Hardware Counters and Data Augmentation for Predicting Vectorization

Rahul Sridhar

rsridha2@uci.edu (ID: 41608676)

December 12, 2017

I. Abstract

The process of vectorization entails converting the scalar implementation of a computer program into a vector implementation. This paper attempts to build on recent work involving the usage of hardware performance counters and techniques from machine learning to predict auto-vectorization of compilers by validating similar machine learning models on a different architecture and compiler, and also shows the benefits of data augmentation through sample synthesis on such applications. Using predictive models along with data augmentation on hardware performance data, we are able to successfully predict whether a compiler was able to auto-vectorize a program or not with 78% accuracy.

II. Introduction

Vectorization, or more specifically, auto-vectorization, refers to a transformation of a code that results in the simultaneous application of an operation on vectors of data. This is an important optimization done by compilers in order to speed up the execution of a program. It would hence be beneficial to understand the performance characteristics of auto-vectorizable programs and subsequently be able to predict if a particular piece of code would be vectorized by a compiler or not. Watkinson et al.[1] had explored the use of dynamic run-time data (hardware performance counters) in order to predict vectorization of compilers using supervised machine learning models. This paper attempts to build on their experiments. While their paper showed results from experiments with GCC and ICC compilers on Harpertown, Sandybridge, and Haswell architectures (amongst others), this paper collects relevant performance data using LLVM's clang compiler on an Intel Core i5-5250U, Broadwell, 8 GB RAM, 1.6 GHz machine. Since Broadwell and Haswell are related architectures, this work hypothesizes that the choice of predictive models, model parameters, and features similar to the ones shown in [1] for the Haswell architecture should work well in this case too. If this is true, then machine learning models built for one set of applications/architecture could potentially be re-used for other related architectures for which we might not have sufficient data.

Additionally, given that data is scarce or too expensive to collect in a lot of applications, this paper tries to explore any potential benefits that data augmentation/synthesis might offer in this scenario. The goals of this paper are thus three-fold:

- Identify performance characteristics of auto-vectorizable programs
- Test predictive model adaptability from one application-architecture combination to another
- Check if data augmentation can be used to improve a model's predictive capability of auto-vectorization

III. Methodology

III.1. Data

The dataset is courtesy the TSVC *"Test Suite for Vectorizing Compilers"* benchmark [2]. It consists of 151 loops written in C. It was primarily developed to evaluate auto-vectorization capabilities of compilers. The number of loops vectorized by clang was 66 (#non-vectorized loops = 85).

III.2. Data collection

Apple's Instruments applications was used to collect data from hardware performance counters. Data for a total of 21 features were collected. The set of features mostly fell in the following buckets:

- Memory Flow
- Cycles
- Instructions
- Branching

More details about these features are included along with the code. These data were collected for the target program 'tsc.c' without performing vectorization (by using the '-fno-vectorize' flag). The vectorization report (number of loops vectorized vs. non-vectorized) was obtained by enabling the vectorization flag and by experimenting with the '-force-vector-width' and '-force-vector-interleave' flags, which are used to control the vectorization SIMD width and loop unrolling factor respectively.

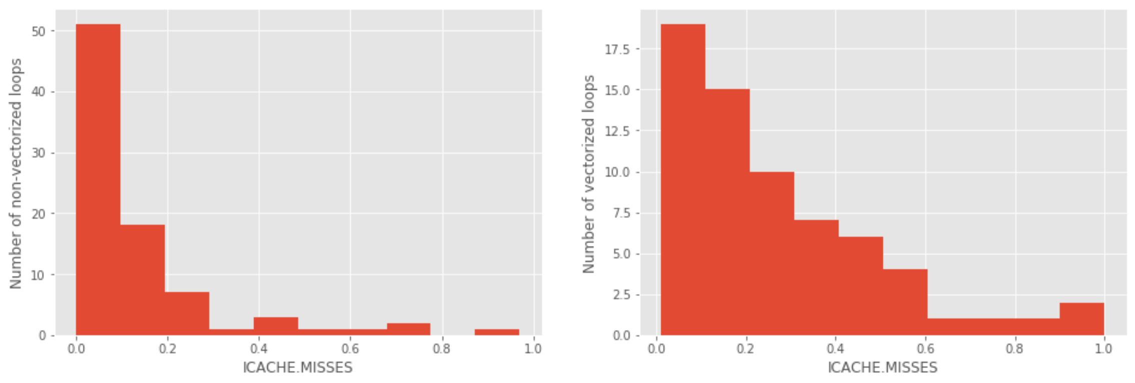

Figure 1: *Distribution of ICACHE.MISSES across Non-Vectorized and Vectorized loops (after normalization)*

III.3. Feature Selection

Given the small number of data points available, and given that data was collected for 21 features, it would be unwise to build the predictive models on all of the features. Hence, a feature selection process was implemented to pick the features that would help the most in predicting auto-vectorizability. More specifically, the Mutual Information (MI) metric was used to select the top features. For each feature, MI was calculated between it and the discrete target variable (was the loop vectorized or not?). MI is a probabilistic measure of the mutual dependence between two variables, and can be used to quantify the amount of information that can be obtained from one variable through another. If MI is high, then the feature contains a lot of information about the target variable, and should potentially aid in predictive model building. The top p features were then selected by thresholding the MI values (the threshold used was the mean MI value computed across all feature-target pairs). This resulted in a total of 10 features. The most important feature turned out to be the number of instruction cache misses, and its (normalized) distribution across non-vectorized and vectorized loops is shown in Figure 1. Evidently, the distributions are highly different, and this is probably why it turned out to be an important feature. The distributions of other features have been included in the file 'Rahul_CS243_Data_Visualization'.

III.4. Predictive Modeling

The following classification models were experimented with before zeroing in on the final set of models to be used for the data augmentation experiment:

- *Naive Bayes*: A simple, probabilistic model that assumes that each feature is independent of the other (given the target class)
- *Gradient Boosting*: A model in which multiple decision trees are fitted on the dataset, with each subsequent decision tree trained to predict the residual (or the error) from the previous decision tree

- *Support Vector Machines*: A model that tries to find a hyperplane that best separates the data in high dimensions
- *Logistic Regression*: A probabilistic model that uses the sigmoid function on a linear combination of the features to predict the class
- *Random Forests*: An ensemble of decision trees that uses bootstrapping (creating datasets by sampling with replacement) and randomized feature subsets at each node split of the tree to predict the class
- *K-nearest neighbors*: A simple, non-parametric model that classifies a sample to be of the class of its k closest neighbors in the data space (closeness could be defined based on a distance metric such as Euclidean distance)

A more detailed explanation of these models is beyond the scope of this short paper. The baseline for all these models is the majority classifier, which just predicts the majority class for every data point. In this case, the majority class is the 'non-vectorized' class. Models were compared based on their 20-fold cross-validation accuracies on the entire dataset (151 loops). Furthermore, in order to compare the performance of models pre and post data augmentation, a train-test split of the dataset was performed. For the pre-augmentation phase, 101 data points were randomly sampled out of the 151 and used for model training. The rest of the data (50 data points) were used as test data. For the post-augmentation phase, the above 101 data points along with 101 new augmented samples were used for training the models. The same 50 data points used above were used for computing the model's test performance, and the test results were compared from both phases to decide if the data augmentation was beneficial. The model parameters were not tuned differently for each phase - they both used the same set of model parameters.

III.5. Data augmentation

Data augmentation was performed with two different methods, and the model results were recorded pre and post augmentation. The following two augmentation methods were used for this experiment:

- *SMOTE* (Synthetic Minority Oversampling Technique) [3]: This method was primarily proposed to tackle problems with imbalanced datasets. It can be used to create new samples from the minority class so as to equal the number of samples in each class. The way it generates new samples is by taking the difference in feature values between two existing samples in the dataset, then randomly samples a number between 0 and 1, and multiplies the difference with this number to generate a new data point. Since the method is generic, it can be used to create samples from any class, and not just from the minority class.

- *Synthpop* [4]: It uses CART (Classification and Regression Trees) to generate samples in a non-parametric manner. Parametric options (that impose assumptions on the type of data distribution) are available, but in the experiments that have been performed, the non-parametric option has been used.

Predicting Vectorizability		
Method	Test data accuracy	Overall accuracy
Majority	0.52	0.56
Naive Bayes	0.64	0.67
Gradient Boosting	0.72	0.6
Random Forests	0.72	0.63
Support Vector Machines	0.64	0.62
Logistic Regression	0.68	0.68
k-nearest neighbors	0.68	0.65

Table 1: *Classification performance (overall accuracy corresponds to the 20-fold cross validation accuracy)*

IV. Results

The results from the first experiment (predicting vectorizability without data augmentation) are shown in Table 1. The predictive models are all benchmarked against a majority classifier, which just predicts the majority class in the data for every data point. Random Forests, Logistic Regression, and k-NN seem to be performing relatively well. These models were also effective in predicting vectorizability in other compilers [1]. This probably indicates that machine learning models built on one particular architecture-compiler combination could be used effectively in applications run on other architectures and compilers. Even though Gradient Boosting also seems to be performing well, it was highly prone to overfit on the training data (given the limited amount of data we have). Hence, this model was not considered for the data augmentation experiments.

The results from the augmentation experiments are shown in Tables 2 and 3. As is evident, for the most part, the data augmentation seems to have improved the models' capabilities to understand the patterns in the data. This improvement also seems to be dependent on the type of model. Models such as Logistic Regression do not seem to benefit from augmentation. In fact, augmentation has a slight detrimental effect on performance. This is because Logistic Regression relies on uncovering linearity in the structure of the data. Since data augmentation using methods such as SMOTE relies on similarity metrics, the newly generated data samples are similar to the ones in the training dataset, and this does not add new information to what the linear model has already discovered from the original data. On the other hand, models such as Random Forests and k-NN benefit from the augmentation, with a 8% increase in accuracy observed using SMOTE with k-NN, and a 6% increase in accuracy observed using synthpop with Random Forests. These models seem to be more reliant on minor variations in the data (when data is limited).

One point to note is that we might be better off relying on models such as Random Forests rather than k-NN since the latter depends on distances between data points, which are not entirely trustworthy in high-dimensional spaces. Also, the results observed were directionally similar in different training-test data splits as well (the numbers reported are for one particular split). Finally, given the realm of limited (dynamic) data that we are working with, obtaining a 78% accuracy using predictive models (against a majority classifier's performance of 52%) is a considerable improvement.

Test data accuracy		
Method	Pre augmentation	Post augmentation
Random Forests	**0.72**	**0.76**
Logistic Regression	0.68	0.66
***k*-NN**	**0.68**	**0.76**

Table 2: *Data augmentation with SMOTE (101 augmented samples)*

Test data accuracy		
Method	Pre augmentation	Post augmentation
Random Forests	**0.72**	**0.78**
Logistic Regression	0.68	0.66
***k*-NN**	**0.68**	**0.72**

Table 3: *Data augmentation with synthpop (101 augmented samples)*

V. Conclusion

We evaluated the effectiveness of using hardware performance data in supervised machine learning models to predict compiler auto-vectorization (for clang). We were able to make predictions with 72% accuracy using decision tree-based models such as Random Forests. We also observed that the knowledge gained by building machine learning models on one particular architecture such as Haswell could potentially be transferred to applications on similar architectures such as Broadwell. This point could be applicable beyond these architectures, and could be tested further. Finally, we were also able to improve the performance of predictive models for vectorization to 78% using data augmentation techniques such as SMOTE and synthpop.

References

[1] N. Watkinson, A. Shivam, Z. Chen, A. Veidenbaum, and A. Nicolau. "Using hardware counters to predict vectorization". *CECS Technical Report*, 17-01, 2017.

[2] S. Maleki, Y. Gao, M. J. Garzarán, T. Wong, and D. A. Padua. “An evaluation of vectorizing compilers”. *Parallel Architectures and Compilation Techniques - Conference Proceedings (PACT)*, pages 372–382, 2011.

[3] N.V. Chawla, K.W. Bowyer, L.O. Hall, and W.P. Kegelmeyer. “SMOTE: Synthetic minority over-sampling technique”. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.

[4] B. Nowok, G. M. Raab, and C. Dibben. “synthpop : Bespoke creation of synthetic data in R”. *Journal of Statistical Software*, 74:1–26, 2016.