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

ML in LLVM at Google

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

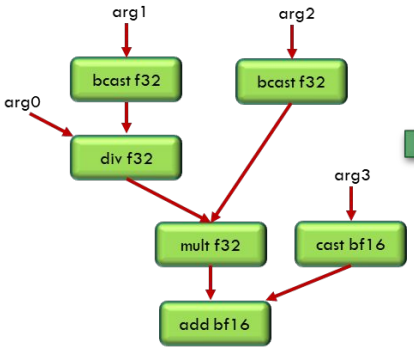
Large Scope Optimizations

- For example currently unapproachable large whole program optimization (LTO)

Dibyendu Das @Intel

ML-driven hardware cost prediction

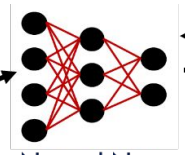
- Observations
  - Applicable to MLIR dialects, LLVM-IR, MIR (Machine IR), asm …
  - Learned cost models shown to be better than llvm-mca, iaca, … ([1])
  - Can be used to predict HW performance characteristics of entire dataflow graphs ([2],[3])
  - Can be used to evaluate optimization plans (like LLVM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




```
func.func @main(%arg0: tensor<640x30522xf32>, %arg1: tensor<640x1xf32>,
                %arg2: tensor<640x1xf32>, %arg3: tensor<640x30522xbf16>)
                → tensor<640x30522xbf16> {
  %0 = xpu.broadcast %arg1 : tensor<640x1xf32> to tensor<640x30522xf32>
  %1 = xpu.div %arg0, %0 : tensor<640x30522xf32>
  %2 = xpu.broadcast %arg2 : tensor<640x1xf32> to tensor<640x30522xf32>
  %3 = xpu.mult %1, %2 : tensor<640x30522xf32>
  %4 = xpu.cast %3 : tensor<640x30522xf32> to tensor<640x30522xbf16>
  %5 = xpu.add %arg3, %4 : tensor<640x30522xbf16>
  return %5 : tensor<640x30522xbf16>
}
```

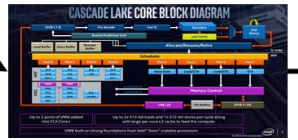
Predict HW Characteristic

MLIR Training DataSet



Neural Net

Back Propagation



HW ground truth/observed characteristics

References:

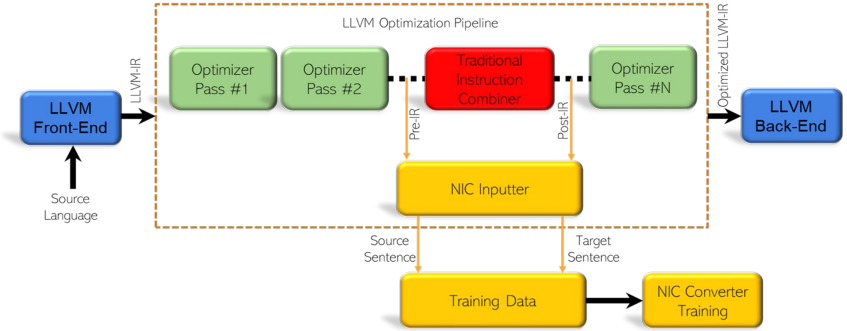
1. *A Learned Performance Model for Tensor Processing Units*, S. J. Kaufman et al. MLSys 2021.
2. *Towards Optimal VPU Compiler Cost Modeling by using Neural Networks to Infer Hardware Performances*, I. Hunter et al. arXiv, May 2022.
3. *Ithemal: Accurate, Portable and Fast Basic Block Throughput Estimation using Deep Neural Networks*, Charith Mendis et al. ICML 2019.
4. *ML-based Hardware Cost Model for High-Level MLIR*, Das et al., llvm-dev Meeting, 2022.

Dibyendu Das @Intel

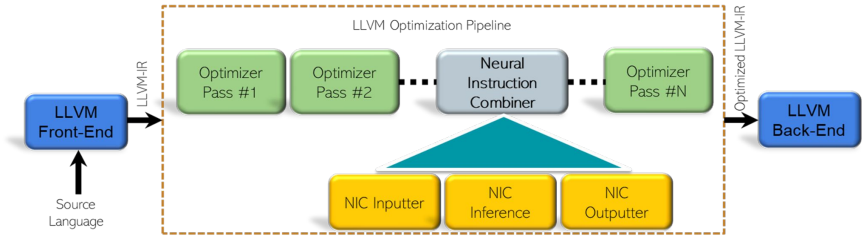
Neural Instruction Combiner

- 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)

NIC Training



NIC Inference



Neural Instruction Combiner, Mannarswamy et al. ICLR DL4C Workshop, 2022 (Also as https://arxiv.org/pdf/2202.12379.pdf)

Amir Ashouri @ Huawei Canada

MLGOPerf: An ML Guided Inliner to Optimize Performance (1/4)

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

Amir Ashouri @ Huawei Canada

MLGOPerf: An ML Guided Inliner to Optimize Performance (2/4)

We extended MLGO to target ***performance*** rather than code-size reduction.

- **MLGOPerf** 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.

Amir Ashouri @ Huawei Canada

MLGOPerf: An ML Guided Inliner to Optimize Performance (3/4)

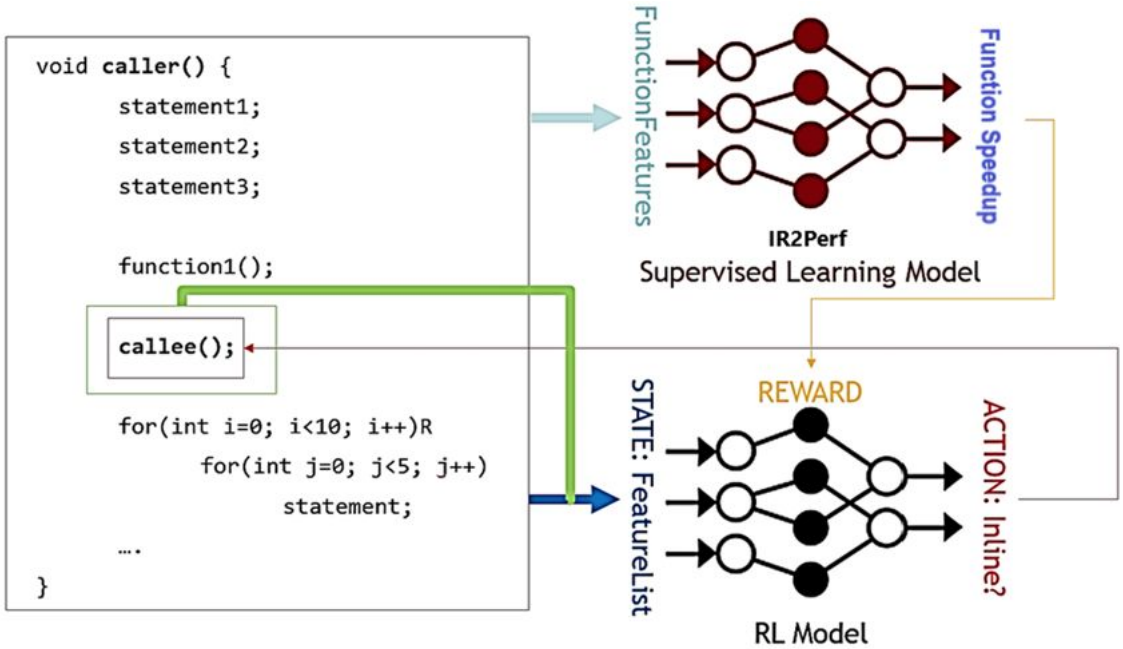


Table 1: MLGOPerf Phases

Phases	IR2Perf	RL Model
IR2Perf Training	Training	-
RL Model Training	Inference	Training
MLGOPerf Deployment	-	Inference

Amir Ashouri @ Huawei Canada

MLGOPerf: An ML Guided Inliner to Optimize Performance (4/4)

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.

(a) SPEC CPU2006 (w/ ref dataset) Performance Improvement

Benchmark	Speedup wrt O3	Measured Variance	Speedup wrt MLGO	Measured Variance	Code size Increase wrt O3	Code size Increase wrt MLGO
401.bzip2	1.052	0.966	1.072	0.594	1.131	1.248
403.gcc	1.022	0.921	1.054	0.004	1.227	1.411
429.mcf	1.009	1.021	1.031	1.242	1.047	1.077
445.gobmk	1.030	0.249	1.044	0.135	1.097	1.104
456.hmmer	0.997	0.040	1.020	0.077	1.227	1.273
458.sjeng	1.003	0.354	1.040	0.031	1.318	1.373
462.libquantum	1.040	1.856	1.051	0.029	1.257	1.428
464.h264ref	1.068	0.620	1.088	0.782	1.389	1.312
471.omnetpp	1.004	1.107	0.999	1.091	1.146	1.198
433.milc	1.021	0.566	0.999	0.486	1.297	1.276
444.namd	0.992	0.530	1.015	0.016	1.002	1.018
453.povray	0.997	0.416	1.035	0.022	1.237	1.418
470.lbm	1.020	0.025	1.004	0.005	1.025	1.031
482.sphinx3	0.993	0.676	0.992	0.070	1.167	1.225
Geomean	**1.018**	**0.434%**	**1.031**	**0.086%**	**1.178**	**1.235**

Cbench	O3		MLGO		MLGOPerf			
	Autotuning Speedup	Tunable Regions	Autotuning Speedup	Tunable Regions	Autotuning Speedup	Tunable Regions		
						Tunable Regions	wrt O3	wrt MLGO
automotive_bitcount	1.027714	20	1.019832	20	1.02781	20	1.000	1.000
automotive_qsort1	1.009412	32	1.008607	32	1.009123	32	1.000	1.000
automotive_susan_c	1.038951	116	1.036704	112	1.037121	188	1.621	1.679
automotive_susan_e	1.031977	116	1.026087	112	1.033349	188	1.621	1.679
automotive_susan_s	1.001988	116	1.065891	112	1.146078	188	1.621	1.679
bzip2d	1.15753	637	1.100431	580	1.165478	747	1.173	1.288
bzip2e	1.032093	637	1.026258	580	1.033333	747	1.173	1.288
consumer_jpeg_c	1.040332	1049	1.017417	891	1.043764	1204	1.148	1.351
consumer_jpeg_d	1.031342	1074	1.014804	885	1.017778	1148	1.069	1.297
consumer_tiff2bw	1.004812	641	1.018229	619	1.017452	903	1.409	1.459
consumer_tiff2rgba	1.047902	633	1.122697	611	1.123338	907	1.433	1.484
consumer_tiffdither	1.012297	640	1.004719	614	1.007853	902	1.409	1.469
consumer_tiffmedian	0.973255	741	1.001938	715	0.988845	1069	1.443	1.495
network_dijkstra	1.078947	13	1.087719	13	1.061404	22	1.692	1.692
network_patricia	1.015152	12	1.015152	12	1.008772	12	1.000	1.000
office_rsynth	0.998958	152	1.001032	147	1.018398	153	1.007	1.041
security_blowfish_d	1.001764	18	1.000441	18	0.998679	18	1.000	1.000
security_blowfish_e	1.001314	18	1.002632	18	1.001314	18	1.000	1.000
security_pgp_d	1.019659	955	1.017919	929	1.036332	1317	1.379	1.418
security_pgp_e	1.039591	955	1.038804	929	1.04023	1317	1.379	1.418
security_rijndael_d	1.040965	22	1.048175	22	1.04694	25	1.136	1.136
security_rijndael_e	1.018811	22	1.02481	22	1.029064	25	1.136	1.136
security_sha	1.009674	10	1.004434	10	1.101781	13	1.300	1.300
telecom_adpcm_c	1.006329	7	1.004211	7	1.002101	7	1.000	1.000
telecom_adpcm_d	1.039636	7	1.039636	7	1.038986	7	1.000	1.000
telecom_CRC32	1.003663	4	1.001217	4	1.003663	5	1.250	1.250
telecom_gsm	1.010018	115	1.009991	112	1.007286	118	1.026	1.054
Geomean	**1.025198**	—	**1.027676**	—	**1.037887**	—	**1.218**	**1.260**

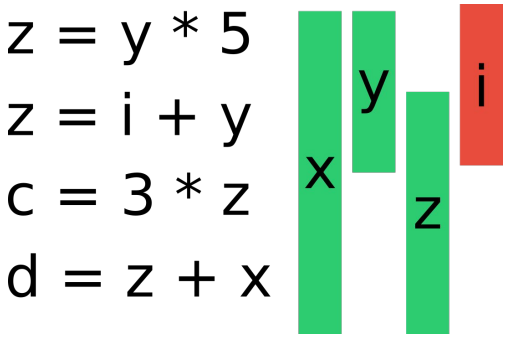
Ondrej - ML for performance modeling

- 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
  - Paper: https://arxiv.org/abs/2210.03894
  - 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

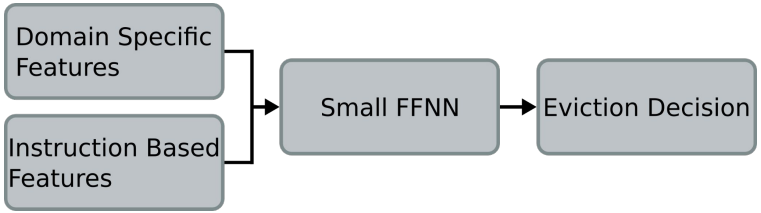
Aiden Grossman @ UC Davis

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



Regalloc example - Three allocated registers on a platform with three registers. When a fourth needs to be allocated, an eviction problem can be created.



MLRegalloc flow - Extraction of features (my work focusing on adding instruction based features), and then passing them to a FFNN which produces a decision.

References/Prior work:

- M. Trofin, Y. Qian, E. Brevdo, Z. Lin, K. Choromanski, and D. Li, “MLGO: a Machine Learning Guided Compiler Optimizations Framework,” arXiv:2101.04808 [cs], Jan. 2021, Accessed: Apr. 07, 2022. [Online]. Available: http://arxiv.org/abs/2101.04808
- D. Das, S. A. Ahmad, and K. Venkataramanan, “Deep Learning-based Hybrid Graph-Coloring Algorithm for Register Allocation,” arXiv:1912.03700 [cs, stat], Dec. 2019, Accessed: Apr. 10, 2022. [Online]. Available: http://arxiv.org/abs/1912.03700
- S. VenkataKeerthy, S. Jain, R. Aggarwal, A. Cohen, and R. Upadrasta, “RL4ReAl: Reinforcement Learning for Register Allocation.” arXiv, Apr. 05, 2022. doi: 10.48550/arXiv.2204.02013.

Chris Cummins

Meta

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**

S. VenkataKeerthy

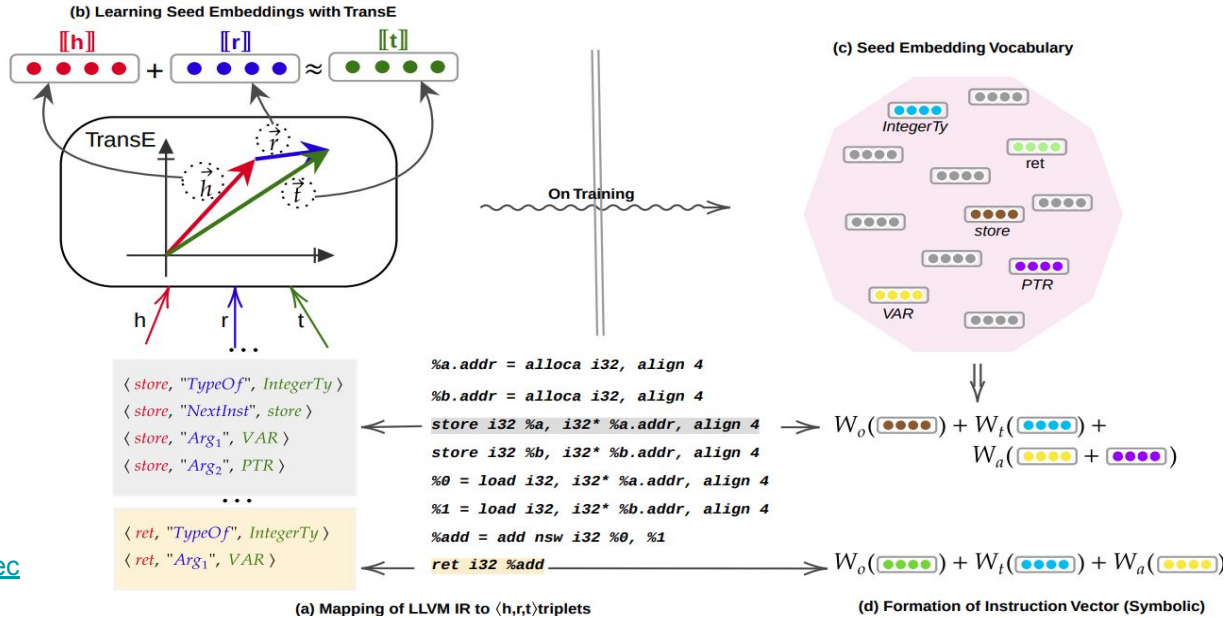
भारतीय सांकेतिक विज्ञान संस्थ हैदराबाद
भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

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)