HEIR meeting notes 2025-08-07

2025-08-07

- HEIR PRs merged since last time (60)
  - AES CGGI demo #1765
  - CRT support in mod_arith #1928
  - Layout conversion cost analysis #1967
  - LWE type migration finished #1972 #2036 #2040
  - Minor improvements for/from HEIR hackathon #1978 #1979 #2007
  - Python frontend supports MLIR string as input #1984
  - Autogenerated documentation examples from lit tests #1987 #2031 Example screenshot below **(help wanted to add more: #2014)**
  - arith.negf -> ckks.negate lowering #1989
  - Reorganize developer docs #1994
  - Lowering comparisons in arithmetic pipeline #1999
  - OpenMP enabled by default now #2010 (nb., openmp not supported on macOS...)
  - Start to lower ckks.relinearize #2013
  - Worst case CKKS range analysis #2018
  - Update C++ standard to C++20 #2025, drop support for clang 15/16 and add clang 20 #2049
  - Work on new layout system #2048
  - New builder style from upstream MLIR (now with 100x better IDE support!) #2060
  - Update CGGI pipeline for e2e loop test #2056
  - Generalized rotate and reduce op for baby-step-giant-step #2071
- In progress PRs
  - Add ISL dependency for client side/plaintext packing codegen #2072
  - First pass layout codegen that will be replaced by ISL #2063
  - Dockerfile for HEIR dev #2055
  - Disable openmp on macos #2046
  - Remove memrefs from CGGI pipeline #1910
  - Initial SBox implementation #2012
  - Scheme selection pass #2043
  - Generalized baby-step-giant-step lowering for rotate-and-reduce op
  - Migrate layout-propagation to new layout attributes
- Notable new issues
  - New papers
    - Aegis #2064
    - CHESS #1970
    - Rotom #2035
    - EinHops #1995

        - "Accurate and Composable Noise Estimates for CKKS with Application to Exact HE Computation" #2011
    - Generalize affine-map layout notation to use IntegerRelations #2047
    - Scheme Selection passes #2019
    - [lwe] add dimension / size to the key attribute #2045
    - Add a packaged script to lower PyTorch / TF models to HEIR inputs (linalg) #2038
    - Generalized baby-step giant-step pass #2027
    - Slow input example for loop over dot products #2008
    - Fun issues noticed during Hackathon #2005
    - pipelines should abort after the current pass if it created errors #2000
    - Unexpected segfault after compilation error in --convert-to-ciphertext-semantics #1992
    - Add pass/pipeline pre-conditions/post-conditions #1985
    - Enable a dry-run cibuildwheel to be triggered from a PR #1969
- Topics
    - ISL - jkun
    - Lowering core crypto primitives in HEIR - alex viand
        - Finish lowering relin
        - Most efficient keyswitch implementation
        - Special topics meeting soon?
        -
    - Scheme selection pass
        - Starting point for scheme switching
        - Guidance for inexperienced users who don't know different schemes
        - Has to be done early
        - Scheme switching will partition code and do scheme selection on each part
        - No real cost models comparable across all schemes
            - github.com/heir-compiler/heir has partitioning + cost model?
            - Seems hard coded
        - Simple cost model: infinite cost if not implemented 🙂
        - Infra side: how can we support work on this, no way to expose what is supported to the cost model?
        - Maybe first step is to consider cost tradeoff for simple IRs
            - IR with a single comparison.
            - Matvec mul + small decision tree
        - Currently counts ops by type, annotate func with counts, hard coded decision based on that
        - Maybe start by just running all these ops to get a blended cost?
        - Openfhe-binfhe (public fork, haven't had time to upstream)
            - Definitely won't be the best CGGI implementation
            - Still want to lower CGGI to LLVM (with PBS) to do proper scheme switching

            - But openfhe-binfhe upstreaming should only take ~1 day and having it as a playground for working on scheme switching will be useful.
    - How can we support profiling/benchmarking in first-party?
        - Define an output format for profiling information
        - Low level instrumentation seems hard but necessary for ASICs/FPGAs
        - For library backends can we just insert timing code around each statement?
        - Timing.start timing.stop ops and emit at high level
    - Can we benchmark NTT-lowering in HEIR vs lattigo?
    - Can we cheaply enable AVX extensions in HEIR after lowering to LLVM? https://stackoverflow.com/questions/22548397/can-i-generate-avx-vectorized-code-using-llvm-jit
- https://github.com/j2kun/patfind
- FYI: LLVM Developer Meeting 2025 registration is open: https://llvm.swoogo.com/2025devmtg

```
Timing.time {
Linalg.matvec
}

Timing.time “linalg.matvec” {
Timing.time “tensor_ext.rotate” {

}
Timing.time “arith.addi” {

}
...
}

Timing.start
linalg.matvec
Timing.end

Timing.start “linalg.matvec”
linalg.matvec
Timing.end “linalg.matvec”
```

Example

Command: `heir-opt --polynomial-approximation tests/Transforms/polynomial_approximation/doctest.mlir`

Input:

```
func.func @test_exp(%x: f32) -> f32 {
  %0 = math.exp %x {
      degree = 3 : i32,
      domain_lower = -1.0 : f64,
      domain_upper = 1.0 : f64} : f32
  return %0 : f32
}
```

Output:

```
#ring_f64 = #polynomial.ring<coefficientType = f64>
!poly = !polynomial.polynomial<ring = #ring_f64>
module {
  func.func @test_exp(%arg0: f32) -> f32 {
    %0 = polynomial.eval #polynomial<typed_float_polynomial <0.99457947632469512 + 0.9956677100276301x + 0.542972788
    return %0 : f32
  }
}
```