HOT METHOD PREDICTION USING SUPPORT VECTOR MACHINES

Sandra Johnson, Dr S Valli
Department of Computer Science and Engineering, Anna University, Chennai – 600 025, India.
sandra_johnk@yahoo.com, valli@annauniv.edu

ABSTRACT

Runtime hot method detection being an important dynamic compiler optimization parameter, has challenged researchers to explore and refine techniques to address the problem of expensive profiling overhead incurred during the process. Although the recent trend has been toward the application of machine learning heuristics in compiler optimization, its role in identification and prediction of hot methods has been ignored. The aim of this work is to develop a model using the machine learning algorithm, the Support Vector Machine (SVM) to identify and predict hot methods in a given program, to which the best set of optimizations could be applied. When trained with ten static program features, the derived model predicts hot methods with an appreciable 62.57% accuracy.

Keywords: Machine Learning, Support Vector Machines, Hot Methods, Virtual Machines.

1 INTRODUCTION

Optimizers depend on profile information to identify hot methods of program segments. The major inadequacy associated with the dynamic optimization technique is the high cost of accurate data profiling via program instrumentation. The major challenge is how to minimize the overhead that includes profile collection, optimization strategy selection and re-optimization.

While there is a significant amount of work relating to cost effective and performance efficient machine learning (ML) techniques to tune individual optimization heuristics, relatively little work has been done on the identification and prediction of frequently executed program hot spots using machine learning algorithms so as to target the best set of optimizations. In this study it is proposed to develop a machine learning based predictive model using the Support Vector Machine (SVM) classifier. Ten features have been derived from the chosen domain knowledge, for training and testing the classifiers. The training data set are collected from the SPEC CPU2000 INT and UTDSP benchmark programs. The SVM classifier is trained offline with the training data set and it is used in predicting the hot methods of a program which are not trained. This system is evaluated for the program’s hot method prediction accuracy.

This paper is structured as follows. Section 2 discusses related work. Section 3 gives a brief overview of Support Vector Machines. In Section 4 this approach and in section 5 the evaluation methodology is described. Section 6 presents the results of the evaluation. Section 7 proposes future work and concludes the paper.

2 RELATED WORK

Machine learning techniques are currently used to automate the construction of well-designed individual optimization heuristics. In addition, the search is on for automatic detection of a program segment for targeted optimization. While no previous work to the best of our knowledge has used ML for predicting program hot spots, this section reviews the research papers which use ML for compiler optimization heuristics.

In a recent review of research on the challenges confronting dynamic compiler optimizers, Arnold et al. [1] give a detailed review of adaptive optimizations used in the virtual machine environment. They conclude that feedback-directed optimization techniques are not well used in production systems.

Shun Long et al. [3] have used the Instance-based learning algorithm to identify the best transformations for each program. For each optimized program, a database stores the transformations selected, the program features and the resulting speedup. The aim is to apply appropriate transformations when a new program is encountered.

Cavazos et al. [4] have applied an offline ML technique to decide whether to inline a method or not. The adaptive system uses online profile data to identify “hot methods” and method calls in the hot methods are in-lined using the ML heuristics.
Cavazos et al. [5, 12] have also used supervised learning to decide on which optimization algorithm to use: either graph coloring or Linear scan for register allocation. They have used three categories of method level features for ML heuristics (i.e.) features of edges of a control flow graph, features related to live intervals and finally, statistical features about the size of a method.

Cavazos et al. [11] report that the best of compiler optimizations is method dependent rather than program dependent. Their paper describes how, logistic regression-based machine learning technique trained using only static features of a method, is used to automatically derive a simple predictive model that selects the best set of optimizations for individual methods within a dynamic compiler. They take into consideration the structures of a particular method within a program to develop a sequence of optimization phases. The automatically constructed regression model is shown to out-perform hand-tuned models.

To identify basic blocks for instruction scheduling Cavazos et al. [20] have used supervised learning. Monsifrot et al. [2] have used a decision tree learning algorithm to identify loops for unrolling. Most of the work [4, 5, 11, 12, 20] is implemented using machine learning to identify the best procedure clone for the current run of the program. M. Stephenson et al. [18] have used two machine learning algorithms, the nearest neighbor (NN) and Support Vector Machines (SVMs), to predict the loop unroll factor. None of these approaches aims at prediction at the method level. However, machine learning has been widely used in work on branch prediction [21, 22, 23, 24].

3 SUPPORT VECTOR MACHINES

The SVM [15, 16] classification maps a training data (xi, yi), i = 1,...,n where each instance is a set of feature values xi ∈ Rn and a class label y ∈ {+1, -1}, into a higher-dimensional feature space φ(x) and defines a separating hyperplane. Only two types of data can be separated by the SVM which is a binary classifier. Fig. 1 shows a linear SVM hyperplane separating two classes.

The linear separation in the feature space is done using the dot product φ(x)·φ(y). Positive definite kernel functions k(x, y) correspond to feature space dot products and are therefore used in the training algorithm instead of the dot product as in Eq. (1):

\[ k(x, y) = (\Phi(x) - \Phi(y)) \]

The decision function given by the SVM is given in Eq. (2):

\[ f(x) = \sum_{i=1}^{n} \gamma_i k(x_i, x) + b \]

where b is a bias parameter, x is the training example and \( \gamma_i \) is the solution to a quadratic optimization problem. The margin of separation extending from the hyperplane gives the solution of the quadratic optimization problem.

4 HOT METHOD PREDICTION

This section briefly describes how machine learning to identify the best procedure clone for the current run of the program. M. Stephenson et al. [18] have used two machine learning algorithms, the nearest neighbor (NN) and Support Vector Machines (SVMs), to predict the loop unroll factor. None of these approaches aims at prediction at the method level. However, machine learning has been widely used in work on branch prediction [21, 22, 23, 24].
learning could be used in developing a model to predict hot methods within a program. A discussion of the approach is followed by the scheme of the SVM-based strategy adopted in this study.

**Figure 2: System architecture of the SVM-based hot method predictive model**

**4.1 The approach**

Static features of each method in a program are collected by offline program analysis. Each of these method level features forms a feature vector which is labeled either hot or cold based on classification by a prior execution of the program. The training data set thus generated is used to train the SVM-based predictive model. Next, the test data set is created by offline program analysis of a newly encountered program. The trained model is used to predict whether a method is hot or cold for the new program. An offline analysis on the Low Level Virtual Machine's (LLVM) [6] bytecode representation of the programs provides the training as well as the test data set. The system architecture for the SVM-based hot method predictive model is shown in Fig.2 and it closely resembles the architecture proposed by the authors C. L. Huang et. al. [26]. Fig. 3 outlines the strategies for building a predictive model.

1. Create training data set.
   a. Collect method level features
      i. Calculate the specified feature for every method in a LLVM bytecode.
      ii. Store the feature set in a vector.
   b. Label each method
      i. Instrument each method in the program with a counter variable [25].
      ii. Execute the program and collect the frequency of the execution of each method.
      iii. Using the profile information, each method is labeled as either hot or cold.
      iv. Write the label and its corresponding feature vector for every method in a file.
   c. Steps (a) & (b) are repeated for as many programs as are required for training.
2. Train the predictive model.
   a. The feature data set is used to train the SVM-based model.
   b. The predictive model is generated as output.
3. Create test data set.
   a. Collect method level features.
      i. Calculate the specified features for every method in a new program.
      ii. Store the feature set in a vector.
      iii. Assign the label ‘0’ for each feature vector in a file.
4. Predict the label as either hot or cold for the test data generated in step 3 using the predictive model derived in step 2.

**Figure 3: System outline**

**4.2 Extracting program features**

The "C" programs used for training are converted into LLVM bytecodes using the LLVM frontend. Every bytecode file is organized into a single module. Each module contains methods which are either user-defined or pre-defined. Only static features of the user-defined methods are extracted from the bytecode module, for the simple reason that they can be easily collected by an offline program analysis. Table 1 lists the 10 static features that are used to train the classifier. Each feature value of a method is calculated in relation to an identical feature value extracted from the entire bytecode module. The collection of all the feature values for a method constitutes the feature vector $x_i$. This feature vector $x_i$ is stored for subsequent labeling. Each feature vector $x_i$ is then labeled $y_i$ and classified as either hot
(+1) or cold (-1) based on an arbitrary threshold scheme described in the next section.

Table 1: static features for identifying hot methods.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Number of loops in a method.</td>
</tr>
<tr>
<td>2</td>
<td>Average loop depth of all the loops in the method.</td>
</tr>
<tr>
<td>3</td>
<td>Number of top-level loops in a method.</td>
</tr>
<tr>
<td>4</td>
<td>Number of bytecode level instructions in the method.</td>
</tr>
<tr>
<td>5</td>
<td>Number of Call instructions in a method.</td>
</tr>
<tr>
<td>6</td>
<td>Number of Load instructions in a method.</td>
</tr>
<tr>
<td>7</td>
<td>Number of Store instructions in a method.</td>
</tr>
<tr>
<td>8</td>
<td>Number of Branch instructions in the method.</td>
</tr>
<tr>
<td>9</td>
<td>Number of Basic Blocks in the method.</td>
</tr>
<tr>
<td>10</td>
<td>Number of call sites for each method.</td>
</tr>
</tbody>
</table>

4.3 Extracting method execution frequencies

Hot methods are frequently executing program segments. To identify hot and cold methods within a training program, profile information is gathered during execution. The training bytecode modules are instrumented with a counter variable in each user-defined method. This instrumented bytecode module is then executed and the execution frequency of each method is collected. Using this profile information, the top ‘N’ most frequently executed methods are classified as hot. This system keeps the value “N” as the “hot method threshold”. In this scheme of classification, each feature vector (x_i) is now labeled y_i (+1) and y_i (-1) for hot and cold methods respectively. The feature vector (x_i) along with its label (y_i) is then written into a training dataset file. Similarly, the training data set of the different training programs is accumulated in the file. This file is used as an input to train the predictive model.

+1 1:1 2:1 3:1 4:0.880046 5:2.51046 6:0.875912 7:0.634229 8:1.23119 9:1.59314 10:29
-1 1:0 2:0 3:0 4:1.16702 5:1.25523 6:1.02197 7:3.8266 8:1.50479 9:1.83824 10:2
+1 1:2 2:2 3:2 4:1.47312 5:0.83682 6:1.89781 7:1.47992 8:2.59918 9:2.81863 10:3

Figure 4: Sample training data set

The general format of a feature vector is

y_i 1:X_1 2:X_2 3:X_3 ... j:X_j

where the labels 1, 2, 3, ..., j are the feature numbers and X_1, X_2, ..., X_j are their corresponding feature values. Fig. 4 shows a sample of three feature vectors from the training dataset collected for the user-defined methods found in the SPEC benchmark program. The first feature vector in Fig. 4 is a hot method and is labeled +1. The values of the ten features are serially listed for example ‘1’ is the value of feature 1 and ‘29’ of 10. The value ‘1’ of feature 1 indicates the percent of loops found in the method. The “hot method threshold” used being 50%, 4 out of the 8 most frequently executed methods in a program are designated as hot methods. The first element in each vector is the label y, (+1 or -1). Each element of the feature vector indicates the feature number followed by the feature values.

4.4 Creating test data set

When a new program is encountered, the test data set is collected in a way similar to the training data set, except that the label is specified as zero.

0 1:0 2:0 3:0 4:0.552341 5:0.970874 6:1.14155 7:0.363636 8:0.385356 9:0.862069 10:4
0 1:1 2:1 3:1 4:1.26249 5:0 6:2.51142 7:2.90909 8:1.15607 9:1.2069 10:40

Figure 5: Sample test data set

4.5 Training and prediction using SVM

Using the training data set file as input, the machine learning algorithm SVM is trained with default parameters (C-SVM, C=1, radial basis function). Once trained the predictive model is generated as output. The derived model is used to predict the label for each feature vector in the test data set file. The training and prediction are done offline. Subsequently, the new program used for creating test data set is instrumented. Executing this instrumented program provides the most frequently executed methods. The prediction accuracy of the system is evaluated by comparing the predicted output with the actual profile values.

5 EVALUATION

5.1 Method

Prediction accuracy is defined as the ratio of events correctly predicted to all the events encountered. This prediction accuracy is of two types: hot method prediction accuracy and total prediction accuracy. Hot method prediction accuracy is the ratio of correct hot method predictions to the actual number of hot methods in a program, whereas total prediction accuracy is the ratio of correct predictions (either hot or cold) to the total number of methods in a program. Hot method prediction accuracy is evaluated at three hot method threshold levels: 50%, 40% and 30%.

The leave-one-out cross-validation method is used in evaluating this system. This is a standard machine learning technique where ‘n’ benchmark programs are used iteratively for evaluation. One out of the ‘n’ programs is used for testing and the rest ‘n-1’ programs are used for training the model. This is repeated for all the ‘n’ programs in the benchmark.
5.2 Benchmarks

Two benchmark suites, SPEC CPU2000 INT [17] and UTDSP [13] have been used for training and prediction. UTDSP is a C benchmark and SPEC CPU2000 INT has C and C++ benchmarks. Evaluation of the system is based on only the C programs of either benchmark. The model trained from the “n-1” benchmark programs in the suite is used to predict the hot methods in the missed out benchmark program.

5.3 Tools and platform

The system is implemented in the Low Level Virtual Machine (LLVM) version 1.6 [6]. LLVM is an open source compiler infrastructure that supports compile time, link time, run time and idle time optimizations. The results are evaluated on an Intel (R) Pentium (R) D with 2.80 GHz and 480MB of RAM running Fedora Core 4. This system uses the libSVM tool [7]. It is a simple library for Support Vector Machines written in C.

![Hot Method Prediction Accuracy](image)

Figure 6: Hot method prediction accuracy on the SPEC CPU2000 INT benchmark

6 RESULTS

Fig. 6 shows the prediction accuracy of the trained model on the SPEC CPU2000 INT benchmark program at three different hot method thresholds: 50%, 40% and 30%. The hot method prediction accuracy for all C programs on the benchmark is found to vary from 0 % to 100 % with an average of 57.86 %, 51.43% and 39.14% for the three hot method thresholds respectively. This averages to 49.48% on the SPEC CPU2000 INT benchmark suite. Similarly, on the UTDSP benchmark suite, in a 0% to 100% range, the hot method prediction accuracy averages for the three thresholds are 84%, 81% and 62% respectively. This averages to 76% on the UTDSP benchmark suite. Overall, this new system can obtain 62.57% hot method prediction accuracy.

![Total Method Prediction Accuracy](image)

Figure 7: Total prediction accuracy on the SPEC CPU2000 INT benchmark

The total method prediction accuracy on the SPEC CPU2000 INT and UTDSP benchmark suites is shown in Fig. 7 and 9. The total method prediction accuracy for all C programs on the SPEC CPU2000 INT varies from 36 % to 100 % with an average of 68.43%, 71.14% and 71.14% for the three hot method thresholds respectively. This averages to 70.24%. The average prediction accuracies obtained on the UTDSP benchmark suite are 69%, 71% and 58% respectively for 50%, 40% and 30% hot method thresholds. This averages to 66%. Overall the system predicts both hot and cold methods in a program with 68.15% accuracy.

7 CONCLUSION AND FUTURE WORK

Optimizers depend on profile information to identify the hot methods of program segments. The major inadequacy associated with the dynamic optimization technique is the high cost of accurate data profiling via program instrumentation. In this work, a method is worked out to identify hot methods in a program using the machine learning algorithm, the SVM. According to our study, with a set of ten static features used in training the system, the derived model predicts total methods within a program with 68.15% accuracy and hot methods with 62.57% accuracy. However, hot method prediction is of greater value because optimizations will be more effective in these methods.

Future work in this area is aimed at improving the prediction accuracy of the system by identifying more effective static and dynamic features of a program. Further research in this system can be extended to enhance it to a dynamic hot method prediction system which can be used by dynamic optimizers. Applying this approach, the prediction accuracy of the other machine learning algorithms can be evaluated to build additional models.
8 REFERENCES


[10] Christophe Dubach, John Cavazos, Björn


