CS 526
Advanced Compiler Construction

https://charithm.web.illinois.edu/cs526/sp2022/
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

Input code (I) → Decide what and how to Optimize → Transform Code → Output code (O)

Optimization Decision Making

Transformation Machinery

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

Input code (I) ➔ Decide what and how to Optimize ➔ Transform Code ➔ Output code (O)

Generate the Code!
Two types of Optimizations

Objective (f)

Input code (I) → Optimization → 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

Input code (I) → Decide what and how to Optimize → Transform Code → Output code (O)

Optimization Decision Making

Goal: \( f(O) > f(I) \)

Transformation Machinery
Faster and Correct Output IR
Optimization Decision Making
Optimization Decision Making

semantically equivalent transformations

Faster and Correct Output IR

Transformation Space

Opt

Input IR

Output IR
Optimization Decision Making

semantically equivalent transformations

Faster and Correct Output IR

Transformation Space

Opt

Input IR → Opt → Output IR
Optimization Decision Making

semantically equivalent transformations

Faster and Correct Output IR

Input IR \rightarrow \text{Opt} \rightarrow \text{Output IR}

Transformation Space

Cost Model
Optimization Decision Making

semantically equivalent transformations

Faster and Correct Output IR

Transformation Space → Optimization Strategy → Cost Model

Input IR → Opt → Output IR
Optimization Decision Making

- Subspace
- Semantically equivalent transformations

Faster and Correct Output IR

- Input IR
- Opt
- Output IR

- Transformation Space
- Optimization Strategy
- Cost Model
Optimization Decision Making

- Subspace
- Faster and Correct Output IR
- Transformation Space
- Optimization Strategy
- Cost Model
Optimization Decision Making

Semantically equivalent transformations

Faster and Correct Output IR

Input IR → Opt → Output IR

Transformation Space → Optimization Strategy → Cost Model

Ideal

All Legal Transformations

Ideal

Optimal

Ideal

Ground Truth Runtime
Optimization Decision Making

Approximated Subspace

semantically equivalent transformations

Manually Constructed:
- Limited Subspace
- Hand-crafted Heuristics
- Hand-crafted simple Cost Models

Faster and Correct Output IR

Transformation Space
Optimization Strategy
Cost Model

Input IR -> Opt -> Output IR
Optimization Decision Making

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

![Diagram showing transformation space, optimization strategy, and cost model](image-url)
Task: Move from A to B cheaply

1. Plan
2. Execute

Robot Analogy

Transformation Space

Optimization Strategy

Cost Model
Robot Analogy

Task: Move from A to B cheaply

1. Plan
2. Execute

Cost: 9
Robot Analogy

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

\[
\begin{align*}
L_1: & \text{for(int } x = 0; x < \text{width - 2; } x++) \\
& \text{ for(int } y = 0; y < \text{height; } y++) \\
& \quad \text{blur}_x[x][y] = (\text{input}[x][y] + \text{input}[x+1][y] + \text{input}[x+2][y])/3;
\end{align*}
\]

\[
\begin{align*}
L_2: & \text{for(int } x = 0; x < \text{width; } x++) \\
& \text{ for(int } y = 0; y < \text{height - 2; } y++) \\
& \quad \text{blur}_y[x][y] = (\text{blur}_x[x][y] + \text{blur}_x[x][y+1] + \text{blur}_x[x+2][y])/3;
\end{align*}
\]

• 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

\[
\begin{align*}
S4 : A4 &= L[1] - A2 \\
\end{align*}
\]

• Mutually exclusive options

• Profitability

\{S1, S2\} \quad \{S4, S5\}
\{S2, S3\} \quad \{S5, S6\}
\{S1, S3\} \quad \{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
Types of Learning

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

No labelled data; learn from experience

State

Choose a “valid” action

Iterate

New State

Reward (Win / Loss)
Announcements

• **Project 2**
  • Presentations on 04/28, upload the presentation before 10am on 04/28
    • Follow piazza post for instructions
  • Report and code due on 05/01
    • Please submit a GitHub / box link for the code and the test cases
    • Make sure there are clear building, running and testing instructions
    • If you need more time, I can grant up to a week extension
  • Submission: use the same form you used for progress reports

• **Final**
  • Whole day on 05/03 (instructions to come in a piazza post)

• Please give feedback for the class.

https://ices.citl.illinois.edu
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

```cpp
-2000 lines

// BMI1 BEXTR/BLS, BMI2 BEX

| defm : HWWriteResPair<WriteBEXTR, [HWPort06,HWPort15], 2, [1,1], 2>; |
| defm : HWWriteResPair<WriteBLS, [HWPort15], 1>; |
| defm : HWWriteResPair<WriteBZHI, [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], 98, [7,7,3,1,111], 32>; |
| defm : X86WriteRes<WriteDiv32, [HWPort0,HWPort5,HWPort6,HWPort01,HWPort0156], 98, [7,7,3,1,111], 32>; |
| defm : X86WriteRes<WriteDiv64, [HWPort0,HWPort1,HWPort5,HWPort6,HWPort01,HWPort0156], 98, [7,7,3,1,111], 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, [4,2,4,14,34], 66>; |
| defm : X86WriteRes<WriteIDiv32, [HWPort0,HWPort5,HWPort6,HWPort06,HWPort0156], 112, [4,2,4,14,34], 66>; |
| defm : X86WriteRes<WriteIDiv64, [HWPort0,HWPort5,HWPort6,HWPort06,HWPort0156], 112, [4,2,4,14,34], 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<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>; |

// Scalar and vector floating point.
| 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,HWPort4], 1, [1,1], 2>; |
| defm : X86WriteRes<WriteFStoreY, [HWPort23,HWPort4], 1, [1,1], 2>; |
| defm : X86WriteRes<WriteFStoreNT, [HWPort23,HWPort4], 1, [1,1], 2>; |
| defm : X86WriteRes<WriteFStoreNTX, [HWPort23,HWPort4], 1, [1,1], 2>; |
| defm : X86WriteRes<WriteFStoreNTY, [HWPort23,HWPort4], 1, [1,1], 2>; |
| defm : X86WriteRes<WriteFMaskedStore32, [HWPort0,HWPort4,HWPort237,HWPort15], 5, [1,1,1,1], 4>; |
| defm : X86WriteRes<WriteFMaskedStore32Y, [HWPort0,HWPort4,HWPort237,HWPort15], 5, [1,1,1,1], 4>; |
```
Data-driven Cost Models

Approach 1: Specify structure and then learn the coefficients

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

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

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

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

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

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 → Features (s) → Runtime (y)

Ansor: Generating High-Performance Tensor Programs for Deep Learning

Lianmin Zheng 1, Chengfan Jia 2, Minmin Sun 2, Zhao Wu 2, Cody Hao Yu 3, Ameer Haj-Ali 1, Yida Wang 2, Jun Yang 2, Danyang Zhuo 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
BHUVAN BHATIA, MIT CSAIL
TZI-JIANG LIN, MIT CSAIL
MICHAEL CHARBII, Intel
BENOIT STEINER, Facebook AI Research
STEVEN JOHNSON, Google
KAYVON BAHARLI, Stanford University
PRES DU RAND, MIT CSAIL
JONATHAN RAGAN-KELLEY, UC Berkeley
Data-driven Cost Models

Approach 3: black box models that are completely learned

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

Machine Code

Computational Graph (XLA IR)

Source Code

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

A Learned Performance Model for Tensor Processing Units

A Deep Learning Based Cost Model for Automatic Code Optimization

Riyadh Baghdadi 1, Mansoura Ebrahimi 1, Mohamed Elshamy Leghriet 3, Kamel Abdoun 3, Taha Arbaoui 4, Karima Benadila 5, Suzan Amerasinghe 6

\[ \sim 20 \rightarrow 8 \text{ MAPE} \]

\[ \sim 30 \rightarrow 4.5 \text{ MAPE} \]
Mendis et. al “Ithemal: Accurate, Portable and Fast Basic Block Throughput Estimation using Deep Neural Networks” [ICML’19]

Use data to learn a model
Basic Block Throughput Estimation

Prediction Layer

Instruction Layer

Token Layer

Weights are not shared
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

Program Configuration

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

Choose a “valid” action

State | Iterate | New State

Markov Decision Process (MDP)

Reward (Win / Loss)
Sequential Decision Making

Building a Basic Block Instruction Scheduler with Reinforcement Learning and Rollouts

AMIT MCGOVERN
ELIOT MOSS
ANDREW G. BARTO
Department of Computer Science, University of Massachusetts, Amherst, Amherst, MA 01003, USA

Instruction Scheduling

(c) Dependence Dag of Instructions

(d) Partial Schedule

Compiler Auto-Vectorization with Imitation Learning

Charith Mendis  MIT CSAIL  charithm@csail.mit.edu
Cambridge Yang  MIT CSAIL  canyang@csail.mit.edu
Yeyun Pu        MIT CSAIL  yeyunp@csail.mit.edu
Samuel Amaratunga MIT CSAIL  saman@csail.mit.edu
Michael Carbin  MIT CSAIL  mcarbin@csail.mit.edu

Auto-vectorization

Optimization decisions trigger state transitions
Vectorization as a Markov Decision Process

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

Choose a “valid” action

```
\{a[3], a[2]\}
```

Iterate

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

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

Choose a “valid” action

Iterate

Use Imitation Learning

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

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

Collect Demonstrations

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

Oracle

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

Learnt Vectorization - Vemal

State-Action Pairs

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

\{a[1], a[2]\}
Learnt Vectorization - Vemal

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

Collect Demonstrations

Oracle
goSLP

{a[3], a[4]}

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

0.1 0.1 0.6 0.1 0.1
Learnt Vectorization - Vemal

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

Training

Augmented Replay Buffer

Back Propagation

Policy Network

DAGGER to augment dataset at each epoch

Ross et. al [AISTATS’11]

Action

1-hot vector

0.1 0.1 0.6 0.1 0.1
We will be covering most of these topics in detail in the ML for Comp and Arch course.

Please check out the course syllabus for Fall 2021. Most content will remain the same for Fall 2022.
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