Use of Code Structural Features for Machine Learning to Predict Effective Optimizations

International Workshop on Automatic Performance Tuning
May 25, 2018@Vancouver, Canada

Yuki Kawarabatake, Mulya Agung, Kazuhiko Komatsu, Ryusuke Egawa, and Hiroyuki TAKIZAWA
(Tohoku University)
Outline

- Introduction
- Use of code structures for machine learning
- Preliminary evaluation results
- Conclusion and future work
Background

What is the dominant factor of actual simulation performance? Peak flop/s rate?

Actual performance is often determined by programmers’ time and/or efforts invested in performance engineering.
Performance-aware Programming

Do **something** to improve performance

Performance Measurement
- Execution & Profiling
- Finding bottleneck
- Estimating expected performance

Performance Optimization

Performance Analysis

Performance Modelling

Motivation

Can we write an explicit algorithm to predict effective optimizations for a given code? → Yes and No.

- Yes. Compilers automatically apply various optimizations to a given code.
- No. Expert programmers still have to select various options in practice.
  - Algorithms, data structures, loop transformations, ...

Since there is no clear algorithm of the prediction, human experts have to predict effective optimizations on a case-by-case basis.

The final goal is to automate the prediction to reduce burdens of performance optimization on programmers.
This paper

Effective Compiler Option Prediction

**Compiler option selection = Performance optimization selection**

- **O2 option flag** = almost all supported optimizations that do not involve a space-speed tradeoff.
- **O3 option flag** = more optimizations are turned on.

Higher is better
Outline

- Introduction
- **Use of code structures for machine learning**
- Preliminary evaluation results
- Conclusion and future work
Penalty Weighted Geometric Accuracy

- **Definition of Average Prediction Accuracy**
  - Misprediction of compiler option flags *sometimes* leads to drastic performance degradation, but *other times not.*
    - Need to consider not only the misprediction count but also the **performance penalty** of each misprediction.
  - Arithmetic mean of ratios such as normalized values can be misleading.
    - Use **Geometric Mean**, instead.

- **Penalty Weighted Geometric Accuracy (PWGA)**
  - Prediction accuracy with considering the performance penalty of misprediction.

\[
PWGA = \sqrt[n]{\frac{\prod_{i=1}^{N} T_{\text{best},i}}{T_{\text{pred},i}}}
\]
Related Work

- Prediction by machine learning with performance profiling information (Cavazos+ 2007)

Better than random selection. But far from perfect.

Reproduction of their results on our environment
How can we improve it?

Why not perfect?

- The number of training data is too small.
  - Only 263 loops are used for our experiment.
- The information about the code itself is not available.
  - Human experts also see the code to find the performance bottleneck to consider effective optimizations.

What happens if the code information is available for machine learning to predict effective compiler options?
Not so easy...

■ How can we express code structures as a vector?
  • It’s not easy to appropriately quantify the features

The accuracy is degraded by using the additional features.

Manually-defined code features
Discovering Useful Features

Success in image recognition/classification

- Conventional approaches
  - Features of images are predefined.
  - **Feature selection** is the key to success.
- Deep learning (LeCun+ 2015)
  - **Feature learning/representation learning**
    - Machine learning can find not only underlying classification rules but also useful features for the classification.
    - Big data with high computing power are the key to success.
Use of code structural features

Profiling

Manual feature selection

Target Application Code

Representation learning

Abstract Syntax Tree

Tree-Based Convolutional Neural Network

Code Structural Features

This must be represented as a fixed-size vector.

Dynamic Features

Predefined Loop Features

Code Features

Multilayer Perceptron

Optimal Compiler Option Prediction
Tree-Based Convolutional Neural Network

The code structure is translated into a vector.

100-dimensional Vector

Feature Vector

Dynamic Pooling

Tree-based Convolution

Vector Representation

Word2vec

AST

Decl

TypeDecl

IdentifierType

ID

BinaryOp

Constant

0 0 0

0 0 0

0 0 0

0 0 0

0 0 0
Outline

- Introduction
- Use of code structures for machine learning
- Preliminary evaluation results
- Conclusion and future work
Experimental Setup

<table>
<thead>
<tr>
<th></th>
<th>Intel Xeon E5-2695v2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peak Performance [Gflop/s]</td>
<td>230.4/socket, 19.2/core</td>
</tr>
<tr>
<td>Number of cores</td>
<td>12</td>
</tr>
<tr>
<td>Vector length/ SIMD width (double)</td>
<td>4</td>
</tr>
<tr>
<td>Cache size</td>
<td>L2:256KB/core, L3:30MB/socket</td>
</tr>
<tr>
<td>Memory bandwidth [GB/s]</td>
<td>59.7</td>
</tr>
<tr>
<td>Compiler</td>
<td>GNU Fortran Compiler 4.4.7</td>
</tr>
<tr>
<td>Performance Counter</td>
<td>PAPI 5.3.2</td>
</tr>
</tbody>
</table>

Cross-validation is performed for the evaluation. Some of data are used for training and the others are for testing. Randomly selected
Prediction Using Only Code Structural Features

Representation learning can achieve almost the same performance as the existing profiling-based approach.

Representation learning can extract useful features from code structures. → A higher performance is achieved for some cases.
Using All the Features

Prediction accuracy improved

Prediction accuracy degraded

International Workshop on Automatic Performance Tuning 2018
Outline

- Introduction
- Use of code structures for machine learning
- Preliminary evaluation results
- Conclusion and future work
Conclusions

■ This work is still ongoing.
  - Code structural features are used to predict effective performance optimizations
    - TBCNN is used to convert a code structure to a fixed-size vector so that it can discover useful features from training data.
  - The prediction accuracy improves if the number of training data is not too small, but degrades if it is too small.
    - It is unclear why use of the code structural features differently affects the accuracy, depending on the number of training data.

■ Conclusion
  - Use of code structural features discovered by representation learning is promising to improve the accuracy if an enough number of data are available.
    → Artificial training data generation (future work)
Acknowledgments

This work was partially supported by

- JST CREST Xevolver Project
- DFG SPPEXA ExaFSA project
- Grant-in-Aid for Scientific Research(B) 16H02822
- Grant-in-Aid for Challenging Exploratory Research 15K12033

- The performance evaluation results were obtained using supercomputing resources of the Cyberscience Center, Tohoku University.