Compilers & Hardware Accelerators

HomomorphicEncryption.org Working Group 2

Summary

- Set expectations and guidelines for discussion
- Discuss two breakout groups, one for defining IRs in MLIR and one for HW acceleration cost models and integration
- Three conjectures to discuss: "FHE is not one thing (even within RLWE schemes)", "FHE should not always be looked at as a circuit", "We don't yet know how to generalize decision making across particular instantiations"
- Compare our goals with ONNX
- https://github.com/google/heir will be the home for the standard IRs

Notes

- Purpose of the meetings: increase collaboration between hardware and compiler developers within and between groups. Should be looking at standardizing IRs for FHE, abstractions/cost models for hardware, collect and make accessible folk lore about HW solutions, develop meaningful benchmarks for compilers and hardware, in collab with benchmark group
- Originally started last year after HACS workshop, informal meetings with various folks & ETH-Google.
- Carry over expectations and guidelines from HACS
    - Please demonstrate respect for others at all times
    - Be present and listen to others with focus
    - Fine to debate, part of the point, but after a certain point of debate, be ready to agree to disagree when you reach that point.
    - Try to speak clearly and avoid jargon, given all of our different background (except FHE)
    - Try and make one point per speak, try to speak 1/n of the time for n people.
- Mostly discussions, subgroups, for IR and for hardware cost models, etc.
- How frequently should we meet, does this time slot work?
- Introductions:
    - Jeremy Kun, Google, here for IR development & providing eng support for everyone else
    - Alex Viand, Intel, ensure there's a good ecosystem of FHE resources and software
    - Florent Michel, Optalysys, new HW for accelerating FHE, here to see what is the state on the compilers and how we can work together w/ HW community
    - Yongwoo Lee, share about abstractions on FHE
    - Asra Ali, Google, help develop unified IR

    - Michiel, KU Leuven, HW for FHE, FPGA field programmable gate array, here to assist in HW stuff, interested in intersection with compiler world
    - Shruthi Gorantala, Google, here for IRs, MLIR, benchmarks
    - Robin Geelen, KU Leuven, acceleration of BGV and CKKS
    - Vinu Joseph, NVIDIA, accelerating primitives on NVIDIA stack CPU/GPU
- Let's include someone from Zama for next time.
- Let's assume everything in these meetings are public; let's assume not attributing anything you say to your company by default
- This is hard for standardization, but profitable time, because if we do this right we save ourselves a lot of reinventing the wheel.
- FHE is not one thing. RLWE (Ring Learning With Errors) (also in re DPRIVE) are all different in how you implement/use them. Very different from CGGI branch (binary gates/low bit-width lookup tables). HW Acceleration efforts are also very diverse. Alex is interested to hear what is new outside of DPRIVE land, i.e., GPU.
- People like to think of FHE programs as circuits, but they're decidedly not. Many modern FHE programs make use of non-circuit features.
- Unavoidable fact (modulo breakthrough) that all FHE computations are inherently noisy; LWE noise, or semantically noisy a la CKKS.
- All nontrivial FHE computations require quantization or approximation. Neither option of RLWE or CGGI options are a one-size-fits-all.
- Given any fixed set of parameters (input, HW, schemes, parameters), we know how to make a good setup, but we don't know how to generalize across them.
- Is/should FHE be viewed as a circuit?
    - Shruthi: static upper bounds matters, rather than circuit specifically.
    - Alex: Difference between being a circuit and being equivalent to a circuit model.
    - Alex: for the IR there are practical differences, but does it really matter.
- Do we know how to generalize across parameters?
    - Asra: agree but does it just depend on the …
    - Alex: depends on the scheme: CKKS look at precision and crank up modulus; TFHE might be different; bootstrap BGV has this weird tradeoff between how much time you spend computing the bootstrap vs the actual approximations.
    - Alex: can't expect people off the street to know all of these weird tradeoffs. Fairly obvious good answers for specific scheme instantiations. Writing a program once and retargeting across schemes seems harder. Folks like Rachel Player @ Royal Holloway are interested (she couldn't make it today)
    - Michiel: can relate to from DPRIVE program. Lots of choices in mapping benchmark to algorithm, how to approximate/quantize.
    - Jeremy: why?
    - Michiel: coupling between how you might implement things and how you quantize.
    - Alex: for one hardware for one scheme it's already hard. Nobody tells you what part of the input program is relevant. How much can I throw away before the task is no longer meaningfully related.
- What can we do going forward?

    - No longer toy compilers; have to deal with a lot of different stages of the process. Taking an input program and representing it so it can be done in FHE. Once you have them, you can reorder/optimize the circuit to make it faster. Then how do you implement the atomic FHE ops (like key switch). Then the HW target optimization.
    - Normal compiler world, LLVM came to the rescue. MLIR now for multi level intermediate representation; breaking down into small things. Can support higher level constructs like linalg. HECO has some FHE-specific dialects.
    - Right now nobody targets accelerators with MLIR, but idea is that the accelerators could slot into the whole view.
    - MLIR: Can easily define new IR dialects with types, operations, etc, that represents a higher level sort of assembly language. SSA=single-static assignment form (can't override variables). Static type system. MLIR defines the syntax, but no semantics inherent in the system. Lowering reduces high level ops to lower level ops until you are at the machine code. For FHE, you'd translate BFV mult into a bunch of polynomial multiplications, and one of those would be converted to a bunch of NTTs, and each would break further down into RNS ops. Then you can canonicalize/optimize to avoid inefficient steps.
    - All the cool proprietary math magic happens in the lowering/optimizing passes. No need to leak the math if you don't want to. If we standardize on the dialects and types, your project can use a different lowering and you can keep it private.
    - You only need to define some small amount of custom stuff specific to your domain, because we get in the built-in language stuff like memory references, arrays/tensors, loops, etc.
    - Discussing with Google: what should these dialects look like?
        - Zama has some dialects as implemented in concrete
        - Alex Viand did some that are roughly copies of the SEAL API.
        - Google would do similar if it was in a vacuum.
- Concrete tasks:
    - Standardize dialects for each of the schemes and the underlying math (ring, etc.)
    - Who is interested in this specific sub-area of the topic?
    - github.com/google/heir as the home for the dialects/types
    - don't want to upstream them to MLIR, but also don't want them to die in some random repository. Google has advantage that someone's on call to fix MLIR issues upstream with internal projects.
- Vinu: interoperability story?
    - With DL acceleration, one step in standardizing across companies/chips/tools, is the creation of ONNX (open neural network exchange format)
    - Vinu believes they are all circuits (first point above); more true in case of CGGI, but if we can agree on a standard circuit format and what each operator is, then you can exchange your code across companies. Channel of independent optimization. Where in this picture would it stand?
    - Alex: MLIR is a set of tools for dealing with things in the MLIR format. MLIR has efficient textual representation/bytecode, it's relatively easy to convert to/from

MLIR (no harder than some custom format like ONNX). Define what are the operators, etc. Using framework of MLIR.
    - Jeremy: let's use MLIR as the "ONNX" for now, mostly to iterate and figure things out, and use the MLIR textual representation/bytecode representation. Once we have things really settled, we can revisit defining an ONNX-style format that is independent of MLIR.
- Alex: FHE doesn't look like circuits when you consider scheduling. With a circuit you have to reconstruct looping information, but with a program format you can do auto vectorization, parallel scheduling, etc.
- Other breakout group: more work on the hardware side, how do we define hardware targets
    - Q to HW folks: what is the burning question to tackle?
    - Michiel: Take a specific benchmark and compile it down to different accelerators/schemes. Discussions on HE Bench framework (some overlap). Very difficult to compare your results to different work using different parameter sets/schemes.
    - Jeremy: someone has to implement it even if we have an MLIR solution
    - Alex: if one pathway works really well for a specific style (special word size), how do you compare for things that don't make that decisions. What is the last point at which you can still reasonably target different accelerators? How you quantize, e.g., may disadvantage particular accelerators. Obv for schemes, but what can we actually distill that still has some freedom to target different accelerators?
    - Vinu: CuFFT been shipping for a long time, NTT primitive similar. Nature of arch decisions: peak performance device is achieved for some fixed set of parameters. This is aligned to CKKS style (by chance); will see CGGI is worse because of that.
    - Some fundamental issues of benchmarking: is it OK if HW is not always fair to some scheme?
    - Alex: changes to things like montgomery prime vs other have a big impact. Choosing a different scheme is "fair game" if it is worse. But within, say, TFHE, there's lots of knobs with gate bootstrapping vs low-width integers; larger parameters vs larger bit width.
- Scheduling!
    - **Suggestion: Tuesdays every two weeks: alternate breakout groups and joint meeting.**
    - Keep them at an hour.
    - Alex will send out a doodle to everyone (including the people that couldn’t make this slot) to see which weekday & time work best