Using CBMC on Codes with Dynamic Results

Yueting Zhu

Oregon State University

Abstract

CBMC is a bound model checker for C and some kinds of C++ program. CBMC runs all the possible inputs under an exactly number of loops and give some counter example if the code has something wrong. CBMC is normally used to check codes or functions with static results. Which are the results with certain number or other constant performances. In this paper, I will talk about the application of CBMC on the codes with results which each on is in a range.

Key Words: CBMC ; Dynamic Result; SSE; Range; Statistic

Introduction

CBMC is a Bounded Model checker for C and C++ programs. It supports C89, C99, most of C11 and most compiler extensions provided by gcc and Visual Studio [1]. However, CBMC has many disadvantages which cannot be solved perfectly. First, CBMC can only find there is an error or not. The error's location and the number of the error will not be presented to the users. Because CBMC can only find the counterexamples under finite unwindings, so it can only tell the counterexamples instead of the information of errors. It is not convenience for user to find out the error in there code. Secondly, it cannot be used to check the code with huge range of loop. As the resource it takes to compute all the possible inputs are enormous, the running time it takes will geometric growth when we add the number of loops we check. It is impossible to check all the loops in an unlimited bound. When we run a code on CBMC, we should add a upper bound of loop that we check to avoid running without ending. Another disadvantage of CBMC is if the code with the results can be treated as correct in a range, CBMC treats them as a counterexample as the performance. For example, as we know, when we compute some codes' running time, the result should be in a range, some may have a trueness error. It may lead to a range for correct result that all the number in this range may all be treated as correct. Because every time we run the code,

the results are all different but may all correct. In this kind of code, we even allow some error exist in small probability. CBMC cannot recognize the correct range, and the worse thing is CBMC will treat all the results as error if it is different from the result in the hardness code. Because of the small probability errors, it is impossible to give a range to the results in hardness code. This problem mostly appears in efficiency study codes or simulation codes. In these codes, the loops with some results based on time and unspecific space coordinate will be a problem when we use CBMC on those codes.

SSE

Sum of Squared Error(SSE) is a simple way to justify the noise in the results. When we have a datasets for the results in the same condition, we can get an average data from this data by $\frac{\sum_{i=1}^{n} d_i}{n}$. Using SSE, the noise of each data is obtained from the expression from $(d_i - d_{average})^2$. Then the average noise $N_{average}$ is

$$\frac{\sum_{i=1}^{n}(d_i - \frac{\sum_{i=1}^{n} d_i}{n})^2}{n} \quad (1)$$

From the $N_{average}$, we may compute the range of results by adding or reducing the $N_{average}$ from the $d_{average}$.

Using SSE in CBMC

As we know, CBMC works as comparing the results from harness code and target code to show the counterexamples from the different results. As we are focusing the dynamic result code, the result in the harness also should be a range. Our aim is keeping the result range from target code in the result range from harness code. To use SSE, we need a large number of samples. So first, we should run the target code for 1000 times (this number can be smaller as 100 and bigger as 10000 or more). The more times we run the more accurate result we get. After doing SSE on each result in target code, we get a range with $[(N_{average} - \sqrt[2]{d_{average}})\sim(N_{average} + \sqrt[2]{d_{average}})]$. As a compare, in the harness code, we also run the code for the same times. Then we can get a number of results from harness code. Choose the biggest and smallest result from the harness code result sets to be the upper bound and lower bound for the range. Then compare the ranges from the harness and target code. If the range from the target code is in the range of harness code, we can guarantee that the result is correct. Otherwise, there is at least one error in the target code.

Error Analysis

The reason why we cannot set a range to harness is the small probability errors exist. By using SSE method, we can make the small probability errors effect to be extremely low. When we add the number of samples, the effect of the small probability errors will be lower. So the only problem is if the error is “supper” error which may affect the correct result. In this method, we treat that if there are “supper” errors in the result, the result should be treated as wrong. Normally, a correct code cannot produce any “supper” errors. So the degree of the error is limited by the sample number and the accuracy of other correct results.

Runtime Analysis

Suppose the time complexity of running target code and harness code are O(p) and O(m) the number of samples is n. Then the time complexity of running this program is O(np+2n+mn). From this, we know that runtime is grown a lot after using this method. If we want to get a more accurate judgment, n will be increased, so the time it costs will be increased.

Experiment and Problem

I have already check a single problem with running time which just adds two functions' runtime together in the loop and the result is such as what I expect. But when I checked a bigger code, it takes me a lot of time. Huge runtime is the biggest problem cost by using this method. I did not successfully solve this problem. I thought of this problem and get an idea, which is using OpenMP parallel method to run this code on more OpenMP threads. Because OpenMP and CBMC are both based on the C. By doing this, the runtime will be reduced a lot. Unfortunately, I did not successfully use OpenMP in CBMC. However, the parallel can reduce the runtime of CBMC [5].

Conclusion

Combining SSE and CBMC can do the Bounded Model Checking for codes with dynamic results. This method make it possible to use CBMC do the Bounded Model Checking on efficiency study, robot, and simulation of phenomenon. There are also some problems that I cannot solve right now. I will finish the idea using parallel programming to reduce the runtime of this method in the future.

References

[1] "The CBMC Homepage." The CBMC Homepage. Web. 10 June 2015.

[2] Rebagliati, Nicola. "Strict Monotonicity of Sum of Squares Error and Normalized Cut in the Lattice of Clusterings." Proceedings of the 30th International Conference on Machine Learning (ICML-13). 2013.

[3] Chandra, Rohit. Parallel programming in OpenMP. Morgan Kaufmann, 2001.

[4] Kroening, Daniel, and Michael Tautschnig. "CBMC–C bounded model checker." Tools and Algorithms for the Construction and Analysis of Systems. Springer Berlin Heidelberg, 2014. 389-391.

[5] Vlugt, Thijs JH. "Efficiency of parallel CBMC simulations." Molecular Simulation 23.1 (1999): 63-78.