VFCC: A Verification Framework of Cache Coherence using Parallel Simulation

Qiaoli Xiong, Jiangfang Yi, Tianbao Song, Zichao Xie, Dong Tong

Microprocessor Research & Development Center, School of Electronics Engineering and Computer Science,
Peking University, Beijing, China, 100871
Tel : 010-6275-9129, Fax : 010-6275-6231
{xiongqiaoli, yijiangfang, songtianbao, xiezichao, tongdong}@mprc.pku.edu.cn

Abstract – A cache coherence protocol is a vital component of a multiprocessor to maintain the data consistency. In this paper, we proposed VFCC, which is a simulation framework to validate a cache-coherence protocol implementation of a commercial 64-bit superscalar multiprocessor. It exploits multiple-level parallelism to accelerate validation without overheads among threads. Our experimental results demonstrate VFCC has a 5.0x speedup than a traditional simulator on a conventional 16-core host machine.

I. Introduction

A cache coherence protocol is a vital component of a multiprocessor to maintain the data consistency in local caches and the main memory. It defines a set of rules coordinating processors, cache controllers, and memory controllers to ensure: the value returned by a read must be always the last value written to that location [1, 2]. Cache coherence protocols are notoriously difficult to design and verify [3]. Due to more and more complicated behavior of a multiprocessor, verifying the correctness of a cache coherence protocol is becoming a major task.

A survey presents three widely accepted formal approaches to verify cache coherence protocols [4]. In fact, these work focus on the protocol semantics rather than the protocol implementation. In order to ensure high performance and to accommodate manufacturing realities such as unordered coherence message networks, current protocols allow simultaneous or outstanding requests and split transactions. Therefore, the characteristics of a protocol implementation, such as parallel processing and out-of-order completion strategies, also play an important role in the correctness of execution results. Simulation is effective to validate the details of a protocol implementation [5].

In this paper, we proposed VFCC, a verification framework of a cache-coherence protocol implementation using parallel simulation. It is used to validate the RTL (Register Transfer Level) design of the MOESI protocol [6,7]. In general, a cache-coherence protocol is composed of three components: a finite set of states, a finite set of events, and a transition relation. There are more details of a cache-coherence protocol implementation, which involves interactions between the protocol implementation and the cache or the main memory, one transaction split into multiple actions, concurrent requests and so on. VFCC validates the implementation details as well as protocol semantics. VFCC is based on transaction-level [8] simulation technique. Its VIP (Verification Intellectual Property) deals with a transaction by three units: a request generator, a protocol simulator and a request checker. The units form a pipeline structure by multithread programming. VFCC also exploits multiple VIPs to simulate multiple requests from different cores concurrently. Another feature of VFCC is the parallelism between the simulator and the RTL, an overlap between simulation time and execution time. VFCC solves the problems of communication and synchronization [9] for each level of parallelism without overheads. A transaction-level simulator is not cycle accurate. There may be inconsistence among checkpoints to be adjusted [10, 11].

The rest of this paper is organized as follows. Section 2 discusses related work. VFCC description comes next in Section 3. Section 4 shows the experiments and results. Section 5 is our conclusion.

II. Related Work

In the last twenty years, some methods have been proposed to verify the correctness of a coherence protocol. A survey has shown that there are three major techniques based on state enumeration, (symbolic) model checking and symbolic state model [4]. These techniques characterize the protocol as some states and search all reachable states exhaustively. Moreover, some recent approaches [12-14] have extended one of these methodologies to verify more complex protocols, like adaptive or hierarchical protocol or directory-based protocols [15, 16]. In fact, these techniques prove correctness of a protocol specification by focusing on the protocol semantics without any consideration of architectural behavior.

Dynamic verification approach has been proposed recently [17, 18]. It is a powerful error detection mechanism, but the existing coherence checkers are costly to implement. Another effective method to validate RTL design is simulation. Worawan and Roland address three properties of a coherence protocol (safety, liveness and inclusion). They propose a specification-based parameter–model interaction (SPMI) technique to detect these cases in a particular DSM cluster simulator [19]. DSiMCluster is a sequential simulation model that runs parallel workloads by multithread interleaving to emulate a multithreaded runtime environment. Simulation time and accuracy are two key factors.

978-1-4673-3030-5/13/$31.00 ©2013 IEEE

Considering to the two features (concurrent and out-of-order requests) of a protocol implementation, parallel simulation is a powerful method to speed up validation and to ensure accuracy by simulating concurrent requests.

Simulation technology focuses on three aspects: stimuli generation, simulation, and results checking. Stimuli can be generated statically or dynamically. Static approach prepares a large number of stimuli before simulation. This approach requires both generation time and storage space. Dynamic approach disperses stimuli generation throughout the simulation process to get an overlap. It also saves storage.

There are sequential simulation technique and parallel simulation technique. By sequential simulation we mean that all functional modules (a generator, a simulator and a checker) from a framework work in time-sharing way. It is not suitable for a parallel design by interleaving, so sequential simulation has a verification-space limitation [20]. Parallel simulation employs multithread programming. It can effectively construct a stimuli scenario for a design with parallel feature, but it faces the difficulty of communication and synchronization.

According to the granularity of checking cycle, simulators can be cycle-accurate or checkpoint-accurate. Fig. 1(a) shows cycle-accurate simulation. It is suitable for a simulator which has to remain consistent with the design very cycle, like the pipeline structure of a processor. However, it has a lot overhead. Fig. 1(b) shows checkpoint-accurate simulation. It is suitable for a design focusing on transaction, like cache structure whose data or status to be changed at the end of a request. This approach avoids unnecessary overhead.

There are four kinds of simulators according to the combination of stimuli generation (static or dynamic) and simulation method (sequential or parallel): a static-sequential simulator, a static-parallel simulator, a dynamic-sequential simulator and a dynamic-parallel simulator. Fig. 2(a) and Fig. 2(b) show the static generation method. It costs a lot of time to prepare the stimulus. Fig. 2(c) and Fig. 2(d) show the dynamic generation method. The overlap advantage of the method will be more obvious as the number of stimuli grows. Fig. 2(a) and Fig. 2(c) show sequential simulation method. There are two drawbacks. The first one is that it takes longer to generate the stimuli one by one. The second one is that it fails to verify the concurrent characteristic of the RTL. Fig. 2(b) and Fig. 2(d) show parallel simulation. This approach provides the situation of concurrent multiple requests. Fig. 2(d) shows the way VFCC works in. The description of its implementation details is in the following parts.

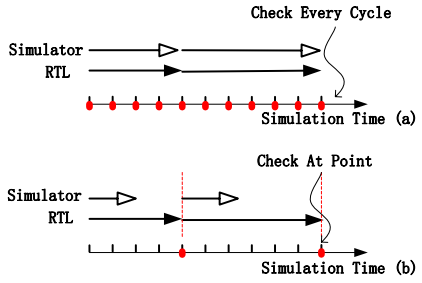

Fig. 1 Cycle-accurate and checkpoint-accurate.

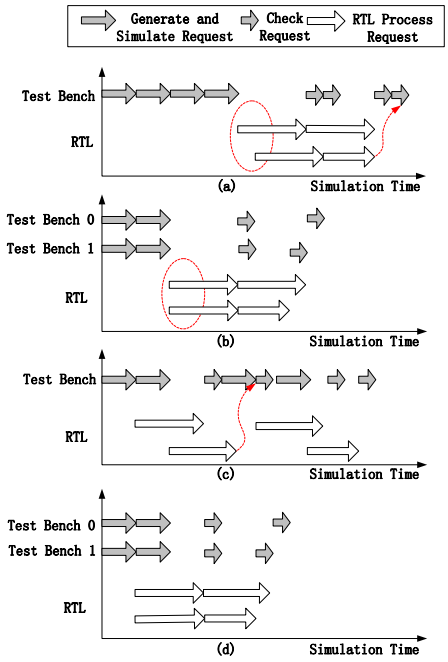

Fig. 2 Comparison among static-sequential simulation(a), static-parallel simulation(b), dynamic-sequential simulation(c) and dynamic-parallel simulation(d).

III. VFCC: Verification Framework of Cache Coherence

We designed a cache coherence protocol component for a commercial 64-bit superscalar quad-core multiprocessor. It employs MOESI protocol based on bus snoopy mechanism. It supports 10 concurrent requests and out-of-order requests completion strategies. Our goal is to fast validate the RTL implementation of the coherence-protocol component.

VFCC is a verification framework for the cache-coherence protocol RTL implementation. It consists of multiple VIPs and multiple peripheral models. Each VIP consists of multiple functional units. The key part of a VIP is a cache-coherence protocol simulator. In this section, we first introduce the structure of VFCC, followed by the three aspects of VFCC: firstly, multi-level parallelism of VFCC; secondly, the solutions to deal with communication and synchronization problem; finally, the methods to keep the transaction-level simulator and the RTL design consistency.

A. Structure of VFCC

Fig. 3 shows the structure of VFCC. It consists of four parts. The key part of VFCC is a VIP. Each VIP simulates the behavior of a request from one core. A VIP consists of a request generator, a cache-coherence protocol simulator and a request checker. There are N VIPs in VFCC to simulate N requests from N cores. The second part of VFCC is a memory model. It simulates the behavior of the main memory and the controller. The third part of VFCC is N cache models. Each VIP gets one. The last part of VFCC is N request records to save its requests and simulation results. The design under test (DUT) is the RTL of protocol.

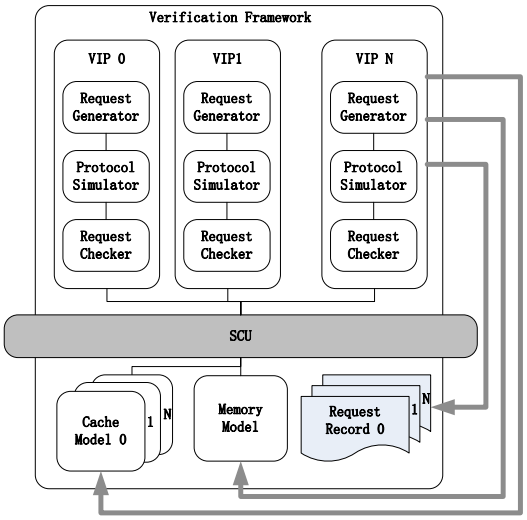

Fig. 3. Structure of VFCC.

Multiple VIPs work concurrently in the same way. In a VIP, a request generator generates a random request first. The request generator sends the stimuli to the simulator and the RTL at the same time and saves the request in a record. Then, it continues to generate a new request. We extract some constrains from the protocol to add into the generator. Therefore, the generator has a self-adjusting ability to abandon an invalid request and prepare a new one. The cache-coherence protocol simulator is a high-level reference model for the RTL. The simulator processes the same request as the RTL and saves the simulation result in the record for the checker. The checker compares the results at a check point. The simulation result and execution result refer to the changes of the cache models and the memory model.

The cache model exploits LRU (least recently used) strategy with parameterized cache size and parameterized set number. The cache model provides a variety of interfaces for the RTL and the simulator, such as snooping status, reading and writing data, updating cache line and so on.

The memory model simulates the behavior of accessing data by address as the main memory. It takes byte as a store unit. The memory model also provides interfaces of reading and writing data.

B. Multi-Level of Parallelism

VFCC utilizes multi-level parallelism to accelerate simulation. The first level parallelism of VFCC is among the three functional units in a VIP. Each unit works in an independent thread. Multiple threads execute concurrently on a multiprocessor host machine. These threads process one request in order, but they form a pipelined structure to process different requests. This mechanism increases the throughput and reduces the total verification time.

The second level parallelism of VFCC is among multiple VIPs. Each VIP works in one process. VFCC exploits parameterized number of concurrent VIPs to obtain a good scalability for RTL may execute different number of concurrent requests. Each VIP simulates and checks requests separately. This level of parallelism creates a test situation the same as a real multiprocessor system.

The last level parallelism of VFCC is the overlap between the protocol simulator and the RTL. VFCC hides the stimuli generating time by utilizing a virtual pipelined structure between the VIP and the RTL. The generator produces a new request while the RTL executes the prior one. Therefore, total verification time is the sum of the time generating the first request, the time checking the last request and the time processing all the requests by the RTL.

VFCC utilizes multiprocessor host machine by exploiting a fine-grain parallelism. The first-level parallelism reduces simulation time by a pipeline structure. The second-level parallelism promises verification accuracy by concurrent requests. The last-level parallelism accelerates simulation by overlapping simulation and execution.

C. Communication and Synchronization

Two major overheads from parallel simulation are the communication problem and the synchronization problem among multiple threads. This part introduces the solutions to reduce the overheads.

Firstly, we introduce the first level parallelism of communication and synchronization. Fig. 4 shows the data flow and the control flow among three threads in a VIP. A request is generated by the generator thread and stored in a request record for the simulator thread. The simulator thread keeps checking if a request is ready, and it processes the request if it is ready, or keeps waiting. In the simulation process, the simulator thread accesses the status and data of the cache model or the data of the memory model. Then the simulator thread saves the simulation result into the record. Meanwhile, the checker thread keeps checking if simulation finishes. The checker thread reads simulation result to compare with the cache model and the memory model.

Therefore, there is no communication among threads directly except the data flow, which is an indirect kind of communication. In VFCC, requests and simulation results are in the form of a global shared variable stored in a request record so that each thread accesses directly. Each thread accesses the cache model and the memory model through their interfaces directly, too.

The control flow is the synchronization among threads. Since multiple threads read the different ready bits to check if the request is available, it is safe for threads to access shared variable directly without synchronization mechanism.

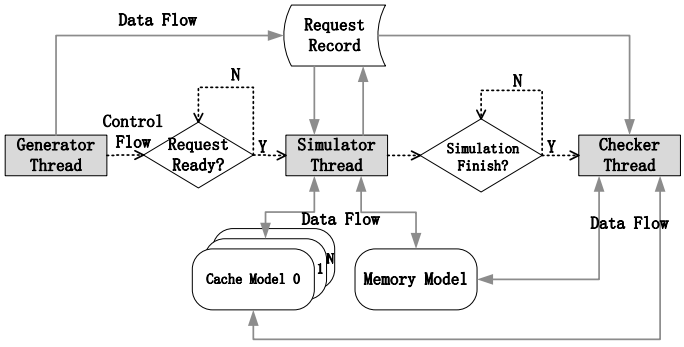

Fig. 4 Communication and synchronization in a VIP.

Secondly, we introduce how to reduce communication and synchronization among VIPs. As we know, the key responsibility of cache coherence is to keep data consistent and up-to-date to every core. In other words, for each core, its read request gets the same data as others; its write request assures that all cores can get the latest data of this address in the next read. Therefore, only requests with the same address needs to be executed in sequential, irrelevant requests are safe to be executed concurrently, which means that multiple VIPs need not to communicate with others.

The synchronization among VIPs needs to be solved when requests with the same address or address conflict. We use an address table and priority flag to keep all the requests on-fly for the simulator threads. If a simulator thread checks an address confliction in the address table and it does not own the priority flag, it stops simulation. This kind of decentralized checking mechanism reduces synchronization overheads. VFCC also optimizes synchronization among VIPs by a scheduling strategy. Fig. 5 shows two strategies to synchronization among VIPs with address conflict. Fig. 5(a) shows a blocking strategy. VIP 1 generates and simulates a request. If VIP 0 generates a request with address conflict this time, the simulator has to wait for VIP 1 until its completion. There is some performance loss for it becomes sequential simulation. Fig. 5(b) shows a scheduling strategy. If VIP 0 generates a request with address conflict, VIP 0 blocks this conflict request and schedules an irrelevant request to process. The simulator checks whether the request is still conflict after the irrelevant request completes. The simulator processes this request if it is conflict-free or schedules next irrelevant one until confliction disappears. Therefore, VFCC solves the problem of synchronization among VIPs by decentralized address conflict checking mechanism and conflict request scheduling strategy.

Finally, we introduce the problem of communication and synchronization between a VIP and the RTL. A request generator changes a request into signals and sends these signals to the RTL. This request flow is the communication between a VIP and the RTL. In this process, the RTL just waits for a request so that there are no overheads. Multiple VIPs communicate with the RTL in the same way.

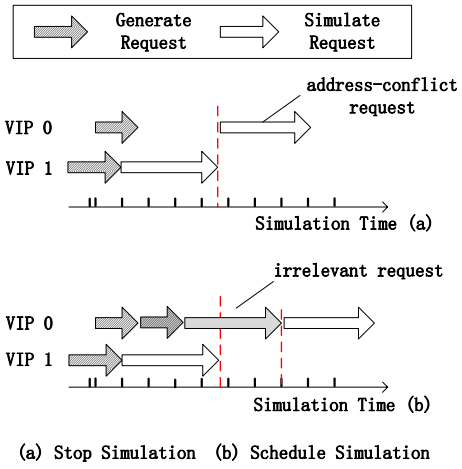

Fig. 5 Two strategies to synchronization among multiple VIPs.

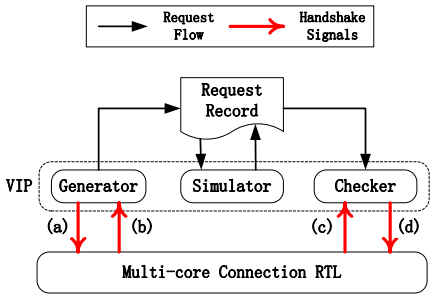

Fig. 6 Synchronization between a VIP and the RTL.

A VIP synchronizes RTL by a set of handshake signals. There may be changes of the cache model or the memory model at the end of each request. To ensure accuracy, the simulator needs to start with the RTL simultaneously and the checker needs to check when the RTL completes a request. Fig. 6 shows the synchronization between a VIP and the RTL. Fig. 6(a) shows a valid signal from the generator to the RTL which signs a ready request. Fig. 6(b) shows a ready signal from the RTL to start the simulator. Fig. 6(c) shows a valid signal from the RTL to the checker, and it starts to compare the results. Fig. 6(d) shows a finish signal from a VIP to release the request. This synchronization mechanism is accurate and avoids overhead.

D. Solving inconsistence between RTL and the simulator

VFCC also focuses on simulation accuracy as well as speed. Fig. 7 shows an example of the simulator inconsistent with the RTL. VFCC adds a self-adjusted mechanism to deal with that problem.

Fig. 7(a) describes a scenario of two requests from cores under MOESI protocol. Core 0 sends a load request with address A. Later, core 1 sends a load request with address B. To simplify the description, we assume that the cache use direct mapping strategy and address A, B are map to the same cache line. In core 0, the initial cache state is invalid, shorted as I. And in core 1, the initial cache state is exclusive, shorted as E. The RTL receives request A and B in order.

Fig. 7(b) describes the simulation process. The simulator processes request A first. After simulation, in core 0, the cache state is S(shared). In core 1, the cache state is S, too. Both cache lines correspond to address A. After simulation of request B, in core 0, the cache state is still S and the cache line stills corresponds to address A. In core 1, the cache state updates to E and the cache line corresponds to address B.

Fig. 7(c) describes the situation that RTL executes requests in the receiving order. The execution process of RTL is same as that of the simulator. Their results are consistence.

Fig. 7(d) describes the situation that RTL executes request out of order. By out-of-order we mean that the RTL receives request A first, while it completes request B first. After execution of request B, in core 1, the cache state updates to E and the cache line corresponds to address B. In core 0, the cache line is still I. After execution of request A next, in core 0, the cache state updates to E and the cache line corresponds to address A. In this situation, the execution result of the simulator is inconsistent with the RTL.

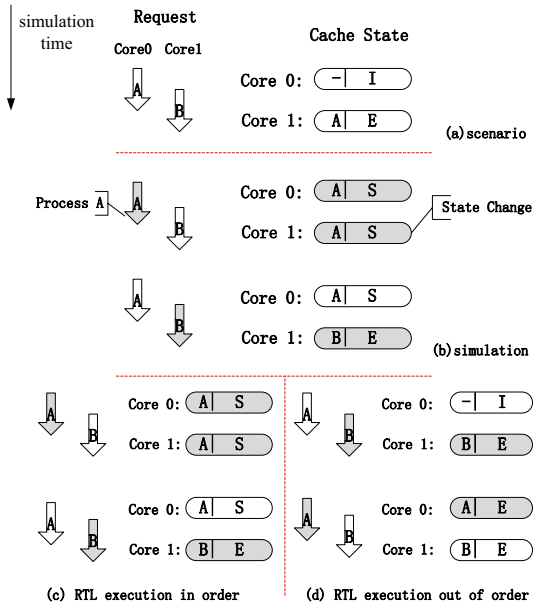

Fig. 7 An example of inconsistence between RTL and the simulator.

```
interface cache-update( int core_num, long address )
//CACHE UPDATE FUNCTION
//SCAN ADDRESS IN REQUEST_RECORD
   for( every core ){
    if( address in record[i] && record[i].sim && state is S or E){
       for( every core )
         request_record[i].cache[j].state=cache[i].state;
    }
   }
}
```

Fig. 8 Scan function in cache-update interface for self-adjusted.

The inconsistence comes from the different cache-update timing for the simulator simulates requests as the receiving order while the RTL executes requests out-of-order. And this inconsistence only happens on state E and state S, whose data is clean. VFCC adds a self-adjusted mechanism in cache-update to change the simulation result. Fig. 8 shows a scan function. There is an additional loop to scan the request record. If a request on-fly with the same address is in the record, it is already simulated and the state is E or S, the cache state needs to be updated to the new state. This mechanism avoids checking inconsistence.

IV. Experiments and Results

VFCC is designed to validate a cache-coherence component of a commercial 64-bit superscalar processor. This fully synthesizable processor runs at about 1GHz under TSMC 65nm technology. In this part, we first evaluate the speed of VFCC by comparing VFCC with a static-sequential simulator and a static-parallel simulator on the same host machine. Then, we analyze the parallelism impact and the dynamic generation impact on VFCC on two host machines. Finally, we illustrate the contributions of VFCC. TABLE I lists the parameters of two host machines.

TABLE I Parameters of host machines.

Parameters	Configuration Value	
Host Machine	A	B
Physical CPU Num	2	2
CPU cores	4	2
Siblings	8	2
Processor Num	16	4
CPU GHz	1.6~2.4	2.0
OS Version	Red Hat Enterprise Linux AS release 4	Red Hat Enterprise Linux AS release 4

We record the verification time of three simulators at the request number of 0.1 million to 2 million. Fig. 9 shows that VFCC gets a better performance than the other simulators. As the number of testing requests goes on, the advantage of VFCC is more obvious, especially than the static-sequential simulator. The dynamic-sequential simulator has a better performance than the static-sequential simulator for the overlap, but sequential simulation still has a limitation. When request number goes to 2 million, VFCC needs about 2.96 hours to finish verification, while the dynamic-sequential simulator needs 1.5 times longer than VFCC and the static-sequential simulator costs about 15.79 hours which is nearly 5.3 times.

We record the verification time of VFCC on two host machines with the request number from 0.1 million to 2 million. VFCC costs shorter time on host machine A for multithread programming. As shown in Fig. 10, the timing gap between two host machines gets larger as the request number grows. When the number of requests goes to 2 million, verification time on host machine B is almost 4 times as that on host machine A. It costs VFCC nearly 10.97 hours to finish 2 million requests on host machine B.

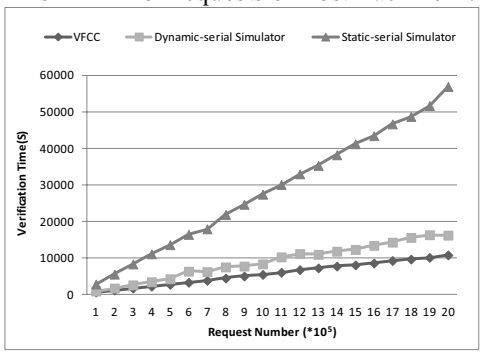

Fig. 9 The simulation time of static-sequential simulator, dynamic-sequential simulator and VFCC on Server A

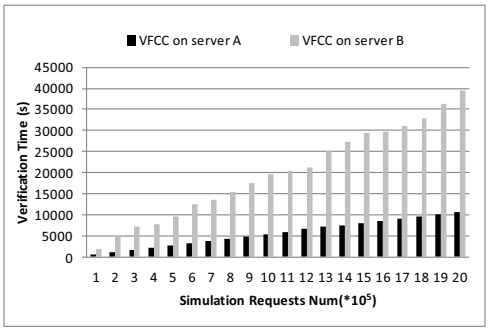

Fig. 10 Verification time of VFCC on host machine A and B.

To analyze the impact of dynamic generation on VFCC, we record request generation time and simulation time of the static-parallel simulator and VFCC.

TABLE II illustrates the comparison between a simulator and VFCC on host machine A and host machine B. The experimental results show that request generation time costs more than 52% of simulation time of the static-parallel simulator on host machine A and more than 33% on host machine B. Request generation is obviously the bottleneck of the static-parallel simulator. VFCC has 2.93 times speedup on host machine B and 5.12 times speedup on host machine A. As the number of cores grows, the advantage of VFCC is more obvious.

We also record the contributions of the static-sequential simulator and VFCC. The total number of bugs found by the static-sequential simulator is 15 while 52 bugs found by VFCC. VFCC contributes more for it constructs parallel scenario to validate concurrent requests instead of one by one. VFCC also tests more random requests in the same execution time than the static-sequential simulator.

V. Conclusion

This paper presents a high efficient verification framework using parallel simulation to validate a cache-coherence protocol component of a multiprocessor. VFCC exploits multi-level parallelism by analyzing the features of the RTL of cache coherence: parallelism among multiple function units in a VIP, parallelism among multiple VIPs and the overlap between a VIP and the RTL. Meanwhile, VFCC has effectively solved the communication and synchronization problem among multi-level parallelism with no overheads. VFCC also concludes and solves the inconsistent problem between the protocol simulator and the RTL to ensure accuracy. The experiments show that VFCC can be at least 5 times faster than a traditional sequential simulator on the same host machine.

Acknowledgments

This work was supported by the National High Technology Research and Development Program of China (Project No.2009ZX01029-001-002).

TABLE II Comparison between two simulators on host A and B.

	Req Num (*10000)		50	100	150	200
On Server A	static-parallel	Gen Time(s)	7333	15021	22237	29667
		Sim Time(s)	6164	12311	18919	27207
		Gen PCT(%)	54.3	55	54	52.2
		Total Time(s)	13497	27332	41156	56874
	VFCC Sim Time(s)		2621	5337	7946	10687
	Sim Speedup		5.15	5.12	5.18	5.32
On Server B	static-parallel	Gen Time(s)	9779	19531	29302	39033
		Sim Time(s)	18327	38192	57696	77696
		Gen PCT(%)	34.8	33.8	33.7	33.4
		Total Time(s)	28106	57723	86998	116729
	VFCC Sim Time(s)		9588	19685	29449	39517
	Sim Speedup		2.93	2.93	2.95	2.95

References

[1]. Archibald, P. A. and J. Baer, "Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model", ACM Transactions on Computer Systems, pp. 273-298, 1986.

[2]. Handy, J, The Cache Memory Book, Academic Press, 1993.

[3]. David L. Dill Andreas J. Drexler Alan J. Hu C. Han Yang, "Protocol Verification as a Hardware Design Aid", In Proceedings of the ICCD Conference, pp. 522-525, 1992.

[4]. Pong, F. and M. Dubois, "Verification techniques for cache coherence protocols", ACM Computing Surveys, pp. 82-126, 1997.

[5]. M. Tomasevic, V.Milutinovic, "A simulation study of snoopy cache coherence protocols", In Proceedings of the Hawaii International Conference on System Sciences, pp. 426-436, 1992.

[6]. P. Sweazy, A. J. Smith, "A Class of Compatible Cache Consistency Protocols and their Support by the IEEE Futurebus", Proceedings of the 13th Annual International Symposium on Computer Architecture, pp. 414-423, 1986.

[7]. J.F. Cantin, M.H. Lipasti, J.E. Smith, "Dynamic Verification of Cache Coherence Protocols", Proc. Second Workshop Memory Performance Issue, 2001.

[8]. Daniel Schwartz-Narbonne, Carven Chan, Yogesh Mahajan, Sharad Malik, "Supporting RTL Flow Compatibility in a Microarchitecture-Level Design Framework", In Proceedings of the CODES+ISSS Conference, pp, 343-352, 2009.

[9]. Dukyoung Yun, Jinwoo Kim, Sungchan Kim, Soonhoi Ha, "Simulation Environment Configuration for Parallel Simulation of Multicore Embedded Systems", In Proceedings of the DAC Conferenc, pp. 345-350, 2011.

[10]. G. Schirner and R. Dömer, "Fast and accurate transaction level models using result oriented modeling", In Proceedings of the ICCAD Conference, pp. 363-368, 2006.

[11]. Shacham, O., Wachs, M., Solomatnikov, A., Firoozshahian, A., Richardson, S., Horowitz, M. "Verification of chip multiprocessor memory systems using a relaxed scoreboard", In Proceedings of the MICRO Conference, pp. 294-305. 2008.

[12]. Sorin, D.J., M. Plakal, A.E. Condon, M.D. Hill, M.M.K. Martin, D.A. Wood, "Specifying and verifying a broadcast and a multicast snooping cache coherence protocol", IEEE Transactions on Parallel and Distributed Systems, pp. 556-578, 2002.

[13]. Delzanno, G., "Constraint-based verification of parameterized cache coherence protocols", Formal Methods in System Design, pp.257-301, 2003.

[14]. Tasiran, S., Y. Yu, B. Batson, "Using a formal specification and a model checker to monitor and direct simulation", In DAC, pp. 356-361, 2003.

[15]. Lv, Y., H. Lin, H. Pan, "Computing invariants for parameter abstraction", Proceedings of the International Conference on Formal Methods and Models for Codesign, pp. 29-38, 2007.

[16]. Rajarshi Mukherjee, Yozo Nakayama, Toshiya Mima, "Verification of an Industrial CC-NUMA Server", In Proceedings of ASP-DAC, pp. 747-752, 2002.

[17]. A. Meixner, D. J. Sorin, "Dynamic Verification of Sequential Consistency", In Proc. of the 32nd Annual Int'l Symposium on Computer Architecture, pp. 482-493, 2005.

[18]. A. Meixner, D. J. Sorin, "Dynamic Verification of Memory Consistency in Cache-Coherent Multithreaded Computer Architectures", In Proc. of the Int'l Conf. on Dependable Systems and Networks, 2006.

[19]. W. Marurngsith, R. N. Ibbett, "Specification-based Verification in a Distributed Shared Memory Simulation Model", SIMULATION, p. 0037549709349843, 2009.

[20]. E. Viaud, F. Pecheux, A. Greiner, "An efficient tlm/t modeling and simulation environment based on conservative parallel discrete event principles", In DATE, pp. 94-99, 2006.