Mining Message Flows from System-on-Chip Execution Traces

Md Rubel Ahmed
Advisor: Hao Zheng
U of South Florida, Tampa, FL
mdrubelahmed@usf.edu, (+1) 813-570-5540

***Abstract*—System-on-Chip (SoC) validation is a costly job in the hardware design life cycle. Even with cutting-edge design automation tools, these validation and verification tasks consume 60-70% of the design time. According to our industry practitioners [1], *concurrent events and interleavings of protocol executions are major sources of design and runtime errors encountered during pre- and post-silicon validation.* One of the many challenges of SoC validation is the lack of comprehensive message flow specifications. I have worked with Intel Labs on developing methods for mining message flow specifications as my Ph.D. dissertation. My works include building simulation models of complex SoCs, capturing communication traces under different execution scenarios, and developing methods to mine meaningful and interesting flows from the execution traces. This mining work has been explored from different perspectives, such as sequential pattern mining, execution model generation, and sequence modeling. The following paragraphs describe my contributions to solving specification mining problems.**

A. *SoC Message Flow Mining*

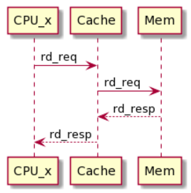
Background. System-level protocols are often represented as *message flows* in system architecture documents. Figure 1 shows a very simple yet representative example of message flows for a multi-core design. This example specifies memory read operations for two CPUs via a shared cache. A message flow specifies temporal relations for a set of messages. As shown in Figure 1(a), each message is a triple $(\texttt{src} : \texttt{dest} : \texttt{cmd})$ where the `src` denotes the originating component of the message while `dest` denotes the receiving component of the message. Field `cmd` denotes the operation to be performed at `dest`. For example, message $(\texttt{CPU0} : \texttt{Cache} : \texttt{rd\_req})$ is the read request from `CPU0` to `Cache`. During the execution of a system design, instances of flows are executed concurrently. Typically, multiple instances of different flows executed concurrently are captured in the traces. Below is an example trace, where each integer represents the corresponding message as in Figure 1(a).

$$(1, 3, 5, 6, 4, 2, 3, 1, 5, 6, 2, 4). \tag{1}$$

Flow mining as sequential patterns. SoC traces are sequential in nature. Therefore, I explored sequential pattern mining approaches as a first step. The highly concurrent nature of SoC traces makes mining more challenging than traditional pattern mining tasks. I found that many state-of-the-art sequential pattern mining tools [2] have one or both issues: 1) exceptionally high run time and 2) a large

```
1 (cpu0:cache:rd_req)
2 (cache:cpu0:rd_resp)
3 (cpu1:cache:rd_req)
4 (cache:cpu1:rd_resp)
5 (cache:mem:rd_req)
6 (mem:cache:rd_resp)
```

(a)



(b)

Figure 1. CPU downstream read flows. (a) Definitions of messages, (b) Message sequence diagram for the flows. This diagram is parameterized with x which can be 0 or 1.

number of patterns as output. The second issue is specifically alarming, as too many patterns can make mining fruitless. Therefore, I incorporated an assertion mining approach to find highly correlated patterns [3]. These smaller patterns then chain into longer sequential patterns to form specifications via inference rules. Additional optimization techniques are also incorporated to leverage the domain-specific heuristic for mining high-quality specifications [4]. A complex and realistic SoC model is also developed to evaluate the proposed mining methods, as commercial SoC traces are often unavailable. The quality of mined patterns is evaluated on two widely used metrics called *Precision* and *Recall*. However, our proposed sequential pattern mining approach suffers from high type I error for low threshold on sequential patterns. We find that the concurrency in the traces causes many false correlations among the flow events. Therefore, we feel the necessity of modeling the executions in terms of a Finite State Machine (FSM) to enhance the understanding of event correlations.

Model synthesis from SoC traces. Finite state models (FSM) extracted from traces is an industry-preferred method for understanding the internals of software and hardware executions. The synthesized models can characterize the system-level communication protocols that govern how blocks exchange messages and coordinate with each other to realize various system functions. In this work [5], we formulate specification mining as a constraint satisfaction (sat) problem. We utilize a widely used satisfiability modulo theories (SMT) solver to solve that sat problem. SMT solvers accept a set of variables and their respective constraints and objectives and generate a set of solutions. An FSM model is then constructed from the solutions produced by the solver. In practice, model synthesis from SoC traces obtained from highly concurrent executions of complex system designs is

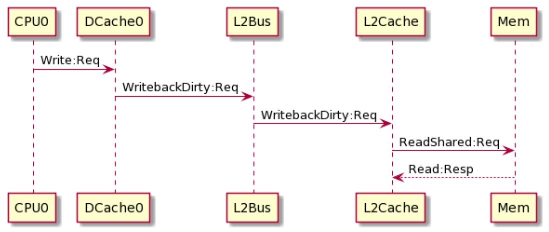


Figure 2. An example of message sequences for dirty memory block write-back operations from the extracted FSA model from gem5 traces by [5].

very challenging. Our proposed method infers reduced and abstract models from such traces and show that it can infer models efficiently in a reasonable time. Our evaluation on a realistic SoC modeled in gem5 shows that inferred models can provide significant information about system transactions such as in Figure 2. However, specification mining still needs more effective methods focusing on events' causality.

Self-Attention for Flow Mining. The flow specification can be treated as the grammar for the blocks inside an SoC to communicate with each other for realization of system-level tasks. We investigate language models to discover the underlying grammar or specification of the traces. We have investigated state-of-the-art language models (LM) called BERT [6] for SoC trace mining. In our method as presented in Figure 3, a causality graph is first built from all messages present in the input traces. This causality graph represents all possible causality relations among the messages, many of which can be invalid. A critical task of our method is to identify and remove these invalid causality relations. To facilitate the learning of LMs, we have proposed effective training methods for SoC traces [7]. Our experiments shows that attention is a powerful mechanism to discover hidden context among traces, but comes with a high cost of training such models. We show that our proposed method can efficiently remove invalid dependencies introduced in the traces due to the sharing of hardware blocks and interleaving in SoC executions.

I have shown that message flow mining can achieve the best results using sequential pattern mining and NLP algorithms.

B. Machine Learning for HLS Design Acceleration

We have worked with re-configurable systems to explore potential SoC models to be used as test beds for the specification mining methods. We find that design space exploration (DSE) for re-configurable systems is a rich avenue of research. Design parameter space creates a multi-object optimization problem among conflicting interests which has been solved by many traditional statistical and machine learning approaches [8]. Traditional methods strive to reach the optimal Pareto front by optimizing loss functions, which can take hours to days for a relatively small hardware design. I worked on a Quantum Neural Network (QNN) based early failure prediction method that solves three issues with the existing methods. First, it evaluates a trail quality and skips the synthesis step by predicting the corresponding loss beforehand. Second, it

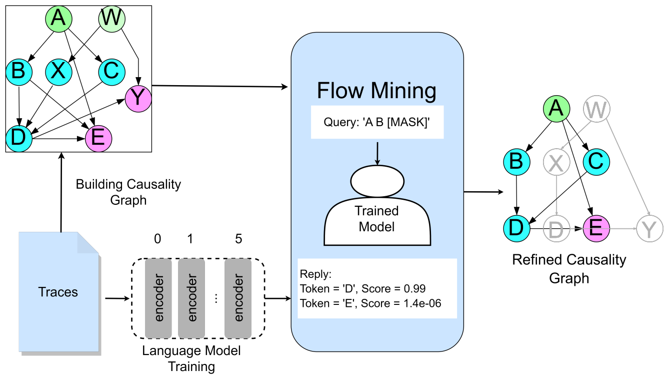


Figure 3. The overview of our method that uses a language model for mining message flow specifications from SoC execution traces. Messages from traces are used to build the causality graph. A language understanding model is trained on the traces. The causality graph is then refined with the help of a trained model to form the message flow specification.

does not require a large training set to generalize, therefore easy to fine-tune, and suitable for applications where large training samples are scarce. Third, the resulting model has a relatively small set of weights, therefore are relatively less computationally expensive, greener machine learning model. Our proposed early failure prediction method shows 20-70x speed up in DSE for several hardware design.

C. SRF@ASP-DAC 2023 Specifics

I haven't presented this summery of my dissertation to any other premier. I plan to graduate by July 2023. The header contains my contact information. I would like to request the organizer to consider me as a travel support recipient.

References

[1] Priyadarsan Patra. On the cusp of a validation wall. *IEEE Design & Test of Computers*, 24(2):193–196, 2007.
[2] Md Rubel Ahmed, Hao Zheng, Parijat Mukherjee, Mahesh C. Ketkar, and Jin Yang. A comparative study of specification mining methods for soc communication traces. In *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 31–36, 2021.
[3] Md Rubel Ahmed, Hao Zheng, Parijat Mukherjee, Mahesh C. Ketkar, and Jin Yang. Mining patterns from concurrent execution traces. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 41(8):2758–2762, 2022.
[4] Md Rubel Ahmed, Hao Zheng, Parijat Mukherjee, Mahesh C. Ketkar, and Jin Yang. Mining message flows from system-on-chip execution traces. In *2021 22nd International Symposium on Quality Electronic Design (ISQED)*, pages 374–380, 2021.
[5] Hao Zheng, Md Rubel Ahmed, Parijat Mukherjee, Mahesh C. Ketkar, and Jin Yang. Model synthesis for communication traces of system designs. In *2021 IEEE 39th International Conference on Computer Design (ICCD)*, pages 492–499, 2021.
[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.
[7] Md Rubel Ahmed, Bardia Nadimi, and Hao Zheng. Mining soc message flows with attention model. *arXiv preprint arXiv:2209.07929*, 2022.
[8] Guyue Huang, Jingbo Hu, Yifan He, Jialong Liu, Mingyuan Ma, Zhaoyang Shen, Juejian Wu, Yuanfan Xu, Hengrui Zhang, Kai Zhong, Xuefei Ning, Yuzhe Ma, Haoyu Yang, Bei Yu, Huazhong Yang, and Yu Wang. Machine learning for electronic design automation: A survey. *ACM Transactions on Design Automation of Electronic Systems*, 26(5):1–46, Jun 2021.