Equality Saturation in MLIR

COMS 6156 Project Proposal

Eric Feng and Chris Yoon (Group Name: "EricAndChris")

ef2648@columbia.edu cjy2129@columbia.edu

Abstract

Equality Saturation [1] has gained considerable interest as a compiler optimization technique, thanks to performance and usability improvements from recent work such as `egg` [2].

Our project aims to bring e-graph optimizations to the MLIR infrastructure [3], a cutting-edge compiler development framework. In particular, we aim to demonstrate that several classic compiler optimizations – e.g vectorization, global value numbering (GVN), and loop invariant code motion (LICM) – can be effectively implemented via equality saturation. Simultaneously, we aim to leverage this technique to identify opportunities for the application of external (existing) optimization passes within the MLIR framework.

Project Description

As stated above, we aim to bring e-graph optimizations to the MLIR framework. Our current plan is to focus on targeting support for the `scf` dialect. Our choice of MLIR is because it is a cutting-edge compiler development framework whose optimization infrastructure is not yet as mature as its predecessor, LLVM. Our choice of the `scf` dialect is because it is one of the lowest-level IRs of the MLIR stack. This means that optimizations at the `scf` level will be useful for most languages targeting the MLIR stack.

We claim that the value proposition of bringing e-graph optimizations to the MLIR stack is as follows:

- MLIR does not officially support e-graph optimizations, while other modern production-grade compilers have begun to adopt it, such as the *wasmtime* `WASM` compiler [4];
- We demonstrate that a single e-graph pass can perform several classic compiler optimizations, removing the need to implement separate passes for each optimization, and making the addition of new optimizations easier;
- Thanks to the theoretical properties of equality saturation, we guarantee that optimizations performed on e-graphs maintain the integrity of the original programs, whereas manually written compiler passes can contain bugs;
- For optimizations that cannot, or would be very difficult, to be expressed as equality saturation rewrite rules, we demonstrate that e-graphs can still help discover opportunities for the application of existing optimization passes

To bring e-graphs to MLIR, we plan to establish the following pipeline:

1. Define a LISP-like language that captures all semantics of the MLIR `scf` dialect, as required by the `egg` framework
2. Write a translator between the aforementioned language and `scf`, enabling equality saturation through `egg`
3. Express optimizations such as vectorization, loop invariant code motion, and global value numbering as rewrite rules in `egg`
4. Use `egg` to perform the optimizations above and assess their correctness and performance
5. Apply e-class analysis [2] to identify areas suitable for existing optimizations in MLIR (such as loop fusion/fission)

Please note that this outline is high-level and our approach may evolve based on further research. As our work mainly serves as a proof-of-concept, we plan to use existing tools like `egg` for equality saturation. This means most of our development will likely be in the `Rust` programming language, which is not the host language (`C++`) for MLIR. However, to discuss the potential upstreaming of equality saturation in MLIR, analogous efforts must be made in `C++`. This includes building a production-grade equality saturation engine, which sadly does not yet exist natively in `C++`. We leave that as a very hopeful future work.

Finally, we note that similar research is being done by researchers at Intel [5] as part of a workflow for optimizing high-level synthesis (HLS) for embedded programs, but their work is proprietary and not publicly accessible. We aim to eventually open-source our work here and share our contributions to the broader MLIR community, showcasing the advantages of equality saturation with more transparency, and encouraging the development of a production-level equality saturation engine in MLIR's host language.

Deliverables

In summary, our course project will aim to deliver the following:

1. A language and a translator for the `scf` dialect that facilitates equality saturation via `egg`
2. A proof-of-concept IR optimization pass based on equality saturation and `egg` that can perform classic compiler optimizations such as LICM, GVN, and vectorization
3. Benchmarks of a suite of programs optimized with our equality saturation approach, against their non-optimized source. To achieve this, we intend to modify the compilation pipeline of Polygeist, a C/C++ front end for MLIR, to integrate our optimization framework. This will allow for a comparison of its performance against a version of Polygeist without optimizations enabled.

References

[1] Charles Gregory Nelson. *Techniques for program verification.* PhD thesis, Stanford, CA, USA, 1980. AAI8011683.

[2] Max Willsey, Chandrakana Nandi, Yisu Remy Wang, Oliver Flatt, Zachary Tatlock, and Pavel Panchekha. egg: Fast and extensible equality saturation. *Proc. ACM Program. Lang.*, 5(POPL), January 2021. doi: 10.1145/3434304. URL `https://doi.org/10.1145/3434304`.

[3] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. MLIR:

Scaling compiler infrastructure for domain specific computation. In *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages 2–14, 2021. doi: 10.1109/CGO51591.2021.9370308.

[4] Chris Fallin. Cranelift progress 2022, 2022. URL `https://bytecodealliance.org/articles/cranelift-progress-2022`.

[5] Jianyi Cheng, Samuel Coward, Lorenzo Chelini, Rafael Barbalho, and Theo Drane. Seer: Super-optimization explorer for hls using e-graph rewriting with mlir. *arXiv preprint arXiv:2308.07654*, 2023.