CS 526
Advanced Compiler Construction
Machine Learning in Compilers
Anatomy of an Optimization Pass

What are possible objectives?

- Produce Correct Code (semantic equivalence)
- Produce Fast Code
- Produce Energy-efficient Code
- Produce secure code
Anatomy of an Optimization Pass

Objective (f)

Optimization Pass

Input code (I) → Decide what and how to Optimize → Transform Code → Output code (O)
Anatomy of an Optimization Pass

Objective (f)

Optimization Pass

Decision Making

Input code (I) -> Decide what and how to Optimize -> Transform Code

Transformation Machinery

Output code (O)

Goal: f(O) > f(I)
Anatomy of an Optimization Pass

Objective \( f \)

Optimization Pass

- Find Dead Code
- Decide on a set of loop transformations
- Decide where to inline

Generate the Code!

Input code \( I \) ➔ Decide what and how to Optimize ➔ Transform Code ➔ Output code \( O \)
Two types of Optimizations

Objective \( f \)

Input code \((I)\) \(\rightarrow\) Optimization \(\rightarrow\) Output code \((O)\)

Type I

• Steps are always Profitable \( f(O) > f(I) \)
• Mostly independent

Type II

• Steps may not lead to global profitability \( f(O) > f(I) \) ??
• Mostly mutually-exclusive
Let’s try to categorize

• Dead-Code elimination
• Sparse Conditional Constant Propagation
• Global Value Numbering
• Inlining
• Loop Transformations (interchange, tiling etc.)
• Vectorization
• Peephole Optimizations
• Automatic Parallelizations
Anatomy of an Optimization Pass

Objective (f)

Optimization Pass

- Decide what and how to Optimize
- Transform Code

Input code (I) → Output code (O)

Optimization Decision Making

Goal: $f(O) > f(I)$
Optimization Decision Making

Faster and Correct
Output IR
Optimization Decision Making

Transformation Space

Faster and Correct Output IR

Input IR → Opt → Output IR
Optimization Decision Making

semantically equivalent transformations

Faster and Correct Output IR

Transformation Space
Optimization Decision Making

semantically equivalent transformations

Faster and Correct Output IR

Transformation Space
Optimization Decision Making

semantically equivalent transformations

Faster and Correct Output IR

Transformation Space

Cost Model

Input IR → Opt → Output IR
Optimization Decision Making

- Semantically equivalent transformations
- Subspace

Faster and Correct Output IR

Optimization Decision Making Diagram:

- Input IR
- Opt
- Output IR

Transformation Space
Optimization Strategy
Cost Model
Optimization Decision Making

- Transformation Space
- Optimization Strategy
- Cost Model

Input IR → Opt → Output IR

Faster and Correct Output IR

Subspace

Semantically equivalent transformations
Optimization Decision Making

semantically equivalent transformations

Faster and Correct Output IR

Transformation Space
Optimization Strategy
Cost Model
Optimization Decision Making

Faster and Correct
Output IR

Input IR → Opt → Output IR

Transformation Space
Optimization Strategy
Cost Model

Ideal
All Legal Transformations
Optimal
Ground Truth Runtime

Ideal
Ideal
Ideal

semantically equivalent transformations

Subspace
Approximated Subspace
semantically equivalent transformations

Faster and Correct Output IR

Optimization Decision Making

Transformation Space → Optimization Strategy

Input IR → Opt → Output IR

Approximated Limited Subspace
Approximated Hand-crafted Heuristics
Approximated Hand-crafted simple Cost Models

Manually Constructed:
Optimization Decision Making

semantically equivalent transformations

Goal: Automate Construction of these components

Machine Learning is going to help!
Robot Analogy

Task: Move from A to B cheaply

1. Plan
2. Execute

Transformation Space
Optimization Strategy
Cost Model
Robot Analogy

Task: Move from A to B cheaply

1. Plan
2. Execute

Transformation Space → Optimization Strategy → Cost Model
Task: Move from A to B cheaply

1. Plan
2. Execute

Cost: 9
Task: Move from A to B cheaply

1. Plan
2. Execute

Cost: 7
Transformation Spaces

• Loop Transformations

• We will use a combination of horizontal and vertical blurs

L1: for(int x = 0; x < width - 2; x++)
    for(int y = 0; y < height; y++)
        blur_x[x][y] = (input[x][y] + input[x+1][y] + input[x+2][y])/3;

L2: for(int x = 0; x < width; x++)
    for(int y = 0; y < height - 2; y++)
        blur_y[x][y] = (blur_x[x][y] + blur_x[x][y+1] + blur_x[x+2][y])/3;

• Loop Stripmine
• Loop peeling
• Loop fusion
• Loop unrolling
• Vectorization
• Parallelization
• compute_at

• Transformations are dependent on past transformations. Examples?
• Order of transformations?
• Profitability?
Transformation Spaces

• SLP Vectorization

\[
\begin{align*}
S1 &: A1 = \frac{L[5]}{L[2]} \\
S2 &: A2 = \frac{L[6]}{L[3]} \\
S3 &: A3 = \frac{L[7]}{L[4]} \\
S4 &: A4 = L[1] - A2 \\
S5 &: A5 = L[2] - A3 \\
\end{align*}
\]

- Mutually exclusive options
- Profitability

{S1,S2} {S4,S5}
{S2,S3} {S5,S6}
{S1,S3} {S4,S6}
Transformation Spaces

Phase Ordering Problem

\{\text{Pass 1, Pass 2, ..., Pass } N\}

N! Options
Where can ML fit in?

• Can ML design transformation spaces?

• Machine Learning is a good fit for
  • Cost Models
  • Optimization Strategies

• Benefits
  • Adaptive and responsive to workload changes
  • Automated; less human burden in the design process
  • Can achieve state-of-the art results

• Drawbacks
  • May be less interpretable than manually written approaches
Types of Learning

- Supervised Learning (labelled data)
- Unsupervised Learning
- Semi-supervised Learning
- Reinforcement Learning

Image Classification

Object Detection

Machine Translation
Types of Learning

• Supervised Learning
• Unsupervised Learning
• Semi-supervised Learning
• Reinforcement Learning

No labelled data; learn from experience

Choose a “valid” action

State

Iterate

New State

Reward (Win / Loss)
Cost Models

• Analytical Models
  
  e.g., Basic block cost estimation: LLVM-MCA

• Hand-written and cumbersome to maintain

• Usually built with many assumptions baked in
  
  Costs are additive
  
  Costs are linear

• Hardware manuals are the ground truth

~2000 lines

// BMI1 BEXTR/BLS, BMI2 BEXTR

```cpp
defm : HWWriteResPair<WriteBEXTR, [HWPort06,HWPort15], 2, [1,1], 2>;
defm : HWWriteResPair<WriteBLS, [HWPort15], 1>;
defm : HWWriteResPair<WriteBMI, [HWPort15], 1>;

// TODO: Why isn't the HWDivider used?
defm : X86WriteRes<WriteDiv8, [HWPort0,HWPort1,HWPort5,HWPort6], 22, [], 9>;
defm : X86WriteRes<WriteDiv16, [HWPort0,HWPort1,HWPort5,HWPort6,HWPort01,HWPort0156], 90, [7,7,3,3,1,11], 32>;
defm : X86WriteRes<WriteDiv32, [HWPort0,HWPort5,HWPort6,HWPort01,HWPort0156], 90, [7,7,3,3,1,11], 32>;
defm : X86WriteRes<WriteDiv64, [HWPort0,HWPort5,HWPort6,HWPort01,HWPort0156], 90, [7,7,3,3,1,11], 32>;
defm : X86WriteRes<WriteDiv8Ld, [HWPort0,HWPort23,HWDivider], 29, [1,1,10], 2>;
defm : X86WriteRes<WriteDiv16Ld, [HWPort0,HWPort23,HWDivider], 29, [1,1,10], 2>;
defm : X86WriteRes<WriteDiv32Ld, [HWPort0,HWPort23,HWDivider], 29, [1,1,10], 2>;
defm : X86WriteRes<WriteDiv64Ld, [HWPort0,HWPort23,HWDivider], 29, [1,1,10], 2>;
defm : X86WriteRes<WriteIDiv8, [HWPort0,HWPort1,HWPort5,HWPort6], 23, [], 9>;
defm : X86WriteRes<WriteIDiv16, [HWPort0,HWPort1,HWPort5,HWPort6,HWPort06,HWPort0156], 112, [4,2,4,8,14,34], 66>;
defm : X86WriteRes<WriteIDiv32, [HWPort0,HWPort5,HWPort6,HWPort06,HWPort0156], 112, [4,2,4,8,14,34], 66>;
defm : X86WriteRes<WriteIDiv64, [HWPort0,HWPort1,HWPort5,HWPort6,HWPort06,HWPort0156], 112, [4,2,4,8,14,34], 66>;
defm : X86WriteRes<WriteIDiv8Ld, [HWPort0,HWPort23,HWDivider], 29, [1,1,10], 2>;
defm : X86WriteRes<WriteIDiv16Ld, [HWPort0,HWPort23,HWDivider], 29, [1,1,10], 2>;
defm : X86WriteRes<WriteIDiv32Ld, [HWPort0,HWPort23,HWDivider], 29, [1,1,10], 2>;
defm : X86WriteRes<WriteIDiv64Ld, [HWPort0,HWPort23,HWDivider], 29, [1,1,10], 2>;
defm : X86WriteRes<WriteFLD0, [HWPort01], 1, [1], 1>;
defm : X86WriteRes<WriteFLD1, [HWPort01], 1, [2], 2>;
defm : X86WriteRes<WriteFLDC, [HWPort01], 1, [2], 2>;
defm : X86WriteRes<WriteFLoad, [HWPort23], 5, [1], 1>;
defm : X86WriteRes<WriteFLoadX, [HWPort23], 6, [1], 1>;
defm : X86WriteRes<WriteFLoadY, [HWPort23], 7, [1], 1>;
defm : X86WriteRes<WriteFMaskedLoad, [HWPort23,HWPort5], 8, [1,2], 3>;
defm : X86WriteRes<WriteFStore, [HWPort23,HWPort5], 9, [1,2], 3>;
defm : X86WriteRes<WriteFStoreX, [HWPort23,HWPort5], 1, [1], 2>;
defm : X86WriteRes<WriteFStoreY, [HWPort23,HWPort5], 1, [1], 2>;
defm : X86WriteRes<WriteFStoreNT, [HWPort23,HWPort5], 1, [1], 2>;
defm : X86WriteRes<WriteFMaskedStore32, [HWPort0,HWPort4,HWPort32,HWPort15], 5, [1,1,1,1], 4>;
defm : X86WriteRes<WriteFMaskedStore32, [HWPort0,HWPort4,HWPort32,HWPort15], 5, [1,1,1,1], 4>;
```
Data-driven Cost Models

Approach 1: Specify structure and then learn the coefficients

\[
\tilde{y}(t, x) = C_{\text{flop}} \times t_{\text{flop}} + C_{\text{msg}} \times t_{\text{msg}} + C_{\text{vol}} \times t_{\text{vol}}
\]

\[
C_{\text{flop}} = \frac{2n^2 (3m - n)}{2p} + \frac{b_r n^2}{2p_c} + \frac{3b_r n(2m - n)}{2p_r} + \frac{b_r^2 n}{3p_r}
\]

\[
C_{\text{msg}} = 3n \log p_r + \frac{2n}{b_r} \log p_c
\]

\[
C_{\text{vol}} = \left( \frac{n^2}{p_c} + b_r n \right) \log p_r + \left( \frac{mn - n^2/2}{p_r} + \frac{b_r n}{2} \right) \log p_c
\]

\[
t_{\text{flop}}, t_{\text{msg}}, t_{\text{vol}} \text{ are learned}
\]

Liu et. al, “GPTune: multitask learning for autotuning exascale applications”, PPoPP 2021
Data-driven Cost Models

Approach 2: Model parameterized with features

Manually extracted

\[ y = f(s) \]

Code \rightarrow Features (s) \rightarrow Runtime (y)

Ansor: Generating High-Performance Tensor Programs for Deep Learning

Lianmin Zheng 1, Chengian Jia 2, Minmin Sun 2, Zhao Wu 2, Cody Hao Yu 3, Ameer Haj-Ali 1, Yida Wang 3, Jun Yang 3, Danyang Zhao 1, 4, Koushik Sen 1, Joseph E. Gonzalez 1, Ion Stoica 1

Learning to Optimize Halide with Tree Search and Random Programs

ANDREW ADAHS, Facebook AI Research
KARIMA MA, UC Berkeley
LUKE ANDERSON, MIT CSAIL
RIYADH BAGHDADE, MIT CSAIL
Tzu-Hsiao Li, MIT CSAIL
MICHAEL CHARB, Adobe
BENOIT STEINER, Facebook AI Research
STEVEN JOHNSON, Google
RAYMON FATAHALI, Stanford University
FRED DU RAND, MIT CSAIL
JONATHAN IRAGAN-KELLEY, UC Berkeley
Approach 3: black box models that are completely learned

\[ y = f(\text{emb}(C)) \]

Ithemal: Accurate, Portable and Fast Basic Block Throughput Estimation using Deep Neural Networks

~20 -> 8 MAPE

~30 -> 4.5 MAPE

**A LEARNED PERFORMANCE MODEL FOR TENSOR PROCESSING UNITS**

**A DEEP LEARNING BASED COST MODEL FOR AUTOMATIC CODE OPTIMIZATION**
Basic Block Throughput Estimation

Mendis et. al “Ithemal: Accurate, Portable and Fast Basic Block Throughput Estimation using Deep Neural Networks” [ICML'19]

Use data to learn a model

44 cycles
Basic Block Throughput Estimation

Prediction Layer

Weights are not shared

Instruction Layer

Token Layer

Token Embedding Lookup Table

Canonicalization

mov ecx, 0x02

add ebx, ecx
Learned TPU Cost Model
Program Embeddings

• In NLP, they use continuous representations of words that can be fed into a NN. These are known as word embeddings.

• They pre-train these embeddings (e.g., word2vec, GloVe embeddings)

• Similarly, programs can be embedded in continuous space.

• Challenges
  • Programs have strict semantics.
  • Programs have graph structure.

• Some efforts
  • Inst2vec
  • Blended Semantic Embeddings
  • PrograML
  • CuBERT
  • Contextual Flow Graphs and so on.
Optimization Strategies

• Two main ML options

  • **Search**
    • Genetic Algorithms
    • Beam Search
    • Monte Carlo Tree Search

  • **Learned**
    • Supervised Learning
    • Sequential Decision Making
    • Bayesian Optimization
Genetic Algorithms

- Find the set of genes (parameters settings) that are the fittest (optimizes an objective) using genetic evolution.
Genetic Algorithms

- Find the set of genes (parameters settings) that are the fittest (optimizes an objective) using genetic evolution.

Repeat until budget exhausted or population meets convergence criteria
Evolution

Mutations

Randomly mutate parts of the gene

Crossovers

Mix of two Genes
Evolution

Population i → Evolutions → Compute Fitness → Keep the Fittest → Population i+1
Auto-tuning

• Generally, tuning parameters of a fixed set of transformations.
  • e.g. deciding on the unroll factor, tiling factor, vectorization factor
  • Also extends to deciding the transformations themselves
    • e.g. Deciding when to unroll or not
  • In either case, auto-tuning searches for the best performing code transformations.
Auto-tuning

Population i → Evolutions → Change configuration → Cost Model / Runtime → Compute Fitness → Keep the Fittest → New Program Configuration → Population i+1
Auto-tuning use cases

Mitigating the Compiler Optimization Phase-Ordering Problem using Machine Learning
Sameer Kulkarni John Cavazos
University of Delaware
{skulkarn,cavazos}@cis.udel.edu

Meta Optimization: Improving Compiler Heuristics with Machine Learning
Mark Stephenson and Saman Amarasinghe
Massachusetts Institute of Technology
Laboratory for Computer Science
Cambridge, MA 02139
{mstephen, saman}@cag.lcs.mit.edu

Martin Martin and Una-May O’Reilly
Massachusetts Institute of Technology
Artificial Intelligence Laboratory
Cambridge, MA 02139
{mcm, unamay}@ai.mit.edu
Auto-tuning using OpenTuner

- A general framework for program auto-tuning
- Provides an interface
  - To specify parameter spaces
  - To specify search strategies
  - To specify multi-objective tuning
- Provides a meta-optimization heuristic
  - Multi-arm bandit technique

Auto-tuning DSLs

Andrew et. al “Learning to Optimize Halide with Tree Search and Random Programs” SIGGRAPH 2019
Optimization Strategies

- Two main ML options

- **Search**
  - Genetic Algorithms
  - Beam Search
  - Monte Carlo Tree Search

- **Learned**
  - Supervised Learning
  - **Sequential Decision Making**
  - Bayesian Optimization
Sequential Decision Making

Markov Decision Process (MDP)

Choose a “valid” action

State → Iterate → New State

Reward (Win / Loss)
Sequential Decision Making

Building a Basic Block Instruction Scheduler with Reinforcement Learning and Rollouts

AMIT MCGOVERN
eli@cs.umass.edu
ANDREW G. BARTO
barto@cs.umass.edu
Department of Computer Science, University of Massachusetts, Amherst, Amherst, MA 01003, USA

Instruction Scheduling

Optimization decisions trigger state transitions

Compiler Auto-Vectorization with Imitation Learning

Charith Mendis
MIT CSAIL
charithm@csail.mit.edu
Cambridge Yang
MIT CSAIL
canyang@csail.mit.edu
Yuewen Pu
MIT CSAIL
yuewenp@csail.mit.edu
Suman Amarasinghe
MIT CSAIL
suman@csail.mit.edu
Michael Carbin
MIT CSAIL
mcarbin@csail.mit.edu

Auto-vectorization
Vectorization as a Markov Decision Process

\[
\begin{align*}
\end{align*}
\]

Choose a “valid” action

\[
\begin{align*}
\{a[3],a[4]\} & \quad \rightarrow \\
\{a[1],[a[3]],a[4]\},a[4] & \\
\end{align*}
\]

Iterate

\[
\begin{align*}
\end{align*}
\]

Reward (Speed of execution)
What we do to solve this MDP

State


New State


Choose a “valid” action

Iterate

Use Imitation Learning
Learnt Vectorization - Vemal

Mendis et. al “Compiler Auto-Vectorization with Imitation Learning” [NeurIPS’19]

Collect Demonstrations

\[
\begin{align*}
\end{align*}
\]

Oracle

goSLP

\[
\begin{align*}
\end{align*}
\]

State-Action Pairs

\[
\begin{align*}
    \quad \{a[1], a[2]\}
\end{align*}
\]
Learnt Vectorization - Vemal

Mendis et. al “Compiler Auto-Vectorization with Imitation Learning” [NeurIPS’19]

Collect Demonstrations

Oracle
goSLP

\{a[3], a[4]\}

\begin{align*}
\end{align*}

State-Action Pairs

\begin{align*}
\end{align*}
Learnt Vectorization - Vemal

Mendis et. al “Compiler Auto-Vectorization with Imitation Learning” [NeurIPS’19]

Training

State-Action pairs (replay buffer)

Policy Network

Back Propagation

Action

1-hot vector
Learnt Vectorization - Vemal

Mendis et. al “Compiler Auto-Vectorization with Imitation Learning” [NeurIPS’19]

Training

Augmented Replay Buffer

Back Propagation

Policy Network

Action

1-hot vector

DAGGER to augment dataset at each epoch

Ross et. al [AISTATS’11]
Thank You!

• We are almost there! Presentations, report and the final to go.

• I enjoyed teaching the class. I hope you all know more about compilers before we started the semester.

• Please give class feedback at,

  https://ices.citl.illinois.edu