ML-LLVM-Tools: Towards Seamless Integration of Machine Learning in Compiler Optimizations

Siddharth Jain¹, S. VenkataKeerthy¹, Umesh Kalvakuntla¹,
Albert Cohen², Ramakrishna Upadrasta¹

IIT Hyderabad¹, Google²

European LLVM Developers’ Meeting
10th May 2023
Overview

- ML in Compiler Optimizations
  - Scheme of ML in Compiler Optimizations
  - Proposed Infrastructure

- LLVM-gRPC: gRPC based framework to support Training
  - LLVM-gRPC Usage
  - Use Case: RL4ReAI, IR2Vec

- LLVM-InferenceEngine: ONNX based framework to support Inference
  - Proposed Inference Flow
  - LLVM-InferenceEngine Usage
  - Compile Time Comparison

- Related Works

- Summary
ML in Compiler Optimizations

- Impact of ML in *hard, heuristic-based* compiler optimizations
  - Success of ML in NLP, Image Processing, etc.

- Several ML based compiler optimizations exist
  - From late 90s to date

- ML based optimizations
  - Loop Vectorization, Loop Distribution, Function Inlining, Phase Ordering, Register Allocation, …

- ML in LLVM
  - Inlining decisions (From 11.x), Eviction in Register Allocation (From 14.x)
Scheme of ML in Compiler Optimizations

Input Program

Compiler

Input + Related Info

ML Models

Training / Inference

Processing

Optimization Query

Model Output + Related Info

Opt 1
Materialize Predictions

Opt n

Materialize Predictions
Scheme of ML in Compiler Optimizations

Input Program

Compiler

Optimization Query

Model Output

Opt 1
Materialize Predictions

Opt n

Optimization Query

Model Output

Training / Inference

Processing

Communication
Focusing on Communication …

**Communication**

- **Highly Important**
  - Scalability
  - Compile Time Issues
  - Memory Issues
  - …

- **Determines the practicality**
  - Deployment
  - Usability
  - …
Focusing on Communication ...

Communication

Current Approaches

- No single standard approach
  - Python wrappers
  - Compiler flags
  - ...

- Model written with C++ APIs
  - Tight coupling of APIs
  - ...

Input Program → Compiler → ML Models

Opt 1: Materialize Predictions
Opt n

Training / Inference → Processing
Limitations of Current Approaches

<table>
<thead>
<tr>
<th>Scalability</th>
<th>Integratability</th>
<th>Programmability</th>
<th>Portability</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Python/C++ wrappers</td>
<td>Not all outputs can be communicated via flags</td>
<td>Models written in C++ are not ML developer friendly</td>
<td>Support for diverse ML frameworks</td>
</tr>
<tr>
<td>● 6x – 100x slowdown</td>
<td>Register Allocation, Instruction Scheduling, ...</td>
<td>RLLib, SciPy, ...</td>
<td>TF, PyTorch, JAX, ...</td>
</tr>
<tr>
<td>Phase Ordering, Loop Distribution, ...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Need for **scalable**, **versatile** and **common** framework for ML-based optimizations in LLVM
Proposed Infrastructure

Framework + Architecture independent Infrastructure in LLVM

Training
- ML model development in any generic framework
- ML practitioners can develop solutions in Python

LLVM-gRPC
gRPC based library

Inference
- Within LLVM
- Trained models to be exported and linked with LLVM toolchain

LLVM-InferenceEngine
ONNX based library
LLVM-gRPC

gRPC based framework to support Training
LLVM-gRPC

- Need for a seamless connection between LLVM and Python ML workloads
  - Interprocess communication

- gRPC: Modern open source high performance Remote Procedure Call

- LLVM-gRPC
  - Works as an LLVM library
  - Easy integration – As simple as implementing a few API calls
  - Support for any ML + RL workloads

- Use-case: RL4ReAI [CC’23]

LLVM-gRPC + Passes

Project 1: Proto Files

<projectName>.proto

Project N: Proto Files

Auto Generate Header files

LLVM-gRPC

gRPCUtils

libLLVMgRPC.a

Project-1-gRPCService

RL Module - Project 1 (Python)

LLVM Pass - Project 1

Project-N-gRPCService

LLVM Pass - Project N

ML Module - Project N (Python)
Example Proto File

```protobuf
syntax = "proto3";

package demopass;

// demo pass Service
service demoPass{
// RPC function to send and receive data
// between server and client
rpc getPassInfo(RequestData) returns (PassInfo) {}}

message RequestData {
  string functionName;
}

message PassInfo {
  int32 numInstruction;
}
```

demoPass: Defines the service (a C++ class) which will be auto generated

getPassInfo: Defines the RPC function which has to be overridden

Files generated on compiling proto file:
- demoPass.grpc.pb.cc
- demoPass.grpc.pb.h
- demoPass.pb.cc
- demoPass.pb.h
- demoPass_pb2_grpc.py
- demoPass_pb2.py

Datafields that will be auto generated using gRPC
types coming from demoPass.grpc.pb.h

Inheriting classes

Implementation of gRPC function

Blocking call to start the C++ server

Exiting blocking call
import grpc
import demoPass_pb2_grpc, demoPass_pb2

class demoPassClient(demoPass_pb2_grpc.demoPassServicer):
    def __init__(self):
        self.host='localhost'
        self.server_port=50051
        self.channel=grpc.insecure_channel('{}:{}'.format(self.host,self.server_port))
        self.stub= demoPass_pb2_grpc.demoPassStub(self.channel)
    
    def getRequest(self,requestData):
        request=demoPass_pb2.RequestData(requestData)
        return (self.stub.getPassInfo(request))

if __name__ == '__main__':
    client=demoPassClient()
    functionName=demoPass_pb2.functionName(self.current_fuction_name)
    instruction_count=client.getPassInfo(functionName)

LLVM-gRPC Usage: Python Client
Use Case: RL4ReAl

**RL4ReAl**: Reinforcement Learning for Register Allocation

- RL based register allocator for LLVM compiler
- Models regalloc as graph coloring problem
- Based on MIR2Vec for Machine IR
  - An Extension of IR2Vec
- Uses LLVM-gRPC

S. VenkataKeerthy, Siddharth Jain, Anilava Kundu, Rohit Aggarwal, Albert Cohen, and Ramakrishna Upadrasta. RL4ReAl: Reinforcement Learning for Register Allocation. CC 2023. [https://compilers.cse.iith.ac.in/publications/rl4real/](https://compilers.cse.iith.ac.in/publications/rl4real/)
IR2Vec: LLVM IR Based Scalable Program Embeddings

RL4ReAl: Reinforcement Learning for Register Allocation
## Pros + Cons ...

<table>
<thead>
<tr>
<th>Ease of development</th>
<th>Ease of training</th>
<th>Reusability of code</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Transparent to DL/RL algorithms/policies</td>
<td>● Multiple GPUs distributed training</td>
<td>Write specifications once and use in both Python and C++</td>
</tr>
<tr>
<td>● Supports diverse ML/DL frameworks</td>
<td>● Parallel workers for sample collection</td>
<td></td>
</tr>
</tbody>
</table>

However, LLVM-gRPC is **insufficient** for inference -

- Overhead on compile time
  - Interprocess communication
- Not transparent to user (application developer)
LLVM-InferenceEngine

ONNX based framework to support Inference
LLVM-InferenceEngine

- Framework neutral, interoperable infrastructure for trained model integration
- ONNX: Open Neural Network Exchange
  - Linux Foundation Project (LF AI & Data)
  - Operates in most of the native languages
  - Supported by all major ML/DL frameworks
- Use case study - RL4ReAl [CC’23], POSET-RL [ISPASS’22]

ONNX. Open Neural Network Exchange. 2017, [https://github.com/onnx/onnx](https://github.com/onnx/onnx)

Shalini Jain, Yashas Andaluri, S. VenkataKeerthy and Ramakrishna Upadrasta. POSET-RL: Phase ordering for Optimizing Size and Execution Time using Reinforcement Learning. ISPASS 2022. [https://compilers.cse.iith.ac.in/projects/posetrl/](https://compilers.cse.iith.ac.in/projects/posetrl/)
Model Integration

- **Step 1**: Exporting trained model from native to ONNX format
- **Step 2**: Importing model in compiler with ONNX (C++) runtime environment
Proposed Inference Flow
LLVM-InferenceEngine Usage

```c
#include "environment.h"
#include "inference-engine.h"

struct Hello : public FunctionPass, Environment {

    bool runOnFunction(Function &F) override {
        ...
        InferenceEngine* inference_driver =
            new InferenceEngine(Environment* env);
        inference_driver->getPassInfo(PassData passData,
                                       OptInfo &predictions);
        ...
    }

Inheriting Environment class

InferenceEngine: Creates instance of class InferenceEngine class

getPassInfo: Function to compute predictions from model
```
Compile Time Comparison

**POSET-RL: Phase ordering for Optimizing Size and Execution Time using Reinforcement Learning**

- **Oz**
- **LLVM-InferenceEngine**
- **POSET-RL Original**

**RL4ReAI: Reinforcement Learning for Register Allocation**

- **Greedy**
- **LLVM-InferenceEngine**
- **LLVM-gRPC**

Benchmarks: ibm, leela, mcf, x264, xz, namd, omnetpp, xalancbmk, povray
## Advantages

Features/Advantages of LLVM-InferenceEngine

<table>
<thead>
<tr>
<th>In-process communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>● No RPC calls, IO, etc.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lesser compilation time overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>● No communication overhead</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Versatile + Common infrastructure</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Framework and model agnostic</td>
</tr>
</tbody>
</table>

| Transparent to the user/programmer            |
Other Related Works

- **MLGO: A Machine Learning Framework for Compiler Optimization**
  - Integrated with LLVM
  - Uses TensorFlow APIs and/or raw inter-process communication
  - We would like to explore different scenarios and use cases
    - RL Vs. ML, …; Single Vs. Multiple communication

- **CompilerGym**
  - Provides environments for training RL based compiler optimizations

---


Summary

- Scalable, Versatile and Common framework for ML-based optimizations in LLVM
  - Framework + Architecture independent Infrastructure

- Two components
  - Training - LLVM-gRPC
  - Inference - LLVM-InferenceEngine

- gRPC based training within Python in a framework independent manner

- In-memory ONNX based library for inference in a transparent manner

- Infrastructure is lightweight showing promising trends

- [https://compilers.cse.iith.ac.in/publications/ml-llvm-tools](https://compilers.cse.iith.ac.in/publications/ml-llvm-tools)
Thank you!

https://compilers.cse.iith.ac.in/publications/ml-llvm-tools