Automated selection of build configuration based on machine learning

Reo Furuhata\textsuperscript{1}, Minglu Zhao\textsuperscript{1}, Keichi Takahashi\textsuperscript{2,1}, Yoichi Shimomura\textsuperscript{2}, and Hiroyuki Takizawa\textsuperscript{2,1}

1. Graduate School of Information Sciences, Tohoku University
2. Cyberscience Center, Tohoku University
Background

• Automatic Software Performance Tuning (Auto-Tuning, AT)
  • Automatically tuning **Performance Knobs** of a code for its target platform
    • Performance knob: parameter of a code affecting the performance e.g., selecting one of multiple code versions for a platform.

• **Build Configuration** = a set of important performance knobs
  • There are many options to compile a code, e.g., **compiler** and its **option flags**.
    • Various kinds of code optimizations are incorporated into each compiler and enabled by compiler option flags.
    • Different compilers provide different optimizations and thus different option flags.

→ A compiler and its option flags must be selected properly.
Compilers and their flags

- Speedup from “gcc –O2” configuration

![Graph showing speedup ratios for different compiler configurations]
Motivation

• A code could potentially have a huge number of build configurations
  • There is no explicit algorithm to find an appropriate build configuration.
  • Full search for finding the best is time-consuming and could be infeasible.

• Research Questions
  • Is it technically feasible to automatically find the best build configuration?
  • From what data, can we predict the best configuration most accurately?
This Work

• Characterizing a code for identifying an appropriate build configuration.
  • Performance Monitoring Counters (PMC)

• What we have done in this work
  • Machine learning models are used for the best build configuration prediction.
    • Predicting the performance with each build configuration.
  • Feature selection to eliminate redundant PMC attributes for inference
Related Work

• Logistic Regression of PMC values (Cavazos et al, 2007)

  Is it possible to improve the accuracy by using more advanced ML models?

• The feasibility of automatically finding an appropriate set of compiler option flags is demonstrated.

  Is it possible to select the compiler itself if multiple compilers are available?

• All available PMC attributes are used for regression.

  Are they all needed? Is it possible to select only necessary features?
Overview of Build Configuration Prediction

Compiled with gcc –O0 and executed.

Redundancy reduction

Z-score standardization
Performance Monitoring Counters

• There are a large number of PMCs available on modern processors.
  • Performance information obtained at runtime (=dynamic information)
  • Available PMC attributes are microarchitecture-dependent 😞
  • PMC values are compiler-dependent 😞

• In this work, PMC values are measured with gcc -O0 (no optimization)
  • Compiler optimization could change the statistics of PMC values.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOT_INS</td>
<td>Total number of instructions</td>
</tr>
<tr>
<td>LST_INS</td>
<td>Number of load and store instructions</td>
</tr>
<tr>
<td>TOT_CYC</td>
<td>Total number of cycles</td>
</tr>
<tr>
<td>SP_OPS</td>
<td>Number of floating-point operations</td>
</tr>
<tr>
<td>L2_TCM</td>
<td>Number of L2 cache misses</td>
</tr>
<tr>
<td>BR_INS</td>
<td>Number of branch instructions</td>
</tr>
</tbody>
</table>
Feature Selection

• Use commonly available PMC attributes across various processors

• Exclude invalid or less useful attributes
  • E.g. some attributes are not affected by build configurations and always constant

• Filter out highly correlated attributes
  • Check if the correlation between two attributes exceeds a threshold.
Feature Selection (cont’d)

• Many PMC values are highly correlated (= multicollinearity)
  • thus expressing identical performance characteristics.

• ML model can learn better from weakly correlated inputs (Alin, 2010).
  → If two PMC values are highly correlated, only one of them is used for ML.

Correlation diagram between PMC attributes

A brighter color means a higher correlation, and many PMC attributes are highly correlated.

Are all PMCs important for characterizing a code? Probably not (experimentally discussed later).
Z-Score Standardization

- PMC attributes have totally different statistics
  - The average values of data cache misses and instruction cache misses are in the orders of $10^9$ and $10^5$, respectively.
    → **Significant differences would lead to information loss at training.**
- In this work, we apply data-scaling and standardization to PMC values.
  - Each value is normalized by TOT_INS (total number of instructions), and then Z-score standardization is applied.

$$Z_i = \frac{x_i - \bar{x}}{S}$$

where $\bar{x}$ and $s$ are the average and standard deviation of $x_i$. 
Machine Learning Model

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization</td>
<td>Adam (rate: 0.0005)</td>
</tr>
<tr>
<td>Epoch count</td>
<td>60</td>
</tr>
<tr>
<td>Loss function</td>
<td>Cross entropy</td>
</tr>
<tr>
<td>Activation function</td>
<td>ReLU</td>
</tr>
<tr>
<td>Batch size</td>
<td>20</td>
</tr>
<tr>
<td>Dropout rate</td>
<td>0.3</td>
</tr>
<tr>
<td>Cross validation</td>
<td>4-fold</td>
</tr>
</tbody>
</table>

The Seventeenth International Workshop on Automatic Performance Tuning (iWAPT2022)
Evaluation Setup

• Build Configurations
  • gcc –O2, gcc –O3
  • icc –O2, icc –O3
  • clang –O2, clang –O3
  • ncc –O2, ncc –O3, ncc –O4

  Compilation for x86 processors with –march=native option

  Compilation for NEC Vector Engine

• Benchmark
  • Test Suite for Vectorization Compilers 2 (TSVC 2)
    • 151 vectorizable loops are provided
    • 1,447 loops are generated by changing their loop lengths.
      • 1D loop length : 100 ~ 512,000
      • 2D loop length : 8 ~ 2,048 (nested)
  • Performance Application Programming Interface (PAPI)
    • PMC values and execution time are obtained at different runs
Evaluation Setup

<table>
<thead>
<tr>
<th></th>
<th>System A</th>
<th>System B</th>
<th>System C</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel Core i7 9700K</td>
<td>AMD EPYC 7402P</td>
<td>AMD EPYC 7702</td>
</tr>
<tr>
<td>VE</td>
<td>NEC VE Type 20B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory</td>
<td>32 GB</td>
<td>256 GB</td>
<td>256 GB</td>
</tr>
<tr>
<td>Linux Kernel</td>
<td>5.11.0</td>
<td>4.18.0</td>
<td>4.18.0</td>
</tr>
<tr>
<td>Compiler</td>
<td>gcc-9.3.0</td>
<td>ncc-3.3.0</td>
<td>gcc-8.4.1, icc-2021.3.0, aocc clang-12.0</td>
</tr>
</tbody>
</table>
Evaluation Metric 1

• PWGA (Penalty-Weighted Geometric Accuracy)

\[
PWGA = \left( \prod_{l=1}^{N} \frac{T_{\text{best},l}}{T_{\text{pred},l}} \right)^{\frac{1}{N}}
\]

• N: Number of data
• \(T_{\text{best},l}\): Execution time of the \(l\)–th code with the best config.
• \(T_{\text{pred},l}\): Execution time of the \(l\)–th code with the predicted config.

PWGA = 1 for perfect prediction.
PWGA becomes smaller if performance with the predicted config is lower than that with the best config.
Evaluation Metric 2

- Speedup Ratio

\[
\text{Speedup Ratio}(l) = \frac{T_{\text{baseline},l}}{T_{*,l}}
\]

- \(T_{*,l}\): Execution Time of the \(l\)-th code with a configuration
- \(T_{\text{baseline},l}\): Execution Time of the \(l\)-th code with gcc –O2. (baseline)
Evaluation Results in PWGA

- ncc –O3: always use ncc –O3 for any loop
- logi : logistic regression (existing work)

1-D: ncc –O3 is almost always best for 1d loops (= long loops).
2-D: proposed approach can select better ones for more 2d loops.
Effect of Feature Selection

• Feature selection improves the prediction accuracy (PWGA) of not only NN but also logistic regression. Without feature selection, the prediction accuracy of NN is almost the same as that of logi. But with feature selection, NN outperforms logi.
Machine Learning Models

- RF: Random Forest
- SVM: Support Vector Machine
- kNN: k Nearest Neighbor
- ncc –O3: always use ncc –O3
- logi: Logistic Regression
- proposed: neural network+feature selection
Speedup by Changing Build Configuration

- (Left) ncc is selected and the speedup ratio is about 19
- (Right) gcc is selected, and the speedup ratio is about 19.

Figure 9: Speedup ratios of each build configuration on a len1d loop (s352 in TSVC 2 with the loop length of 31,039).

Figure 10: Speedup ratios of each build configuration on a len2d loop (s231 in TSVC 2 with the loop length of 87).

Vectorizable but very slow on VE. The proposed approach can predict it.
Conclusions

• Build configuration selection
  • Compiler and its option flags can significantly affect the performance
  • The best configuration for each code could be different

• PMC-based approach
  • **PMC values could be used to predict the best build configuration**
  • Many of PMC attributes are highly correlated and **feature selection to reduce the redundancy could improve the prediction accuracy**.

• Evaluation results with TSVC-2
  • 1-dimensional vectorizable long loops $\rightarrow$ “ncc –O3” works best
  • 2-dimensional nested loops $\rightarrow$ the proposed approach works better than others.
Acknowledgements

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,” and JSPS KAKENHI Grant Number JP20H00593.