Predicting LLVM Loop Unroll Factors Using Machine Learning Methods

Remzi Can Aksoy
remzican@umich.edu

Jungho Bang
bjungho@umich.edu

Sulaimon Lasisi
slasisi@umich.edu

ABSTRACT

In this project, we tried to find heuristics to predict the best loop unroll factor for the given loop features by applying various machine learning approaches. In order to collect data set, we built a custom LLVM pass that is able to extract features from each loop and measure the execution duration by applying different unroll factors.

Time measuring experiment was done across loops from depth 1 through 5 to get execution times. The unroll factor with the best execution time for each loop was chosen as the label along with the loop's features which were used as the input to our machine learning models.

We evaluated accuracy and performance of different machine learning models compared to default settings in LLVM and random selection of decisions. We found that Random Forest and Decision Tree performs better than other models in both metrics. Finally, we discussed further improvements that can be done to achieve better accuracy and performance.

1. INTRODUCTION

For the final project, our group has worked on improving the loop unrolling heuristics through machine learning approaches for compiler optimization. First, we reviewed previous works on the compiler heuristics optimization problems with different machine learning methods.

The first paper was *A Machine Learning Approach to Automatic Production of Compiler Heuristics* (A. Monsifrot et al.)[3]. Since this paper addressed loop unrolling heuristics, we could get a lot of insights about unrolling such as what kind of features we want to extract from the loop. The second paper was *Automatic Construction of Inlining Heuristics using Machine Learning* (D. Simon el al.)[4]. Even though it was not about loop unrolling, we could learn general idea of applying machine learning methods to solve compiler optimization problem. Whereas the first paper used only decision tree, the second one tried multiple approaches including Neuro-Evolution of Augmenting Topologies(NEAT) and Genetic Algorithms. After the background research, we initially set our project goal to improve the heuristics of Monsifrot's paper using different machine learning approaches.

Through the initial experiment, however, we found the limitations of the Monsifrot's loop unrolling heuristics paper. The paper's approach needs trip count as a feature to train models. Also, the decision tree in the paper provides only whether or not to unroll. However, for about 65 percent of the loops from our experiment, we could not determine the trip count statically. Without static trip count, LLVM's UnrollLoop function needs unroll-factor to unroll loops dynamically.

Finally, we re-defined our problem for this project as finding heuristics to predict the best loop unroll factor (or leave as-is) for the given loop features.

Features
Loop ID (hashed value)
Total instructions count
Arithmetic operations count
Array accesses count
Conditional instructions count
Integer operations count
Float operations count
Load instructions count
Store instructions count
Block in the loop count
Blocks in the function count
Loop Depth

Table 1: Loop features extracted

2. IMPLEMENTATION

To extract features from each loop and measure the execution duration by applying different unroll factors, we developed a LLVM pass.

2.1 Feature extraction

The custom LLVM pass we built can extract features of each loop in the compile time and export the data into a file. In addition to the features that Monsifrot's paper used, we added more to differentiate the characteristics of each loop. Table 1 shows the features that we collected.

2.2 Loop unrolling

To unroll loops, we used LLVM's UnrollLoop function in a different way than the LLVM LoopUnrollPass does. First, we modified the default behavior when trip count is known. Instead of using the threshold for total instructions count, our pass always uses runtime unrolling with unroll factor. Also, whereas LoopUnrollPass uses default unroll factor 8 for unrolling, our pass takes a parameter to determine the factor. In the experiment, we performed unrolling with 8 different factors for each loop.

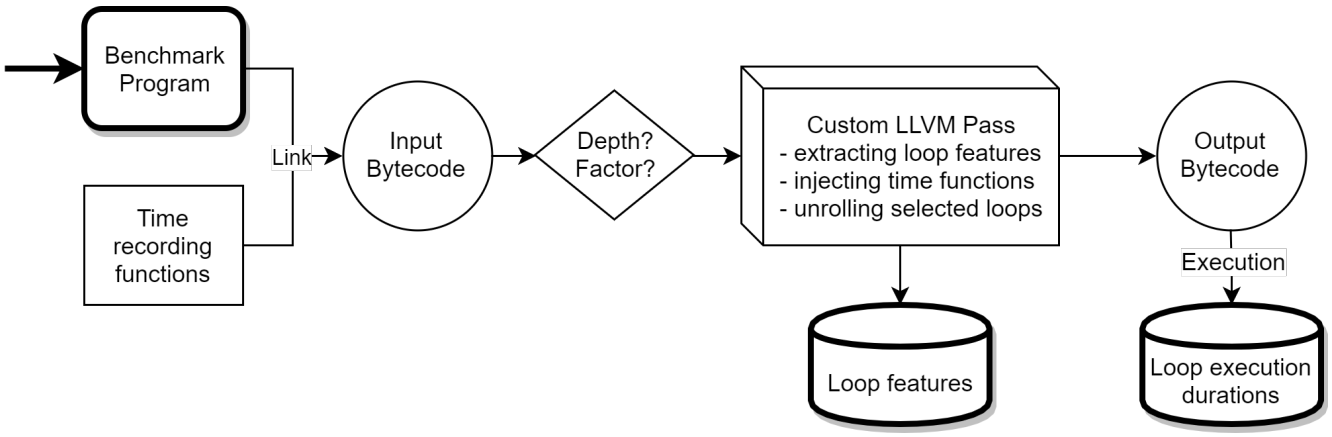

Figure 1: Unroll / Measuring System Architecture

2.3 Execution time measuring

In order to compare the performance of each unroll factor, we needed to measure the execution duration of each loop separately. Since we had to run experiments for hundreds of benchmark programs, we decided to automate time-stamp injection.

To achieve this, each benchmark program gets linked with the time recording program that we built. It has functions to record starting and finishing time of a given ID. Our LLVM pass injects function calls at pre-header and exit block of each loop with identifiable loop hash ID. Also, the pass adds another call at the very end of the main function to export the profile. The profile has unroll factor and execution duration of each loop.

If we unroll all loops at once, the time measuring function calls inside the loop body will affect the duration of the outer loop. If we have to apply the pass and generate individual bytecode for each loop, we could solve the overhead problem, however, it will take unreasonably long time to experiment. We found that loops in the same depth wouldn't affect other loop's measuring. Therefore, we added loop depth as a parameter to our pass, so that it performs function injection and unrolling only for the specified depth.

3. EXPERIMENTATION

3.1 Dataset

The datasets in the experiment come from loops extracted from the NASA Kernel Benchmark Programs [1] and WCET benchmarks [2]. In total, we have a set of 102 programs to extract loops from. Loops from depths 1 through 5 from these programs were used in our experiments. Our range of loop-depth was because given the few number of programs we had, we wanted to extract as many loops as possible from each program. Table 2 shows the distribution of loops along depths in our experiment.

3.2 Time Measuring

Our experiments were executed on the course machines provided. We split our benchmarks into two sets to minimize the time it took for each set to run and produce results. We ran each loop-depth configuration five times and calculated the average execution time based on the execution results. As mentioned above, we went through each loop in depth 1 to 5 for each program and performed unrolling with factors in the set [1,2,3,4,5,8,10,16]. The unroll factor of 1 is the same as leaving the loop as it is while other factors unrolled the loop into blocks determined by the unroll factor. The entire experiment took 4.4 hours on one machine and 6.2 hours on the other course machine. The output of the execution experiment was the corresponding time it took each loop to run in its given unroll factor configuration.

3.3 Machine Learning

The machine learning experiment used data from the time-measuring step as input to the different machine learning models we explored . We collected and merged execution times for each unroll factor configuration of each loop. After merging all unroll factor times per loop, we chose the unroll factor with the fastest execution time as the label for each loop and the features recorded were the input data for the machine learning models we tried. Initially, our choice of unroll factor values [1,2,3,4,5,8,10,16] was based on powers of 2, as well as numbers that divide nicely like 5 and 10. However, after collecting unroll factors labels for best execution times, we discovered that all the unroll factors we chose were well represented as labels in the dataset. The distribution of unroll factors as labels in our dataset is shown in Table 3:

Once we got labels for all loops in the dataset, we split into train/test set in a 90/10 ratio. We also kept the distribution of the entire dataset similar to the distributions within the train and test set to ensure that data imbalance was at a minimum. We used the SKLearn python library to perform machine learning experiments on our dataset. The SKLearn library provided sufficient hyperparameter selec-

Loop Depth	Count
1	604
2	308
3	90
4	24
5	2
Total	1028

Table 2: Loop depth distribution

Unroll Factor	Label Count
1	121
2	150
3	140
4	135
5	119
8	102
10	172
16	89
Total	1028

Table 3: Label distribution

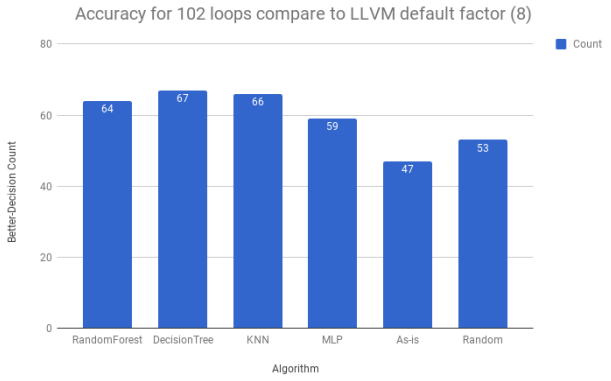

Figure 2: Accuracy of unroll factor decisions over LLVM default unroll factor(8) for 102 loops

tion support that worked for our problem. Multi-class classification support, neural network solvers for our data size and iteration tuning affordances were available and used in our machine learning experiment. We ran classification experiments using Neural Network, Decision Tree, K-Nearest Neighbors and Random Forest ensemble methods. In the results section below, we discuss the outcome of these experiments and comparisons with the LLVM Default, unchanged loop (as-is) and the Random Labeling baseline.

4. RESULTS

Our charts try to answer following questions:

- How are the accuracy of decisions by different algorithms compared to LLVM default unroll factor?
- How are the accuracy of decisions by different algorithms compared to the as-is version which is the version without any unrolling?
- How are the total execution time of all benchmarks with the decisions of our models compared to as-is performance or LLVM default performance?

While making these comparisons, we also tried to compare with a model which randomly decides for each unroll factor to see if we have performance improvement over totally arbitrary decisions and to see if our models are trained well in order to take better decisions than just a random chance.

Figure 2 shows our first comparison. In this chart, each bar represents the number of better decisions out of 102 loops compared to choosing 8 as a constant unroll factor

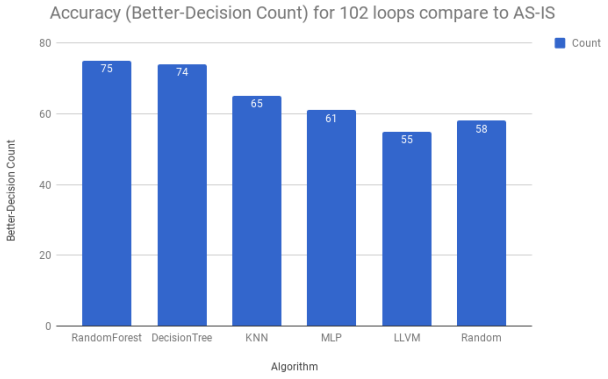

Figure 3: Accuracy of unroll factor decisions over as-is version of the 102 loops

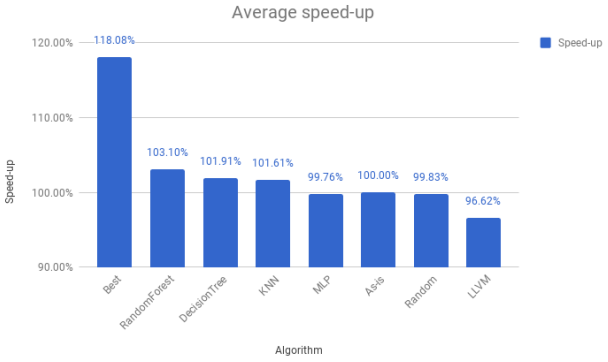

Figure 4: Speed up reached by different models compare to as-is versions of the loops

which is LLVM default. For example, Random Forest, Decision Tree and KNN are able to take better decision in 64, 67 and 66 loops compared to the LLVM default choice of using 8 as the unroll factor always. Random selection also made 53 better choices than the LLVM default unroll factor.

Similarly, Figure 3 shows performance of different models compared to as-is version. In other words, each bar represents how many times our models can have better decisions instead of leaving the loops without unrolling. For instance, Random Forest and Decision Tree were able to choose better unroll factors 74% of the loops compared to no unrolling, while a random selection could choose 57% of the loops.

In our last comparison, we analyzed the total execution time of all programs to see overall speed up which is provided by each model. In this comparison, we normalized the execution time of each loop by its as-is execution time to prevent domination of longer loops in the results. Figure 4 shows the speed up achieved by each model and also the maximum speed up which a model can get if it takes best decision for each loop. As can be seen in the figure, the best possible execution time is 18% better than as-is execution time of all 102 loops. In this range, Random Forest can achieve 3% speed up as our best model.

5. CONCLUSION

At the end of our experiments, we can conclude that some machine learning models especially Random Forest and Decision tree can perform better than LLVM default heuristics and Random selection. However, Multi-layer Perceptron (Neural network) model could not perform well in all metrics. We think the main reason behind this is our data set size was not big enough to train the neural net.

Also, there is some room for further improvement according to Figure 4. We figure out that one essential reason behind why the machine learning models are far from the best possible speed up is the way of training data creation. When we label the features with only best unroll factor, the models cannot learn how much better this factor is than the second and third best decisions. In other words, the models overfit to the best unroll factors while second and third best factors can also be good decisions. We lose the valuable association between unroll factor, features and execution times for these discarded unroll factors.

Finally, there are some differences between the paper *Predicting Unroll Factors Using Supervised Classification* which does very similar work to our projects [5]. We think that the following differences between our work and this paper could be important for us to reach a better accuracy and performance:

- This paper uses -1 for trip count if it is unknown, while we decided to not use trip count as a modification to our initial paper.
- It started with large set of features and applied feature selection algorithms (Mutual Information Score & Greedy Feature Selection) to prevent overfitting. However, we just used some additional features which we found important in addition to the features from our initial paper.
- This paper has larger dataset for training and they only work on innermost loops. While we used the loops from all depths because of limited dataset available to us.

Areas where we thought we made improvements in comparison to the paper are listed below:

- This paper uses unroll factors just from the set of [1,8]. While we decided to use also some bigger factors like 10 and 16. Since there are some loops which perform better with unroll factor 10 and 16, we think that using bigger unroll factors was a better decision.
- This paper just uses KNN and SVM, but we tried four different models including KNN. We found that Random Forest and Decision Tree was the best performing models, therefore it can be said that our selection of models was an improvement on their work.

6. CONTRIBUTIONS

Each member in our group made fair amount of contributions to the project. We have had regular meetings to discuss the issues and follow-up works. All members participated in overall progress of the project, yet the following is the main focus of each member:

- Remzi Can Aksoy: Running experiments, data cleaning for machine learning - 33.33%
- Jungho Bang: Implement LLVM pass for unrolling and time measurement - 33.33%
- Sulaimon Lasisi: Collect and organize benchmarks, perform machine learning - 33.33%

7. REFERENCES

[1] D. Bailey and B. JT. The nas kernel benchmark program. 09 2014.

[2] J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper. The mclardalen wcet benchmarks: Past, present and future. 15:136–146, 01 2010.

[3] A. Monsifrot, F. Bodin, and R. Quiniou. A machine learning approach to automatic production of compiler heuristics. In *Proceedings of the 10th International Conference on Artificial Intelligence: Methodology, Systems, and Applications*, AIMSA '02, pages 41–50, London, UK, UK, 2002. Springer-Verlag.

[4] D. Simon, J. Cavazos, C. Wimmer, and S. Kulkarni. Automatic construction of inlining heuristics using machine learning. In *Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, CGO '13, pages 1–12, Washington, DC, USA, 2013. IEEE Computer Society.

[5] M. Stephenson and S. Amarasinghe. Predicting unroll factors using supervised classification. In *Proceedings of the International Symposium on Code Generation and Optimization*, CGO '05, pages 123–134, Washington, DC, USA, 2005. IEEE Computer Society.