Parallel Stimulus Generation Based on Model Checking for Coherence Protocol Verification

Kang Zhao and Wenbo Shen

***Abstract*— The complexity of the multicore communication protocols makes it a huge effort to validate the corresponding register transfer level (RTL). To achieve the high coverage of simulation, this brief proposes a covalidation method to generate the RTL testbench based on the model-checking technique. An object-oriented event-mapping technique is proposed to transform the sequential traces created by formal method to parallel RTL stimulus. A case study on the modified, exclusive, shared and invalid protocol was performed and showed that the covalidation method could save significant effort to create RTL testbenches while maintaining high coverage.**

***Index Terms*— Formal verification, model checking, system testing.**

I. Introduction

To design a multiprocessor system, it is a critical task how to verify its coherence protocol. It involves proving that a protocol specification is following a desired memory consistency model, such as sequential consistency. Two verification methods are commonly used: 1) formal verification and 2) simulated verification. The common formal method to verify the correctness of protocols is the model checking, which examines all the possible states of the systems. It is a holistic and exhaustive solution. However, the construction process needs more designer experience. The simulation-based verification is developed widely and it is black-box. The designers develop a set of stimulus to test the target system. The correctness of the system depends on the quality of the stimulus set. The advantage is that it does not require another functional model instead of the real system. However, this method cannot ensure adequate coverage.

Top–down flow is the general choice to implement a multiprocessor system. Designers first build a functional model to verify the key behaviors, such as cache coherence. Model checking is usually the choice in this step. Although no hardware (HW) details are touched, this step is still important to reduce the design cycle. Then, the register transfer level (RTL) is implemented and tested based on the simulation. However, designers often face with three problems. First, how to achieve higher coverage for simulation-based testing? Here, we are not aiming at the circuit coverage as traditional testing does, but rather at the coverage of essential behaviors as the design specification requires. Second, how to ensure the equivalency between functional formal model and implemented HW (RTL)? Third, because the formal model has no implementation details and it checks only the key behaviors, it is equal to an abstracted subset of the RTL. That introduces the question of how to ensure the correctness of the key behaviors in the RTL system. To deal with the problems, this brief will integrate the formal verification and simulation. The strategy is to generate stimulus from the formal model traces, and the stimulus is used as the inputs for RTL simulation. The model checking is usually used to enumerate all the possible traffic in a finite state machine (FSM). The parallel traffic is represented with different enumeration orders in the log of model checking. Its output trace has the software (SW) feature, which is sequential. Due to the gap between SW and HW, our focus is on how to get parallel stimulus (HW) from sequential traces (SW). As a case study, this brief proposes a covalidation method applied to verify the cache coherence of a real multicore system. The stimulus is generated based on formal verification results and achieve high coverage. To fill the HW/SW gap during stimulus generation, an object-oriented event-mapping technique is proposed.

The rest of this brief is organized as follows. Section II discusses the related work. Section III presents the problem. Section IV describes the details of the covalidation system. Section V describes the mapping technique used in covalidation. Section VI presents the experimental results. Section VII draws the conclusion.

Manuscript received October 28, 2013; revised August 11, 2014 and October 14, 2014; accepted December 14, 2014. Date of publication January 14, 2015; date of current version November 20, 2015. This work was supported in part by Intel Labs, Shanghai, China, in part by the National Natural Science Foundation of China under Grant 61106030, and in part by the Tsinghua University Initiative Scientific Research Program.

K. Zhao is with Intel Corporation, Beijing 100013, China, and also with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: kang.zhao@intel.com).

W. Shen is with Intel Corporation, Beijing 100013, China (e-mail: zhaokang233@126.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2014.2384040

II. Related Work

Most previous works did not propose an entire integrated method of the formal-based and simulation-based verification methods for the coherence protocols.

First, some works support automatic partitioning for the target system. The target design is partitioned to several parts. Some are fit for the static formal verification, and the others for the simulation [1]–[3]. For example, Li *et al.* [1] proposed a tool for the complexity analyzer that is used during the partitioning. According to different complexities under different constraints, the design is partitioned to several parts: 1) the equivalence check; 2) the simulation verification; 3) the symbolic model checking; and 4) the property check. After different types of verification, a coverage analyzer will run and integrate the different verification results. The limitation of such method is that it uses the advantages of different verification methods but cannot avoid their disadvantages.

Second, some works use hybrid verification strategy. Random simulate generation is implemented first, and then the formal verification tool is used to inform the test bench writer missing cases only for coverage analysis [4]–[7]. For example, Martyna [7] proposed a method to fill the gap between the formal verification and simulation for the wireless sensor network. It first selected a popular test random generation algorithm and obtained the test bench. Then the simulation results were measured using coverage analysis based on the formal verification model. To verify the missing cases, the simulate generator is improved and then better coverage results are obtained. The limitation of such method is its long verification period. During the testing process, the formal verification is used only as the assistant approach to improve the simulation results. If the simulation totally depends on the formal verification, this verification period can be reduced.

Furthermore, Singh and Landis [8] proposed a real covalidation method that integrates the formal verification and simulation. This method is based on the model checking. It lists all the possible cases of the modified, shared and invalid protocol manually, and obtains

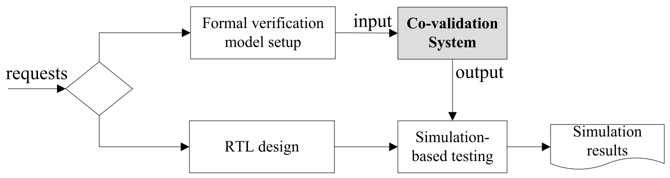


Fig. 1. Covalidation bridges the formal verification and simulation.

the test bench based on the enumeration for all state combinations. However, the target protocol is simple, and so the test bench is summarized manually. Such an approach is not fit for the complex multicore system. In addition, Goldberg [9] used satisfiability (SAT) technique and bridged the simulation with formal verification. Its strategy is to get the test set through SAT instead of random sample generation. However, the coverage quality cannot be guaranteed.

To address this issue, this brief proposes a covalidation method. It has two contributions: 1) it can generate testbenches with higher coverage than the constrained random simulation and 2) it deals with the gap seamlessly between the sequential formal traces and the parallel RTL stimulus.

III. Problem Analysis

The nature of the covalidation process is to convert the model-checking outputs to the RTL stimulus.

In fact, the formal model is a FSM that is represented with high-level languages. It can enumerate all possible traffic in the FSM. Although the formal model describes a parallel system, its output traces are sequential. Instead, it uses different orders of enumerations to denote the parallel states. For example, the states s_0 and s_1 have no dependence and they could run in parallel. Then the output has both the traces: 1) $s_0 \rightarrow s_1$ and 2) $s_1 \rightarrow s_0$.

However, the RTL implementations cannot identify such traces. The real stimulus is described with HW description language (HDL) and it has the parallel data structure, such as fork-join (a parallel structure in Verilog). The focus is on how to achieve the seamless conversion from the sequential events to the parallel RTL operations. To clarify the mapping problem, we present a definition.

Definition 1: $f = \langle f_1, f_2, \ldots, f_m \rangle$ denotes a transaction, which includes a sequence of events from f_1 to f_m; $s = \langle s_1, s_2, \ldots, s_n \rangle$ denotes a list of RTL operations, from s_1 to s_n. $F = \{f\}$ denotes the formal verification results, and $S = \{s\}$ denotes the simulation inputs.

The focus is on the mapping and translation from F to S. Since the model checking output is sequential representation, the neighboring events in f may run in parallel or in sequence when running in the real HW. As a result, each event in f may be mapped to more than one task. Next we build the covalidation system and propose an object-oriented mapping (OOM) method to achieve the seamless conversion.

IV. Covalidation System

Fig. 1 shows the covalidation system. Its input is the formal verification result, and its output is the set of stimulus for the RTL validation. The formal verification uses a systematically exhaustive exploration based on the model-checking technique, which can cover all possible functional traces. This covalidation system is applicable to any protocol validation for the concurrent platforms. It includes three steps.

1) The high-level model is used to monitor and extract the high-level checkpoints from the model-checking results (Fig. 2). The high-level checkpoints are defined as key behaviors that are related to the RTL stimuli. They are collected with a format of breath-first-search (BFS) tree by the monitor. Each checkpoint corresponds to one state, and each state includes all predefined variables with different values. Then all the cases from root to leaf in this BFS tree is extracted.
2) The translator is used to translate the sequential high-level traces to the parallel RTL traces, as shown in Fig. 3. The high-level model is based on SW that is sequential with no timing information, and the RTL is based on HW with cycle-accurate information. So the key is how to achieve a correct mapping. In the translator, an OOM method is proposed to resolve this problem. The details are presented in Section V.
3) The synthesizer is used to capture the testing events from the translator. It implements corresponding RTL accurate-cycle tasks, as shown in Fig. 4. First, a template library is built up corresponding to the key behaviors defined during the model checking. Then the translator results are merged with the templates for the simulation. The transition coverage is mainly for RTL testing, and the functional coverage is for the model checking. These two coverage are different. Because the model checking can check only the functional key behaviors, the generated RTL test bench can achieve high functional coverage for those key behaviors.

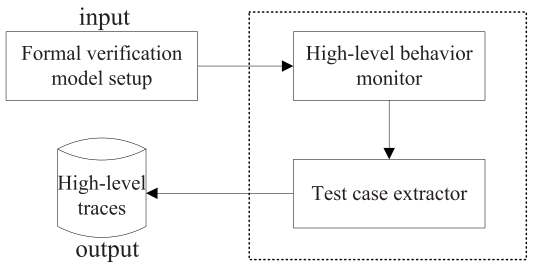


Fig. 2. High-level model.

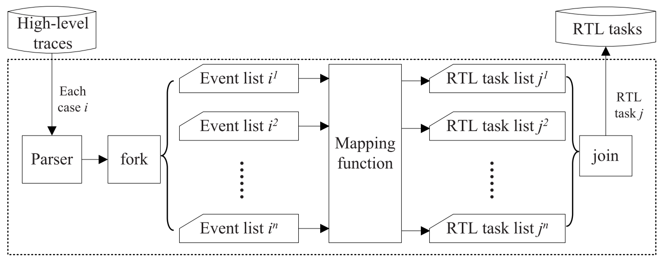


Fig. 3. Translator implementation.

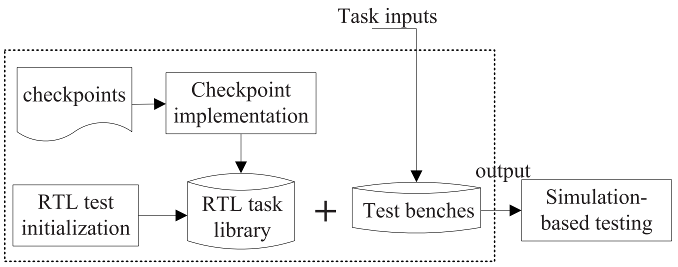


Fig. 4. Synthesizer implementation.

We have implemented the covalidation system (Fig. 5). First, the model checking is built using Murphi 5.4.9 [11]. All possible states are represented with the BFS format and then each test case is extracted. Second, the translator analyzes each coherent checkpoint and they map to the corresponding RTL function calls. Third, we have implemented all the RTL function templates corresponding to the coherent checkpoints. Based on the template library, the test bench is generated.

In addition, there is a limitation for this system. Because the generated RTL test bench relies on the formal model, the bugs in

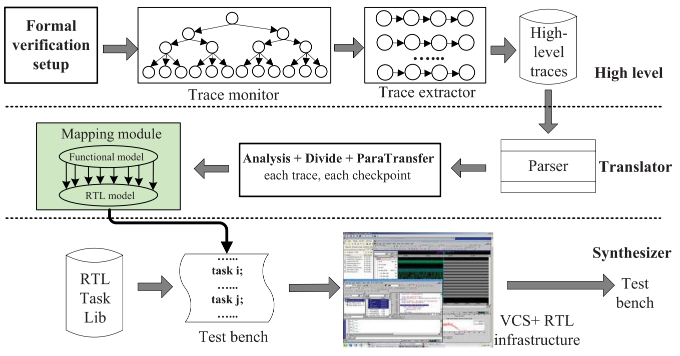


Fig. 5. Covalidation system implementation.

the formal model will transfer to the final results. As a result, the correctness of the formal model is the precondition.

V. Object-Oriented Mapping Technique

This section presents the OOM technique in the translator (Fig. 3). The motivation is to deal with the key differences between the high-level traces from the formal model and the RTL function calls. First, the high-level trace is sequential that uses different enumeration orders to represent the parallel behavior, but the RTL function call runs directly in parallel. Second, the high-level list uses sequence order instead of timing information, but RTL list has the real timing information. Therefore, the key is how to implement a seamless translation from the formal model to RTL.

The general method to translate the sequential list to the parallel RTL is the high-level synthesis (HLS) [13]. Its input is the C/C++/SystemC program, and its output is the Verilog/VHDL program. The traditional techniques of HLS are to extract the data flow dependency of C inputs, bind each operation to the limited HW units, and finally schedule those operations under multiple constraints. However, the HLS technique is complex and it is still not popularized to the market. Furthermore, our problem is much simpler than the traditional HLS. First, the sequential traces have no complex data dependencies like C/C++; instead, the dependence only exists for each interface. In addition, the generated parallel lists are much simpler than the HLS output. Since our focus is on the functional coherence checking, the key is the running order of different tasks. Therefore, a simple but efficient mapping method is proposed instead.

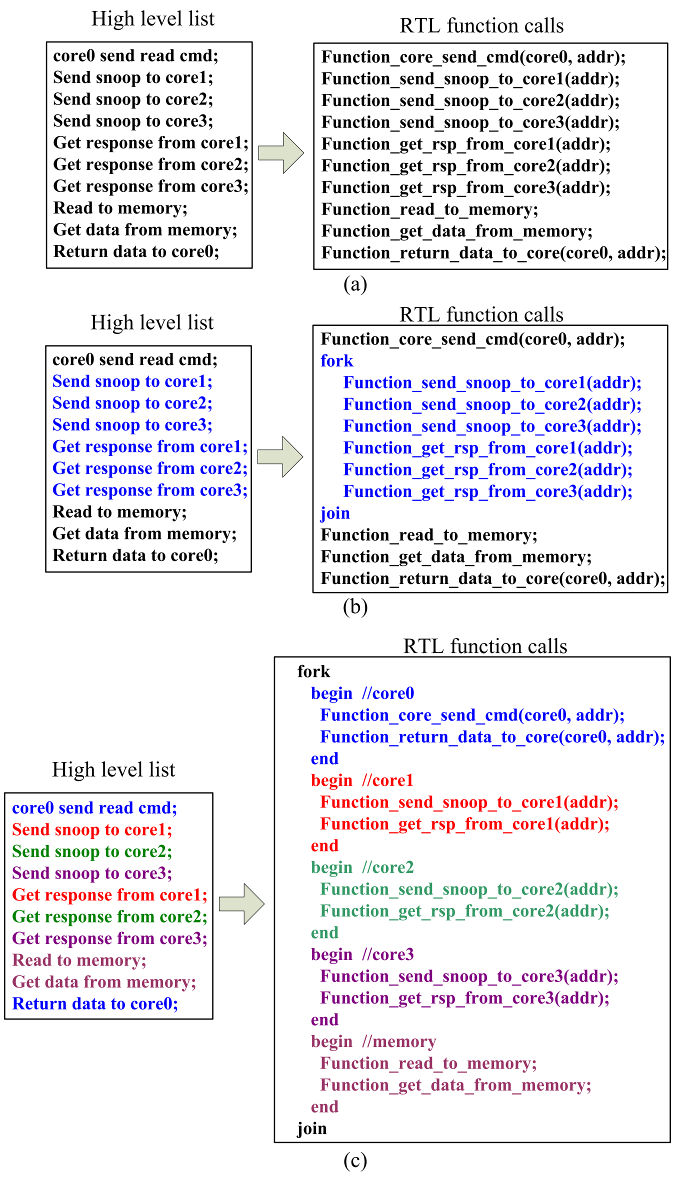
The OOM method is proposed to resolve this problem. Its strategy is to ensure the sequential orders based on each interface of the multiprocessor system. The behaviors between different interfaces can be guaranteed by the model-checking rules. So the behaviors between different interfaces are not considered in OOM. Here, the interface means the coherence module of the multiprocessor, which is used as different carrier during the formal model, such as L1 cache, L2, and memory. Its correctness can be guaranteed by the following theorems.

Theorem 1: The events during one interface run in sequential; and the events between different interfaces run in parallel.

Proof: It is obvious that all events in one interface must run in sequential, because the interface is only one. In multiprocessor system, the data communication between different interfaces has no conflict and it can run in parallel. For example, L2 and memory can send data to each other in parallel. So the events between different interfaces can run in parallel.

Theorem 2: The model-checking process can guarantee the running orders between different interfaces.

Proof: This feature can be proved by the model-checking definition. For each two events running in parallel between


(a)

High level list
```
core0 send read cmd;
Send snoop to core1;
Send snoop to core2;
Send snoop to core3;
Get response from core1;
Get response from core2;
Get response from core3;
Read to memory;
Get data from memory;
Return data to core0;
```
RTL function calls
```
Function_core_send_cmd(core0, addr);
Function_send_snoop_to_core1(addr);
Function_send_snoop_to_core2(addr);
Function_send_snoop_to_core3(addr);
Function_get_rsp_from_core1(addr);
Function_get_rsp_from_core2(addr);
Function_get_rsp_from_core3(addr);
Function_read_to_memory;
Function_get_data_from_memory;
Function_return_data_to_core(core0, addr);
```

(b)

High level list
```
core0 send read cmd;
Send snoop to core1;
Send snoop to core2;
Send snoop to core3;
Get response from core1;
Get response from core2;
Get response from core3;
Read to memory;
Get data from memory;
Return data to core0;
```
RTL function calls
```
Function_core_send_cmd(core0, addr);
fork
   Function_send_snoop_to_core1(addr);
   Function_send_snoop_to_core2(addr);
   Function_send_snoop_to_core3(addr);
   Function_get_rsp_from_core1(addr);
   Function_get_rsp_from_core2(addr);
   Function_get_rsp_from_core3(addr);
join
Function_read_to_memory;
Function_get_data_from_memory;
Function_return_data_to_core(core0, addr);
```

(c)

High level list
```
core0 send read cmd;
Send snoop to core1;
Send snoop to core2;
Send snoop to core3;
Get response from core1;
Get response from core2;
Get response from core3;
Read to memory;
Get data from memory;
Return data to core0;
```
RTL function calls
```
fork
  begin //core0
    Function_core_send_cmd(core0, addr);
    Function_return_data_to_core(core0, addr);
  end
  begin //core1
    Function_send_snoop_to_core1(addr);
    Function_get_rsp_from_core1(addr);
  end
  begin //core2
    Function_send_snoop_to_core2(addr);
    Function_get_rsp_from_core2(addr);
  end
  begin //core3
    Function_send_snoop_to_core3(addr);
    Function_get_rsp_from_core3(addr);
  end
  begin //memory
    Function_read_to_memory;
    Function_get_data_from_memory;
  end
join
```

Fig. 6. Examples for different types of mapping. (a) Direct mapping. (b) Partly direct mapping. (c) OOM.

different interfaces, the model-checking results use different enumeration orders to represent parallel behaviors.

Theorem 3: If the used RTL task library is the same, the set of the generated stimuli is determinate.

Proof: Since the used RTL task library is the same, the implementation for each translated task will be the same. Then the focus is on the running orders of tasks. As the example shown in Fig. 6(c), it is obvious that the order in each segment (begin–end) is sequential, but the order between segments is parallel. So the HDL tools (such as Verilog Compile Simulator) will generate one correct stimulus following such order constraint. Furthermore, many different orders of stimuli will be generated due to the parallel feature. Different permutation and combination will be able to cover all the cases. This determination is based on lots of stimuli generation, and not only one stimulus. So we can say that the set of stimuli is determinate.

In conclusion, direct mapping cannot get parallel information. Instead, we must classify and group the events based on different interfaces. This can both ensure the sequential order for each interface, and also ensure the parallel feature between different interfaces. However, the timing information cannot be created without

```
1. define key behaviors (checkpoints);
2. FOR each high-level trace, DO
3.      identify its behavior, map to checkpoint;
4.      translate to corresponding RTL function calls;
5.      classify based on different interfaces;
6. END FOR
7. FOR each interface DO
8.      group the sequential inner function calls together;
9. END FOR
10. group all interface-based sets with fork-join (parallel);
```

Fig. 7. Flow of the OOM method.

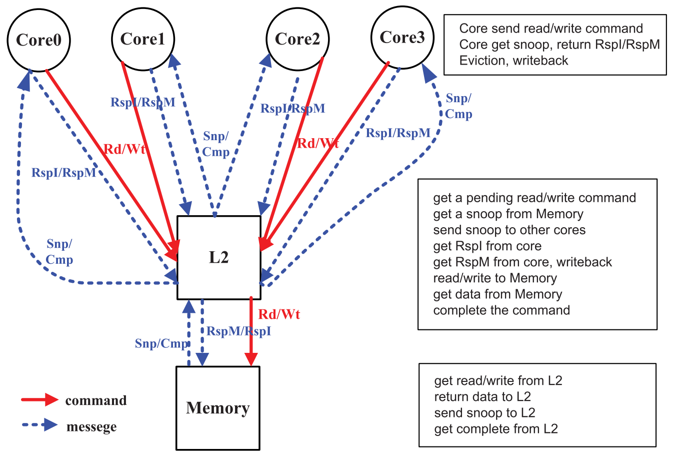


Fig. 8. Functional structure and coherence checkpoints.

foundation. In fact, the timing information is predefined in the template library for the synthesizer according to the target design, as shown in Fig. 4. The timing delays must be correlated with the real requirements for the target design.

The details of OOM method is shown in Fig. 7. After defining the checkpoints (line 1), each high-level trace from the formal model is identified. What is more, they are classified with different types corresponding to different interfaces (line 5) and grouped together with begin–end structure (line 8). Since the behaviors between different interfaces are in parallel, the interface-based sets are grouped with fork-join structure. The coherence is guaranteed for each interface.

This method has a limitation. Because the mapping process uses the fork-join HDL structure, there are repeating test cases. To reduce the redundant cases, the heuristic symmetry reduction technique is used when generating traces with Murphi [10]. The details are omitted due to space limitation.

VI. Case Study

The testing target is a multicore system. This system includes the Qemu simulator and the chipset components which have been implemented on the Synopsys HAPS 51T emulation board. In this system, the READ/WRITE commands are launched by the Qemu simulator, and transferred to the chipset through the traffic dispatcher and PCIe; the generated snoop signals will be sent back to the Qemu in the opposite direction. The cache coherence protocol for this system is modified, exclusive, shared and invalid (MESI) which is commonly used for write-back caches.

We have implemented the formal model and the OOM method, as shown in Fig. 8. It includes three interfaces: 1) cores (L1 cache); 2) L2; and 3) memory. The checkpoints include READ/WRITE command, snoop (Snp), response invalid or modified (RspI/RspM), and completion (Cmp) messages. The formal model based on model checking can explore all the possible cases for the MESI protocol, and generate different types of coherence event lists (as the high-level list shown in Fig. 6).

TABLE I
Results for the Formal Model and Generated Testing Benches

Formal model		Co-validation system	
Variable storage	596 bits	Address space	2
Possible state number	2^{596}	Mode	write-back
Storage ratio	$4890205/2^{596}$	Multi-cmd	Parallel
Explored BFS level	108	Test-bench space	8,644,116
Model checking time	about 250min	time/case	$\leq$10sec

Since RTL running is in parallel, it may not be under the control of the formal model traces if the translator uses a simple translation rule. As shown in Fig. 6(a), after the read command, the order of snoops and the responses may not be completely same as the formal model orders. For example, the order may be: snoop1 $\rightarrow$ snoop2 $\rightarrow$ snoop3 $\rightarrow$ response1 $\rightarrow$ response2 $\rightarrow$ response3; but the order may also be: snoop1 $\rightarrow$ snoop3 $\rightarrow$ snoop2 $\rightarrow$ response2 $\rightarrow$ response1 $\rightarrow$ response3. If the order in the HW is different from the formal results, it will be far from our motivation. If we use Fig. 6(b) instead, it is still not the best choice especially for the complex multitransaction cases. Therefore, the mapping results using OOM in Fig. 6(c) can ensure both sequential order in each interface and also the parallel feature between different interfaces.

The statistical results are summarized in Table I, which contains the statistical information of both the formal model and the covalidation system. For the formal model, the variable storage is the definition of the model checking, which denotes the maximum representation space of the variable set; so the possible state number is the exponential result of the former value. For the covalidation system, we test only the multicommand in parallel with two different addresses. It means that the cases of conflict addresses can be modeled. The experimental results indicate that the formal-based stimulus generation method is feasible for the coherence protocol verification.

Since most related works are not real integration of the model checking and the RTL simulation, the experimental comparison is difficult. We select [8] as the target, because it is the first real covalidation method and the test bench is also obtained via formal verification. We have implemented the two methods proposed in [8]. The transaction count is selected as 200, and the number of cache lines is 10. In addition, the method proposed in [8] cannot support complex protocols, because it did not consider the event classification based on interfaces. So we build its translator and FSM for the MESI protocol by manual. The NorthBridge chipset implementation for the Intel Quark Clanton processor is selected as the testing circuit. Its input commands are generated from a Qemu simulator, and the motivation of using this simulator is to upgrade core numbers conveniently. The testing core numbers change from 2 to 12.

Fig. 9 presents the comparison of the transition coverage. It is related to the percentage of test vectors generated automatically. Both the directed random method and the constrained random method are proposed in [8]. We can see that OOM could bring relatively stable transition coverage, with about 2%–5% fall when the core number increases; the worst case is the directed random method, which brings about 5%–25% fall when the core number increases. Furthermore, OOM can get higher transition coverage for most cases, about 10% (maximum) higher than the constrained random method and 30% (maximum) higher than the directed random method. In addition, Fig. 9 shows that the transition coverage for the two-core case is a little lower than the two methods in [8]. Since the

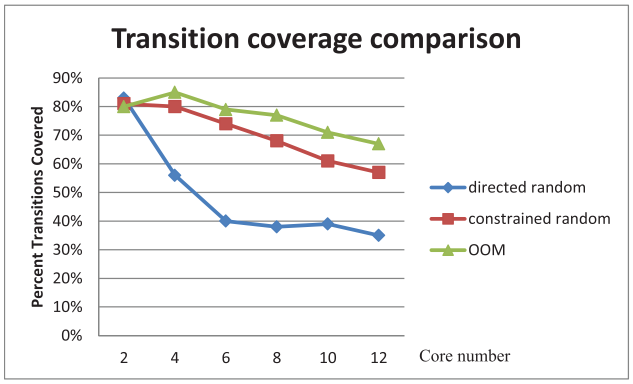


Fig. 9. Transition coverage compared with the two methods in [8].

two-core structure is the simplest case, the random techniques might be able to cover more branches in the FSM, we suppose.

Except for the coverage, the advantage of OOM includes other two points: 1) lower cost and 2) more protocol support. First, the two methods proposed in [8] cannot do the translation from the sequential traces to the parallel traces. Instead, it requires the redundant manual translation. This is also an important reason that the two methods in [8] cannot support complex protocols. OOM has implemented an automated translator, which can bring a lower cost than the methods in [8]. Second, OOM can support more complex protocols than the methods in [8]. The reason is the same. Therefore, OOM should be a good choice during the covalidation.

VII. Conclusion

This brief proposes a covalidation system to generate the parallel stimulus for the multicore coherence verification. To achieve a high coverage, the formal-based verification is integrated with the simulation seamlessly. The OOM method is proposed to translate sequence to parallelism. In future, we will focus on how to reduce the huge state space and save the missing coverage when bug exists.

References

[1] L. Li, M. A. Thornton, and S. A. Szygenda, "Integrated design validation: Combining simulation and formal verification for digital integrated circuits," *J. Syst., Cybern., Inf.*, vol. 4, no. 2, pp. 22–30, 2006.

[2] J. Jose and S. A. Basheer, "A comparison of assertion based formal verification with coverage driven constrained random simulation, experience on a legacy IP," Wipro Technol., Bengaluru, India, Tech. Rep. 18353, 2007.

[3] E. Segev, S. Goldshlager, H. Miller, O. Shua, O. Sher, and S. Greenberg, "Evaluating and comparing simulation verification vs. formal verification approach on block level design," in *Proc. 11th IEEE Int. Conf. Electron., Circuits, Syst.*, Dec. 2004, pp. 515–518.

[4] G. D. Cunningham, P. B. Jackson, and J. A. B. Dines, "Expression coverability analysis: Improving code coverage with model checking," in *Proc. Design Verification Conf.*, San Jose, CA, USA, Mar. 2004, pp. 1–8.

[5] K. Shimizu and D. L. Dill, "Deriving a simulation input generator and a coverage metric from a formal specification," in *Proc. 39th Annu. Design Autom. Conf. (DAC)*, New Orleans, LA, USA, 2002, pp. 801–806.

[6] S. Tasiran, B. Batson, and Y. Yu, "Linking simulation with formal verification at a higher level," *IEEE Design Test Comput.*, vol. 21, no. 6, pp. 472–482, Nov. 2004.

[7] J. Martyna, "Linking simulation with formal verification and modeling of wireless sensor network in TLA+," in *Computer Networks* ser. Communications in Computer and Information Science, vol. 79. Berlin, Germany: Springer-Verlag, 2010, pp. 131–140.

[8] P. Singh and D. L. Landis, "Test generation for CMP designs," in *Proc. 11th Int. Workshop Microprocessor Test Verification*, Dec. 2010, pp. 67–70.

[9] E. Goldberg, "On bridging simulation and formal verification," in *Proc. 9th Int. Conf. Verification, Model Checking, Abstract Interpretation*, 2008, pp. 127–141.

[10] C. Norris Ip and D. L. Dill, "Better verification through symmetry," *Formal Methods Syst. Design*, vol. 9, nos. 1–2, pp. 41–75, 1996.

[11] *CMurphi*. [Online]. Available: http://mclab.di.uniroma1.it/site/index.php/software/18-cmurphi, accessed Nov. 10, 2014.

[12] *Synopsys Inc.* [Online]. Available: http://www.synopsys.com, accessed Nov. 10, 2014.

[13] P. Coussy, *High Level Synthesis—From Algorithm to Digital Circuit*. Amsterdam, The Netherlands: Springer-Verlag, 2008.