generality
        - One or two steps away from being fully general
        - Take offline
- Hardware
    - Standardize hardware interfaces for FHE
    - Focus more with the hardware team
- Poly ops that aren’t just ring ops
    - Rounding
    - Arbitrary permutation of coefficients
    - Coefficient extraction
    - NTT (not strictly within the ring)
    - Evaluation
    - CRT/RNS style decompositions of ring coefficients
- github.com/google/heir

    - Issues: for concrete work tasks
    - Discussions: for initial discussion, pull into issues as work is established
        - Register interest in particular workstreams
    - OK to sign CLA? If not, we will have to find another solution
    - Since we already have the goal to upstream, figure out what is patch #1 (limit 200-300 lines), MVP to commit to MLIR.
- Lean formalization: https://github.com/leanprover-community/mathlib4/tree/master/Mathlib/RingTheory/Polynomial/Cyclotomic
-

Background Info

In case people aren't familiar with the different MLIR-based FHE projects, here is some info:

- *Concrete*: github.com/zama-ai/concrete for code, Blog Post for a bit of an explanation, and direct link to the existing Dialects
    - Represented by Quentin Bourgerie

- *Transpiler*: github.com/google/fully-homomorphic-encryption for code,Blog Post on the compiler, and direct link to (future) dialects
    - Represented by Jeremy Kun

- *HECO*: github.com/marbleHE/HECO/tree/dev for code, paper pdf (it's not quite a blog, but pretty high-level if you skip the optimization chapter), and direct link to the dialects
    - Represented by Alexander Viand

- *HEaaN.MLIR*: Paper, LLVM DevMtg Slides, code is (afaik) not available.
    - Represented by Sanghoon Park (CryptoLabs)

- *HECATE*: Paper, code is (afaik) not available