Machine Learning Guided Optimizations (MLGO) in LLVM

Johannes Doerfert (moderator), Petr Hosek, Chris Cummins, Aiden Grossman, Mircea Trofin, Zoom: Yundi Qian, Ondrej Sykora, Dibyendu Das, Amir Ashouri, Mostafa Elhoushi, S. VenkataKeerthy
Google uses ML in LLVM on a number of projects:
Google3 (Search, Infra), Fuchsia, and Chrome (on Android)

We use Inlining-for-Size and Register Allocation and we have seen up to 20% size savings and up to 1.5% improvements in QPS respectively

Further Reading: MLGO: A Machine Learning Framework for Compiler Optimization
Lessons learned

1. ML is not a magical silver bullet
   The results can be non-obvious and may require further analysis

2. Performance is critical for productionizing ML techniques
   This includes both model training and use inside the compiler

3. Integrating existing ML frameworks into LLVM can be challenging
   Toolchain vendors often have special requirements (e.g. static linking)

4. ML frameworks have non-trivial set of dependencies
   This complicates software supply chain management, licensing, etc.
New opportunities

Have direct impact, in LLVM, on running code

**Accurate Rewards**
- Effective, predictable training
- Datacenter challenge: parallelism, cache effects

**Maintainability Evolvability**
- *It's really the “make or break”*: acceptable, predictable operational cost
- In-depth research opportunities on real applications (e.g. Model needs retraining? Regression fix how-to? Evolution methodology?)

**Profile Accuracy**
- We are good at fixing hot spots
- ML finds opportunities in across lukewarm places

**Large Scope Optimizations**
- For example currently unapproachable large whole program optimization (LTO)
ML-driven hardware cost prediction

- Observations
  - Applicable to MLIR dialects, LLVM-IR, MIR (Machine IR), asm ...
  - Learned cost models shown to be better than llvm-mca, iaca, … ([1])
  - Can be used to predict HW performance characteristics of entire dataflow graphs ([2],[3])
  - Can be used to evaluate optimization plans (like LLVM VPlan) or search an opt space (autoTVM)
  - Our current work based on NLP-like models with the IR as a text input

References:
4. ML-based Hardware Cost Model for High-Level MLIR, Das et al., llvm-dev Meeting, 2022.
Neural Instruction Combiner

- Instruction Combiner (IC) a critical pass
  - Thousands of instruction-combining patterns
- IC is the most frequently updated component in the LLVM compiler [Zhou et al. 2020]
- NIC has three major components
  - NIC inputter: creates an encoded representation from LLVM IR instruction corresponding to a basic block
  - NIC Converter: (Seq2Seq Neural network model) takes the output from NIC Inputter and generates an equivalent optimized encoded instruction sequence
  - NIC Outputter: converts the NIC Converter output back to full-fledged LLVM IR instruction sequence of a basic block. Also checks translation validity via ALIVE2 tool (among others)
Function Inlining Optimization

Pros:
- Reduces overhead due to entering and exiting functions.
- Eliminates the instruction required to function calling
- Doesn’t need registers to pass arguments (reduces register spilling)
- Opens opportunities to subsequent optimizations
  - e.g., Constant propagation, hoisting out part of the function in LICM, expand the scope of register allocation

Cons:
- Increases code size
- Larger code size reduces the temporal locality
  - Thus, decreases the performance of the instruction cache
We extended MLGO to target **performance** rather than code-size reduction.

**MLGOMPerf** employs **two ML models**, first of which (**IR2Perf**) predicts the function speedup post-inlining to help generate the rewards needed to train the second for which makes the decision to whether or not to inline a callsite inside LLVM’s function inlining.

- We trained the first model, i.e., IR2Perf, by leveraging our autotuner to generate +300k meaningful inlining configurations using SPEC CPU2006 on aarch64. We do so by generating 20 handcrafted features we designed and tested.
- We leveraged IR2Perf to train the second model, our RL agent, for more than a million iterations in a matter of few days which otherwise wouldn’t be possible without.
void caller() {
    statement1;
    statement2;
    statement3;
    function1();
    callee();
    for(int i=0; i<10; i++) {
        for(int j=0; j<5; j++) {
            statement;
        ...
    }
}
1. Results show that on average MLGOPerf is able to outperform O3 by 1.8% and 2.2% on SPEC CPU2006 and Cbench when tested on aarch64.

2. Additionally, MLGOPerf provides more opportunities for subsequent optimization passes, i.e., loop unroll and loop vectorize, and an autotuning experiment reveals we can gain at a faster rate and up to 3.7% improvement with respect to O3.
Ondrej - ML for performance modeling

- ML models for throughput prediction
- Input: assembly-like code, basic blocks
- Output: inverse throughput prediction
- Current state:
  - State of the art in ML modeling
  - Graph neural net end-to-end model, no feature engineering
  - Independent library, plays nicely with LLVM
  - Open-source version: soon!
- Future work & research interests
  - Search for practical applications
  - Transfer learning for faster training
  - Analysis beyond basic blocks
MLRegalloc

- ML heuristic for the live range eviction problem
- Worked on adding instruction-based features
  - Also opened doors for graph-based features
- No new performance gains (currently), but some interesting results
  - Just instruction embeddings can provide comparable results to the current register allocator.
- Used Chromium as a corpus for RL learning
  - Exposed several new compiler/linker bugs
- Currently working on upstreaming in Chromium
  - Precisely quantifying improvements can be difficult
  - Model is adaptable to different code bases

References/Prior work:
Chris Cummins

I have used LLVM in…

research
CGO '22 & '17, ICML '21, MLSys '21, ISSTA '17, PACT '17, …

open source
CompilerGym

worky work

My wish list for LLVM:
1. better **discoverability**
2. better **modularity**
IR2Vec - Program Embeddings

(ACM TACO 2020)

- Program representations for ML
- Language & Machine Independent
  - LLVM IR based embeddings
- Application independent
  - Different compiler optimizations
  - Software engineering applications
- Captures syntax and semantics
- Device mapping & Thread coarsening tasks

Source: https://github.com/IITH-Compilers/IR2Vec
https://compilers.cse.iith.ac.in/projects/ir2vec/

Recent Works

**RL4ReAl**
Reinforcement Learning for Register Allocation
(ArXiv 2022)

**Loop Distribution**
Distribution for better Locality & Parallelization
(LLVM HPC 2022)

**POSET-RL**
Phase ordering for Size and Time optimization
(ISPASS 2022)

**Algorithm Identification**
ML based algorithm identification
(APNET 2022)