Token-based Approach for Real-time Plagiarism Detection in Digital Designs

Han Wan, Kangxu Liu, Xiaopeng Gao
School of Computer Science and Engineering
Beihang University
Beijing, China
{wanhan, liukangxu, gxp}@buaa.edu.cn

***Abstract*—This Research to Practice Work in Progress Paper presents a token-based approach to detecting plagiarism in university courses with hardware programming assignments. Detecting plagiarism manually is a difficult and time-consuming work. In the last two decades, various of plagiarism detection tools have been developed. These techniques could be mainly divided into the following categories: Textual Match, Program Dependence Graph Comparison, Abstract Syntax Tree Analysis and Low-Level Form Code Comparison. Although there had been a lot of researches on detecting code clones in software programming languages (e.g. Basic, C/C++, Java, Python, etc.), research that focused on hardware description languages is still lacking. Based on the effective of the locality sensitive hash function (simhash), which was usually used in detecting near-duplicates for web crawling, we proposed an improved real-time plagiarism detection approach for Verilog HDL (hardware description language) programming assignments. The core detecting steps are extracting weighted tokens from source code as high-dimensional feature, and mapping it to a *f*-bit fingerprints with simhash technique. On account of the syntax characteristics of Verilog HDL, a token extraction strategy was designed to maximize the valid information that a fixed length hash value could represent. Experiments over real course data sets were conducted to evaluate the performance of token-based approach comparing with an existing plagiarism detection tool (Moss). The result shows that our token-based approach does qualify the plagiarism detecting job for both online-query and batch-query in digital designs. Furthermore, token-based plagiarism detection approach could enable conduct incremental plagiarism detection for a single submission without excessive overhead. Finally, we also give a discussion of current way limitations and future research directions.**

Keywords—token-based, plagiarism detection, program similarity, Verilog HDL, simhash

I. INTRODUCTION

As a low-cost and highly flexible choice for implementing and verifying digital system, field-programmable gate array (FPGA) has been widely used in digital signal processing, high-performance computing, artificial intelligence and other fields [1][2][3]. The behavior of the FPGA device could be configured and simulated easily using hardware description language (HDL), such as VHDL and Verilog HDL, which makes it possible to offer FPGA-based hardware experiment courses in college.

However, same as the traditional programming courses, code plagiarism also occurs in hardware design assignments. Plagiarism means copying existing code from peer student or past students and then pasting it into their own design with some obfuscation. If such behavior cannot be detected effectively, it will undermine the fairness of the course and lead to a poor teaching result. Hence, it is necessary for teaching staffs to discover possible plagiarism in time.

During the CS education development process, various plagiarism detection tools have been proposed. C. Liu et al. developed the GPLAG, a tool that using program dependence graphs (PDGs) to detect plagiarism [4]. For better search efficiency, they designed a statistical lossy filter also. Theoretically, GPLAG can be easily extended to any programing language with a specific parsing front-end, but it only supports C/C++ and Java currently. L. Prechelt et al. developed a web service based on Greedy String Tiling [5] algorithm, called JPlag [6]. It compares programs pairwise by turning them into a string of canonical tokens. C. Zhao et al. proposed a solution called BuaaSim, which combined compiling optimization and disassembling techniques together to detect plagiarism in C/C++ programs [7]. Their experiment showed that it achieved significantly higher performance than JPlag in practice. A. Aiken at Stanford University provided a web-based tool that can analyze code written in 23 languages including VHDL and Verilog HDL, called Moss [8]. The method that it used to measure the similarity between programs is a document fingerprinting algorithm called winnowing [9]. Overall, GPLAG and BuaaSim both require the input programs to be compilable. JPlag using a pairwise comparison strategy, which makes it relatively slow when processing plenty of samples. As far as we know, Moss is the only tool that supports almost all programming languages. However, it's a closed-source system supporting batch queries only which makes it unsuitable for retrieving real-time information of potential plagiarism after submission.

Although there has been a lot of researches on detecting code clones in traditional programming languages (e.g. Basic, C/C++, Java, Python), research that focused on hardware description languages is still lacking. As far as we known, Moss is the only tool that can handle Verilog HDL. However, Moss supports batch-query only, which means that user needs to upload all history submission files to identify whether a new submission is a plagiarism, even those history submissions have already been

978-1-5386-1174-6/18/$31.00 ©2018 IEEE

tested. In this paper, a token-based approach is proposed for real-time plagiarism detection in HDL programming assignments. With the effectiveness of simhash technique, our approach achieved a similar performance with the Moss from Stanford University along with the capability of supporting both online-query and batch-query.

II. Locality Sensitive Hashing

The key to plagiarism detection is how to measure the similarities between programs. The common solution is to extract features (e.g., program dependence graphs, canonical tokens, abstract syntax tree analysis, etc.) from each program and then use pairwise comparison to find out possible plagiarism pairs. However, the time complexity of any detection algorithm that using pairwise comparison strategy is not less than $O(n^2)$. As for the normal duplicates detection, there is an easy approach that using hashing techniques like checksum functions to identify all duplication in $O(n)$. If there is a function that can map similar inputs to similar hash values, then we can use it to solve near-duplicate detection problem.

Charikar proposed a locality sensitive hash function constructed on rounding algorithm, called simhash [10]. By mapping high-dimensional features to a *f*-bit fingerprint, it aims to minimize the distance between similar inputs. Furthermore, Charikar suggested an algorithm for searching approximate nearest neighbor in Hamming distance. Researchers from Google proved that simhash is practically useful for detecting near-duplicates in massive web pages [11]. Besides, they also designed a hybrid method which supports both online-query (single fingerprint) and batch-query (multiple fingerprints) for Hamming space searching problem. There are two simple brute approaches for solving Hamming distance problem within a constant time: one is building a sorted table of all fingerprints, the other is pre-compute all possible fingerprint combinations that have a Hamming distance no more than k with existing fingerprints. Both of the above are impractical because of the huge space complexity. Eventually, Manku at el. developed an hybrid algorithm that lies between these two approaches, which had been proven to work properly with a balanced time and space complexity.

In this paper, we will show how to apply simhash technique along with the improved Hamming space searching algorithm into plagiarism detection process, and prove the effective of our proposed approach by comparing the performance with an existing plagiarism detection system called Moss on real data sets.

III. Plagiarism Detection Process

As shown in Fig. 1, sources files from each submission are feed into plagiarism detection process as input and the suspected plagiarism pairs are output. The entire process of our token-based plagiarism detection approach consists of the following steps: pre-processing source files, extracting tokens, fingerprinting with simhash, indexing and clustering to get result.

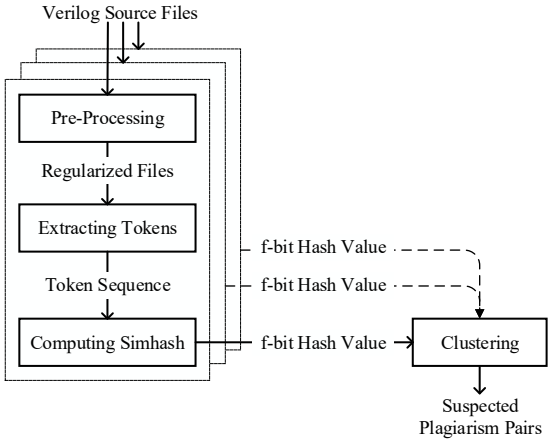

Fig. 1. Plagiarism detection process.

A. Pre-processing Source Files

It can be noticed that Verilog HDL supports C-style code comments. In practice, one of the common types of plagiarism that we had observed including adding, deleting and modifying comments in the code. Additionally, Verilog HDL is case-sensitive, and all keywords are entered in lower case. For more effective plagiarism detection, remove all comments and extra white space characters, and then convert all ASCII characters into lowercase for case-insensitive searching in the following steps.

B. Extracting Tokens

The next step is to extract weighted token sequence from each submission. In fact, some of the syntax rules which are used by Verilog HDL are similar to those in C programming language. For instance, as for module, the basic logic entity in Verilog HDL, its definition is delimited by the keywords *module* and *endmodule*. As shown in Fig. 2, the port declaration specified by keyword *input*, *output* or *inout* is either following the module definition header or within it. To reduce the searching space, some lines of code that are less relevant to the hardware behavior, such as module definition headers, port declarations and some reserved keywords, are removed in this step.

```
/* module declaration */
module 8bit_adder(a, b, cin,
cout, result);
   /* port declarations */
   input [7:0] a;
   input [7:0] b;
   input cin;
   output cout;
   output [7:0] result;
   /* behavior related code */
   assign {cout, result} = a +
b + cin;
   endmodule
```

(a)

```
/* combined declarations for
module and port */
module 8bit_adder(
   input [7:0] a,
   input [7:0] b,
   input cin,
   output cout,
   output [7:0] result
);
   /* behavior related code */
   assign {cout, result} = a +
b + cin;
   endmodule
```

(b)

Fig. 2. Modules written in different syntax styles. (a) is the old style. (b) is the ANSI-style.

After comparing a series of different token extraction strategies, sampling the input with fixed-length tokens is chosen in our implement. Given a constant *d* as the length of token, for the code that has removed irrelevant lines and whitespace characters, we cut it into a sequence of substrings of length d as shown in Fig. 3. Furthermore, the corresponding weight of a token is defined as the number of times that it appears in the token set.

C. Fingerprinting with Simhash

We implemented Charikar's simhash algorithm (as shown in Fig. 4) using Python 3.5. As described in section II, simhash is a dimensionality reduction algorithm that mapping a document represented by weighted tokens into a *f*-bit fingerprint. For the web crawling deduplication application, Manku et al. [11] from Google validated that a 64-bit fingerprint is a balance of accuracy and speed for large-scale pages. In our implementation, we used a 128-bit fingerprint for a better accuracy with acceptable speed, since that the size of plagiarism detection problem is much smaller than search engine. And we used xxHash [12] to generate the basic hash of each token as it is an extremely fast non-cryptographic hash algorithm working at speeds close to RAM limits.

D. Indexing and Clustering

The final step of the plagiarism detection process is to find out the suspected similar pairs from existing sample set based on their fingerprints. It can be transformed into the classic Hamming Distance Problem [13], which is to identify all existing fingerprints whose Hamming distance from a given fingerprint is not greater than k. Manku et al. also proposed a hybrid algorithm with a set of sorted table and a two-level query strategy for the above problem. They reported that it could shrink the query time to $O(\log(p_i))$(p_i is the total number of bits in the chosen blocks of each table) when dealing with a completely random bit sequence. In the next section, we will elaborate the experiment results under different parameter k.

IV. Result and Discussion

In this section, the performance of our proposed approach and the Moss system are compared. As mentioned in section I, Moss is a web service for detecting plagiarism in programming field that has been widely used by teachers and TAs since 1994. In our past practice, it had been used to locate students who might copy others' hardware design homework. It turned out that Moss did perform well. For each suspected pair, Moss will give a similarity score calculated from the number of matching lines between those two files. And the report given by Moss is in a descending order of similarity score, which users need to confirm whether the similar pair is a plagiarism one by one.

```
Original code:
    assign {cout, result} = a + b + cin;

Remove whitespace characters:
    assign{cout,result}=a+b+cin;

Token sequence:
    [``assig", ``n{cou", ``t,res", ``ult}=", ``a+b+c", ``in;"]
```

Fig. 3. Transforming code into token sequence.

```
Input:
    f: The length of fingerprint in bits;
    P: The set of tokens for current submission;
    W: The set of weights corresponding to P;
Output:
    hash: The f-bit fingerprint of current submission;
initialize hash with f-bit full-zero vector
for token, weight in P, W do
    token_hash = hash_function(token)
    for i ← 0 to f do
        if token_hash[i] == 1 then
            hash[i] ← hash[i] + weight
        else
            hash[i] ← hash[i] − weight
        end if
    end for
end for
for i ← 0 to f do
    if hash[i] > 0 then
        hash[i] ← 1
    else
        hash[i] ← 0
    end if
end for
```

Fig. 4. Charikar's simhash algorithm.

A. Experiment Results

We plan to compare the performance of Moss and token-based approach with respect to the number of plagiarism detected and precision under different parameter k. In this experiment, we used two real data sets collected in the Fall 2017 iteration. As our course paced in each week, students were allowed to modify their designs at any time before each week's in-class checkpoint. During the in-class test, students are required to extend the design they accomplished that week. Students who failed in the test could refine their design and take the test again in the next week checkpoint. Thus, to cover all possible plagiarism in the learning process, we chosen everyone's last submission in each week to make up the test data sets.

The first data set included 431 Verilog HDL projects and each implemented a single-cycle MIPS processor, with an average of 443 lines of code. The second one consists of 419 pipelined MIPS processor projects with an average of 1026 lines of code. We use "-n 1000" option in Moss to expand the number of matching files shown in the results, since the same student's submission in different weeks may be considered as similar ones by Moss.

As shown in TABLE I. , Moss showed reasonably great detection capability over both data sets. For the first data set, 8 of the top 10 suspected pairs reported by Moss were confirmed to be plagiarism by teaching staffs, which achieved a precision of 60.00%. Correspondingly, it can be observed that as the max

distance k increases, the number of suspected pairs found by our approach increases along with the precision decreases as shown in Fig. 5 and Fig. 6. Experiment conducted on the second data set showed a similar trend as the first experiment. In general, our approach and Moss are comparable in accuracy with a suitable k value.

B. Parameter Selection

To sum up, the above experiments with real data sets indicate that our approach performance is close to that of Moss. While the choice of Hamming distance parameter k is crucial for the performance. A trade-off between precision and recall is clearly showed in Fig. 5 and Fig. 6. According to our experience, $k = 16$ or $k = 18$ is reasonable for detecting most plagiarism in complex Verilog HDL assignment, as it balances the recall and precision. A larger k means that more suspected plagiarism pairs may be found, but meanwhile it will increase the workload of handling false positive samples. Users can determine the value of k by detecting samples collected from previous course iteration.

C. Real-time Detection

Compared with the existing tool, our method achieves a similar accuracy over real data sets. Moreover, using simhash, a state-of-the-art locality sensitive hashing technique, enables our approach to conduct incremental plagiarism detection for a single new submission without excessive overhead, which is the greatest weakness of Moss. Specifically, when using Moss to conduct plagiarism detection in an online-judge system, for each newly submitted sample, it needs to be uploaded together with all known samples to the Moss server, though those samples have already been tested, which is a waste of network bandwidth and computing resources.

D. Limitation and Future Work

Both our approach and Moss focused on comparing text similarity when detecting plagiarism in Verilog HDL. The advantage of text-based approach is that it doesn't require the input programs to be compilable. But those two methods could be affected by actions such as replacing statements with equivalent expression, adding dead code (the code that never be executed) and splitting parallel blocks. To solve this issue, GPLAG and BuaaSim pointed out a feasible path for software programming language. In the future, we will investigate how to combine compile related techniques to enhance plagiarism detection in hardware description language.

TABLE I. PERFORMANCES OF MOSS ON REAL COURSE DATA SETS

Data Set	Parameter	Confirmed	Found	Precision
1	Top 5	4	5	80.00%
	Top 10	6	10	60.00%
	Top 15	8	15	53.33%
2	Top 5	5	5	100.00%
	Top 10	8	10	80.00%
	Top 15	11	15	73.33%

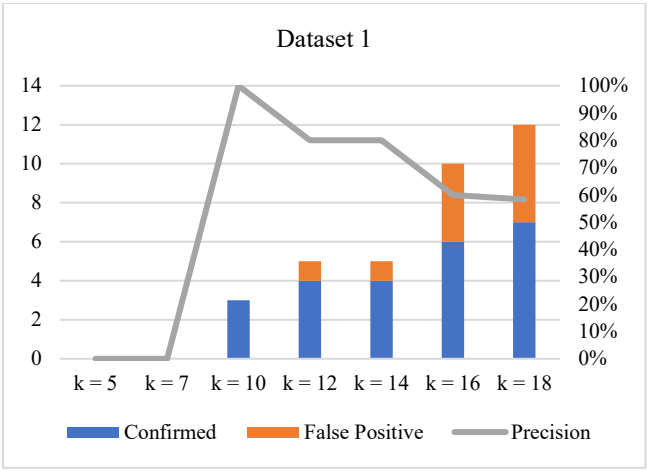

Fig. 5. Performances of token-based approach on dataset 1.

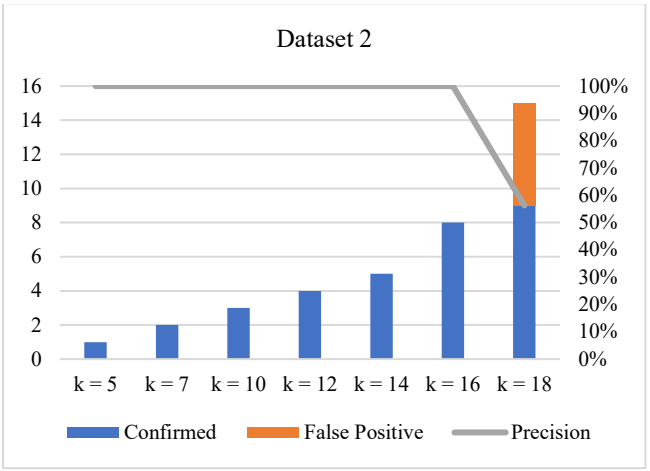

Fig. 6. Performances of token-based approach on dataset 2.

V. CONCLUSION

Most plagiarism detection tools are designed for processing the entire collection of submissions at a time. For teaching staffs, discovering possible plagiarism timely is the key to effective intervention. In this paper, we proposed a token-based approach using simhash technique for detecting plagiarism in digital designs implemented by Verilog HDL, which supports both batch-query and online-query. We conducted experiments on real data sets to compare the performance of our approach and Moss. The result indicated that our approach is practicable for incremental detecting plagiarism in hardware experiment courses. We also discussed the limitation of current methods and the possible future research directions.

ACKNOWLEDGMENT

This work was partially supported by the Beihang University education reform funding and by the Tencent collaborative grant from the New Engineering and Technical Disciplines on this topic: “Exploration and Practice of Computer Architecture Courses Reform”.

REFERENCES

[1] R. J. Petersen and B. L. Hutchings, "An assessment of the suitability of FPGA-based systems for use in digital signal processing," in Field-Programmable Logic and Applications, 1995, pp. 293–302.

[2] M. C. Herbordt et al., "Achieving High Performance with FPGA-Based Computing," Computer, vol. 40, no. 3, pp. 50–57, Mar. 2007.

[3] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "A Survey of FPGA Based Neural Network Accelerator," arXiv:1712.08934 [cs], Dec. 2017.

[4] C. Liu, C. Chen, J. Han, and P. S. Yu, "GPLAG: Detection of Software Plagiarism by Program Dependence Graph Analysis," in Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, 2006, pp. 872–881.

[5] M. J. Wise, "String similarity via greedy string tiling and running Karp-Rabin matching," Online Preprint, Dec, vol. 119, 1993.

[6] L. Prechelt, G. Malpohl, and M. Philippsen, "Finding plagiarisms among a set of programs with JPlag," J. UCS, vol. 8, no. 11, p. 1016, 2002.

[7] Z. C. Y. H. J. Maozhong, "Approach based on compiling optimization and disassembling to detect program similarity," Journal of Beijing University of Aeronautics and Astronautics, vol. 6, p. 025, 2008.

[8] A. Aiken, "Moss: A system for detecting software similarity." [Online]. Available: http://theory.stanford.edu/ aiken/moss/

[9] S. Schleimer, D. S. Wilkerson, and A. Aiken, "Winnowing: Local Algorithms for Document Fingerprinting," in Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, New York, NY, USA, 2003, pp. 76–85.

[10] M. S. Charikar, "Similarity Estimation Techniques from Rounding Algorithms," in Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing, New York, NY, USA, 2002, pp. 380–388.

[11] G. S. Manku, A. Jain, and A. Das Sarma, "Detecting Near-duplicates for Web Crawling," in Proceedings of the 16th International Conference on World Wide Web, New York, NY, USA, 2007, pp. 141–150.

[12] C. Yann, "xxhash - extremely fast non-cryptographic hash algorithm." [Online]. Available: http://cyan4973.github.io/xxHash/

[13] A. Newell, "Perceptrons. An Introduction to Computational Geometry. Marvin Minsky and Seymour Papert. M.I.T. Press, Cambridge, Mass., 1969. vi + 258 pp., illus. Cloth, $12; paper, $4.95," Science, vol. 165, no. 3895, pp. 780–782, Aug. 1969.