S M A R T

Workshop on **S**tatistical and **M**achine Learning approaches applied to **Ar**chi**t**ectures and Compilation

Workshop Proceedings

January 28, 2007

Sofitel, Ghent, Belgium

An Effective Local Search Algorithm for an Adaptive Compiler

Yi Guo, Devika Subramanian, and Keith D. Cooper

Rice University, Department of Computer Science,
Houston, Texas 77005 USA
`{yguo,devika,keith}@rice.edu`

Abstract. Most algorithms currently used to find good compilation sequences in an iterative adaptive compiler, such as genetic algorithms and hill climbing, search in the space of sequences of fixed length. In this paper, we argue that restricting the search to fixed-length sequences limits the ability of search algorithms to find good sequences for some benchmarks. We propose a new local search algorithm that uses greedy construction and cleanup to effectively explore the neighborhood of a start sequence by randomized insertion and deletion of transformations. Preliminary experimental results show that the quality of the local minima found by our local search algorithm are superior to those sequences found by GAs and HCs, and are close to the best sequence we know. Such local minima are found with significantly lower search effort than GAs and HCs working with fixed-length sequences.

1   Introduction

Over the last several years, various groups [1, 6, 3] have studied the code transformation selection and ordering problem in an iterative adaptive compiler. Several search techniques for biased random sampling of the combinatorial space of program-specific optimization sequences have been proposed. Genetic algorithms (GAs) and hill climbing are two popular search algorithms implemented in research iterative compilers. While the choice of specific parameters may vary, these algorithms share one common characteristic: solutions are represented as fixed-length sequences of code transformations and the length of the solution is not varied during the search process. This fixed-length framework is dictated by the use of standard genetic operators, i.e. 1-point crossover and mutation, used in GAs to generate variations for the next generation, and the use of Hamming distance to define neighbors in hill climbing.

Figure 1 presents evidence that current search algorithms based on the fixed-length framework do not find the best solutions for some benchmarks. For `spline`, even after 500 trials, our optimized genetic algorithm achieves less than 45% of the performance speedup of the best known sequence. We believe there are two reasons for this: (1) the fundamental GA operations do not explore the space of optimization sequences effectively. The mutation operator, which randomly generates point variations of a sequence, makes local changes very slowly,
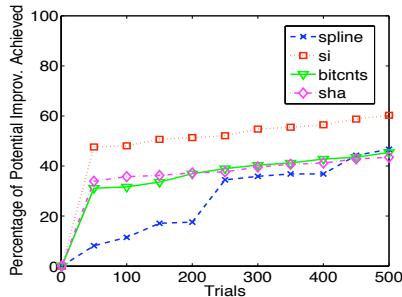
Fig. 1. The percentage of potential improvement achieved by an optimized GA over 500 trials for four benchmarks.

since mutation rates are generally set low. The 1-point crossover operator in GAs is quite destructive and does not generate semantically meaningful variations, and (2) The constraint of fixed-length sequences also limits the range of performance gains that can be achieved. Some benchmarks need fairly short compilation sequences. For those cases, a fixed-length GA spends valuable time and resources learning no-op subsequences to pad the sequence to the full predetermined length. Other benchmarks need longer sequences, and the GA fails to realize the full benefit of the range of optimizations available in the compiler. Attempts to fix these two inherent weaknesses in exploration strategy by tuning algorithm parameters such as the population size, as well as the size of the fixed-length representation, yield little improvement in search performance for several benchmarks.

In this paper, we show that if the neighbor set of a given sequence is explored effectively, the local minima have quality competitive with the best sequences we know. Instead of defining neighbors by Hamming distance, we define neighbors by edit distance, and use greedy construction and cleanup to generate a richer set of meaningful variations by insertion and deletion of transformations. Preliminary experimental results for some benchmarks on the SPARC backend show that the local search algorithm can significantly outperform GAs and hill climbers working in the space of fixed-length sequences.

2 Related Work

Schielke's 1999 paper [2] appears to be the first use of a GA to find compilation sequences. He showed improvement in both code size and execution speed. The framework of the GAs used in the paper is similar to those used in current research iterative compilers. Several techniques have been proposed to improve the GAs' searching performance without changing its fixed-length framework. Kulkarni et al. [6] proposed techniques to speed up searches for compilation sequence in genetic algorithms by detecting and removing redundant trials of equivalent programs, and prohibiting certain dormant or disabled transformations.

Statistical and machine learning techniques have been used to improve the performance of searching. Agakov et al [1] selected a set of benchmarks and learned an offline model for each benchmark. When given a new program, the model of the benchmark that is most similar to the new program is used to focus the search space.

In [3], Grosul describes and compares several variations in an adaptive compiler and found that GAs outperform hill climbing and other algorithms on a budget of several hundreds to a few thousand compilations. In this paper, we use the same experimental setup and compare our local search algorithm to the genetic and hill climbing algorithm in [3]. We show that our local search algorithm significantly outperforms GAs and hill climbing by finding better sequences with far fewer compilations.

Several groups have worked on the problem of finding good parameter settings for specific transformations. Triantafyllis et al. [7] demonstrated the promise of using multiple compilation configurations in a practical compiler. Zhao et al. [8] described an approach for modeling interactions in a predictive framework. Kisuki et al. [5] have used various search algorithms to find good optimization settings for loops in numerical kernels.

3 Neighbor Exploration: Finding Local Minima

The definition of neighbor is fundamental to search algorithms. In the fixed-length framework, it is easy to define neighbors by Hamming distance: two sequences of the same length are considered neighbors if they differ in exactly one character. The experiments in [3] show that the hill climbers using Hamming distance do not deliver satisfactory results. Our local search algorithm defines neighbors by edit distance, i.e., two sequences are considered neighbors if one can be derived from another by inserting or removing one transformation.

Two procedures are used to find local minima in sequence space by exploring the neighborhood of a starting sequence: cleanup and greedy construction. The cleanup procedure removes transformations that are redundant or detrimental to the quality of the given sequence. Such transformations can appear during both greedy construction and random sequence generation. Greedy construction extends the base sequence one transformation at a time. At each step it picks a transformation and inserts it into the position that delivers the most improvement. If a transformation does not yield improvement, it is discarded.
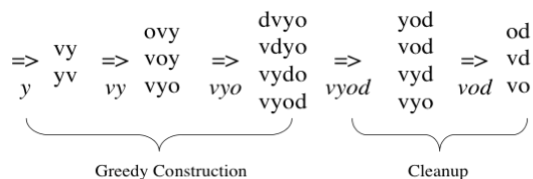

Fig. 2. Neighbor Exploration

Figure 2 shows an example of greedy construction and cleanup. Transformation v, o and d are inserted into sequence y, and transformation y is removed by cleanup. Our algorithm starts the search from random sequences. Local minima are found by iteratively running greedy construction and cleanup. Detailed algorithm description can be found in our technical report [4].

4 Experiments

In this paper, we use 16 transformations, which are listed in Table 3.1 on page 17 of [3]. They are low-level code transformation based on ILOC, which is a RISC-like assembly language. Our implementation ensures that each transformation can take any valid ILOC program as input and producesa valid ILOC program as output. This feature allows us to run the compilation transformations in arbitrary order, which is critical for an adaptive compiler. When using the default compilation sequence *rvzcodtvzcod*, the performance of the code generated by our ILOC compiler is comparable to the *GCC* compiler using *-O2* flag.

We compare our local search algorithm to GA and Hill Climbing (HC). The parameter settings for GA and impatient hill climbing(HC-10) are described on page 76 in [3]. For GA, we tried three length settings, 15, 20 and 25, and the curve represents the best among the three. The sequence length for HC is fixed at 15. Figure 3 shows the search performance for three algorithms within 1000 trials. The speedup of a sequence is normalized to 0-100% where the default sequence *rvzcodtvzcod* is set to 0% and the best sequence is set to 100%.

According to Figure 3, our search algorithm excels other algorithms after 200 trials, and after 1000 trials, the quality of sequence we found is close to the best. Table 4 shows the length of the best sequence we known and the best GA's length setting. The best length settings of GA is program-specific, and there is no obvious relation to the length of the best sequence.

Benchmark	Source Suite	Len. of the Best Seq.	Best GA's Len. Settings
spline	fmm	13	20
si	spec	24	20
bitcnts	mibench	29	25
sha	mibench	29	20

Table 1. Benchmark

5 Conclusion

This paper considered two hypotheses for the poor performance of GAs and HCs on complex compilation sequence spaces for some benchmarks. The first is the fixed-sequence length limitation and the second is the choice of genetic operators for constructing variations to explore during search. We introduces a local search algorithm with a richer neighborhood definition generating variable length sequences. We demonstrate that this new local search algorithm outperforms GAs and HCs on a set of benchmarks, both on the quality of solutions and the search effort needed to find them.
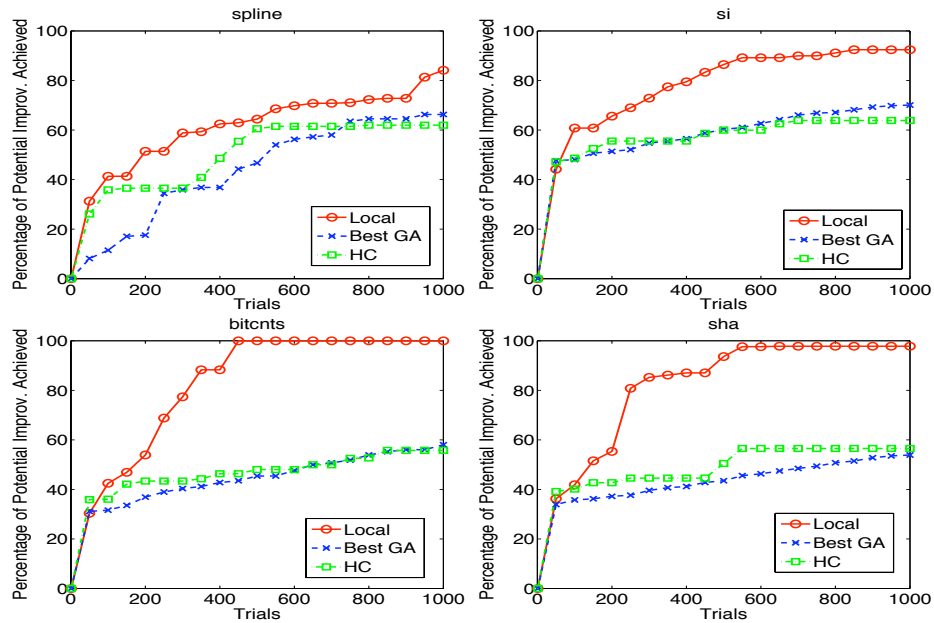
Fig. 3. Average percentage of potential improvement achieved by our local search algorithm, GA and HC on the SPARC backend for four benchmarks

References

1. AGAKOV, F., BONILLA, E., CAVAZOS, J., FRANKE, B., FURSIN, G., O'BOYLE, M. F. P., THOMSON, J., TOUSSAINT, M., AND WILLIAMS, C. K. I. Using machine learning to focus iterative optimization. In *CGO '06: Proceedings of the International Symposium on Code Generation and Optimization* (2006).
2. COOPER, K. D., SCHIELKE, P. J., AND SUBRAMANIAN, D. Optimizing for reduced code space using genetic algorithms. In *LCTES '99* (1999).
3. GROSUL, A. *Adaptive Ordering of Code Transformations in an Optimizing Compiler.* PhD thesis, Rice University, 2005.
4. GUO, Y., SUBRAMANIAN, D., AND COOPER, K. A new local search algorithm for effective exploration of compilation sequences. Tech. rep., Rice University, 2006.
5. KISUKI, T., KNIJNENBURG, P. M. W., AND O'BOYLE, M. F. P. Combined selection of tile sizes and unroll factors using iterative compilation. In *PACT '00: Proceedings of the 2000 International Conference on Parallel Architectures and Compilation Techniques* (2000).
6. KULKARNI, P. A., HINES, S. R., WHALLEY, D. B., HISER, J. D., DAVIDSON, J. W., AND JONES, D. L. Fast and efficient searches for effective optimization-phase sequences. *ACM Trans. Archit. Code Optim. 2*, 2 (2005), 165–198.
7. TRIANTAFYLLIS, S., VACHHARAJANI, M., VACHHARAJANI, N., AND AUGUST, D. I. Compiler optimization-space exploration. In *CGO '03: Proceedings of the international symposium on Code generation and optimization* (2003).
8. ZHAO, M., CHILDERS, B., AND SOFFA, M. L. Predicting the impact of optimizations for embedded systems. In *LCTES '03: Proceedings of the 2003 ACM SIGPLAN conference on Language, compiler, and tool for embedded systems* (2003).