A Predictive Modeling Framework for the Compiler Phase-ordering Problem

Amir H. Ashouri, Andrea Bignoli, Gianluca Palermo and Cristina Silvano
Politecnico Di Milano, ITALY

CGO 2016 Poster Session

The Compiler Optimization Problem

- Diversity of today’s architectures including many platforms and compiler frameworks
- Two major classes: (i) The Selection Problem (ii) The Phase-ordering Problem
- Having Different goals in different domains (Power, Performance, Area, etc.)
- Many different optimization flags working with/versus each other within different levels
  - Around 100 opt in LLVM, 200 in GCC and 75 in ICC.

The Phase-ordering Problem

- Conventional approaches can not be useful for the phase-ordering problem as the design space is huge. i.e. having repetition, n opt and m as maximum length: \( |\Omega_{\text{without repetition}}| = \sum_{n=1}^{m} \binom{n+m-2}{m-2} \)
- Classic predictive modeling needs fixed-length feature vectors
- Necessity for adapting specific speedup predictor that comes up with: (i) immediate-speedup prediction or (ii) the complete sequence prediction

Proposed Approach

1. Predictive modeling capable of capturing the correlation between the program features and the compiler optimizations in the complete sequence prediction manner
2. The integration of the predictive models within a compiler framework
3. A mapping technique, utilizing one-hot encoding scheme, capable of mapping variable-length representation of the optimizations to fixed-length representation.
4. Bundling different recurrent compiler optimizations relatively fixed inside each tag.

An example of the proposed mapper on the case we have repetitions and a variable-length up to M=4. Each letter represents a compiler tag containing the different compiler optimizations. In this mode, the predictive models can avoid arbitrary mapping due to having the variable-length in the compiler vectors.

Boosted Traversing Heuristic for the Predictive Models

In Recommender Systems, Adjusted Cosine Similarity is used to correlate users and items. We integrated the idea in our framework, identifying the analogies between the compiler phase-ordering problem and the Recommender Systems field. The users are mapped to the programs in the bin data, while the items to the optimization sequences under analysis. We can then compute the Adjusted Cosine Similarity between item i and j as:

\[
\cos(x, y) = \frac{\sum_{p \in \Omega} \max(0, \min(x, y) - p + 0.5)}{\sqrt{\sum_{p \in \Omega} \max(0, \min(x, y) - p + 0.5)^2}}
\]

The Mapper

The template comes with a number of pre-formatted graphs. By default, all graphs are optimized for the master slide with a size of 16x9. Adjust the height to 10x5 for the slide layout.

In the graphs, axes are labeled as X_AXIS, Y_AXIS, and Z_AXIS, and in the text, as X, Y, and Z. The methods used for parameter tuning are described in the methods section of the paper.

Experimental Results

(A) Performance of the utilization of the proposed mapper

B) Speedup values and comparison with LLVM standard optimization levels –O2 and –O3

Table 1: Average Speedups w.r.t LLVM standard optimization. Reported numbers are A (\%): (A) how fast (in terms of number of predictions) in average the proposed methodology outperforms LLVM standard optimizations. (B) The percentage of the five numbers in the current optimization space

(C) Speedup comparison with respect to the Random Iterative Compilation (RIC)

(D) Speedup comparison with respect to the state-of-the-art immediate-speedup predictors[1,2]

References


Acknowledgements

This project is partially funded by the European Union’s H2020 grant FET-HP joint programme.