A Cost Model for Compilers Based on Transfer Learning

Yuta Sasaki*, Keichi Takahashi‡*, Yoichi Shimomura‡, Hiroyuki Takizawa‡*

* Graduate School of Information Sciences, Tohoku University, ysasaki@hpc.is.tohoku.ac.jp
‡ Cyberscience Center, Tohoku University, {keichi,shimomura32,takizawa}@tohoku.ac.jp
Yuta Sasaki is presently with NTT DATA Corporation.

The Seventeenth International Workshop on Automatic Performance Tuning (iWAPT2022)
Outline

• Introduction
• Proposed Method
• Dataset and Evaluation Metrics
• Evaluation and Results
• Discussions and Conclusions
Introduction

- **HPC system architectures are getting more complicated**
  - Automatic optimization by compilers, *compiler optimization*, is becoming more crucial

- **Compilers perform code optimizations for high-performance**
  - Various optimization passes are implemented and can be applied automatically
  - Sometimes applying these passes might even decrease the performance depending on the target system and application

- **Compiler needs to select which passes to apply to maximize the performance**
  - In what order to apply them? / What parameters to use?
  - Need to evaluate the candidates of optimization passes
    - Execution a huge number of candidates results in long compilation time
Cost Model for Compiler Optimization

Cost models are used to predict the performance improvement without running the program

- Machine learning is often used to empirically construct cost models in a data-driven way
- Analytical modeling of a modern complex computing system is infeasible

Cost model based on machine learning

- Built from performance data, which are collected by running a huge number of programs on the target system
  - Time-consuming
- Many cost models based on machine learning is specialized for training system
  - Users need to collect performance data in their systems to build their own models
Overview of the Proposed Method

- Building a cost model of a target system from as few data as possible
  - Adopts transfer learning to build a cost model of the target system from a pre-trained cost model, *a source model*, of another system
  - Can build multiple models from a single source model with fewer data
The Cost of Building a Training Dataset

- A data-driven approach to build a cost model needs a large dataset
- The cost of building a training dataset is strongly correlated to the number of times to run programs on the target system

  - A program is defined by its source code and a sequence of optimization passes
  - Each sample in training data is a pair of a program and its performance on the target system
  - It is potentially possible to improve the prediction accuracy by carefully selecting training data with the same number of training data
Dataset for the Evaluation

■ TenSet [1]: A large-scale dataset to train the cost model for TVM
  • Consists of trained deep neural networks and sequences of optimization passes
    • Neural networks are divided into subgraphs called tasks
    • TVM compiler optimizes the whole network by applying a sequence of optimization passes called a schedule to each task
  • Annotated with performance labels on 4 CPU and 2 GPU systems
    • 4 CPU systems: Xeon E5-2673, Xeon Platinum 8272, AMD EPYC 7452 and ARM Graviton2
    • 2 GPU systems: NVIDIA Tesla T4 and NVIDIA Tesla K80

■ We use DNNs in two ways
  • To build a cost model
  • A program to be optimized

21 deep neural networks

Training Data (18 models)

Testing Data (3 models)

2,308 tasks

Up to 4,000 schedules / task

Execution time (2~8 measurements)

51 million measurement data
Evaluation Metrics

Learning efficiency of transfer learning

- # of programs required to reach the baseline performance by transfer learning
  - # of programs used to train the baseline model
  - Not include # of training data used to train the source model in the numerator

Prediction accuracy (Pairwise Comparison Accuracy)

- Predicts performance of N programs
- M: # of pairs of which the predicted and measured performance values match

- The cost model with PCA close to 1 will be able to select a better optimization pass

\[
PCA = \frac{M}{\binom{N}{2}} = \frac{M}{N \cdot (N - 1)/2}
\]

- Predicted execution time
- Measured execution time
  - PCA = 1
  - PCA = 5/6 = 0.83
  - PCA = 0
Overview of the Evaluation

To achieve higher prediction accuracy with less training data
  • Source model selection
  • Transfer learning technique
  • Training data selection

Using the model trained on a small number of data, optimize the program and evaluate its performance
Evaluation Setup

**Eval.1: Build baseline models**
- Train six cost models using all training data on the six systems.
- Test on the data obtained from the **same/different** systems from training
  - Baseline models: Targets of Accuracy
  - Source models: Initial state of TL

**Eval.2: Transfer learning**
- From partial training data of the target system, re-train other five models in Eval.1
- **Target systems**
  - CPU system, Xeon E5-2673
  - GPU system, Tesla T4
Results of Eval.1

- **Baseline/Source model**
  - Testing data are obtained from the same/different systems from training data
    - Cost models trained for each system
      - Highest PCA for each system
    - Cost models trained for other CPU systems
      - Lower PCA, differences even between models
    - Cost models of different architecture systems
      - PCA is around 0.5, which is equivalent to random

- **CPU performance prediction uses different features than GPU performance prediction**
  - Different architectures could need different program features for prediction
Results of Eval.2

■ Target system: Xeon E5-2673
  • Normal learning
  • Transfer learning
  • Baseline model
  • Source models

■ Single task (4,000 programs)
  • Transfer learning shows a higher PCA

■ Selecting the most accurate source model can finally achieve high prediction accuracy with less training data in TL
Transfer Learning Technique

Use 5-layer perceptron

**Fine-tuning** (Eval.2)
- Updates all layers in the same way as in normal model learning where network weights are initialized randomly
- The same accuracy can be expected if enough training data are available

**Feature Extractor** (Eval.3)
- Retains the weights in some layers of the source model and update only other layers during training
- Reduces the degree of freedom of the network, faster convergence

A fine-tuned model for system B
A pre-trained model for system A
A cost model for system B using the feature extractor of A
Results of Eval.3

- Fixed four layers as feature extractor and updated only single output layer
  - Source: EPYC 7452
  - Target: Xeon E5-2673

- The accuracy saturates at a lower value
  - Because the number of weights tuned for the target system is smaller

- Fine tuning is better when a sufficient amount of data are available
  - But the feature extractor approach could be one option when only few data are available
Training Data Selection (Eval.4)

Compared three methods for reducing the training data in normal learning

• (1) the number of schedules is fixed to 4,000 and the number of tasks is reduced by half
• (2) the number of tasks is fixed to 1,600 and the number of schedules is reduced by half
• (3) the number of programs and tasks is reduced by half respectively

A higher priority to getting more tasks achieve higher accuracy even with the same amount of performance data
Training Data Selection (Eval.4)

- Compare three methods for reducing the training data in **transfer learning**
  - Source: EPYC 7452
  - Target: Xeon E5-2673

- Unlike normal learning, the decrease in accuracy is small when reducing data
  - Source model is trained with all the available data of EPYC
  - Learns program features useful for prediction very well

- A higher priority to getting more tasks achieve higher accuracy
Learning Efficiency

- Construct a cost model with the same prediction accuracy from a smaller number of training data
  - Source model with the highest PCA
  - Fine tuning is performed to update all layers
  - Fix the maximum number of tasks to 1,600, gradually increase the number of schedules

- TL requires only about 1200 to achieve the same accuracy as normal learning
  - which is 30% of the baseline model
Learning Efficiency

- **TL model of Tesla T4**
  - Achieves the same prediction accuracy using 1,000 schedules which is 25% of the baseline model

- The proposed method is effective in reducing not only the amount of data but also training time
  - TL can reduce the execution time by 78% to achieve the same accuracy
Program Performance

- Optimize a pre-trained inference model, ResNet-50 for Xeon E5-2673
  - Baseline model
  - Transfer learning trained with 30% data
  - Source Model

- The transfer learning model achieves the same reduction in inference time as the baseline model
  - The optimized inference models achieved a 16% reduction in execution time

- The model built from a small amount of performance data achieves program speedup as the model trained with a large amount of data
Conclusions

We proposed a data-driven method to build cost models for compiler optimization

- Focus on reducing the performance data of a target system, by using transfer learning
- Proposed method can significantly reduce the training data
- TL can make it more affordable to build a cost model for compiler optimization in a data-driven way

Future work

- Use this approach also to other applications while further improving the accuracy with less training data
- Explore a way to provide even higher performance in a variety of combinations of systems and applications.
Acknowledgments

This work is partially supported by

- MEXT Next Generation High-Performance Computing Infrastructures and Applications R&D Program “R&D of A Quantum-Annealing-Assisted Next Generation HPC Infrastructure and its Applications”
- JSPS KAKENHI Grant Number JP20H00593