Distributed Simulation of UVM Testbench

Theta Yang
Advanced Micro Devices
Bldg33 River Front Harbor, Zhangjiang Hi-Tech Park Shanghai,
PRC 201203

***Abstract*-A generic distributed simulation system is proposed to deal with the complexity of SOC level simulation testbench. The whole SOC level testbench is divided into multiple IP-level testbenches connected with inter process agents called IPC adapter. By running multiple simulation nodes in parallel and exchanging internode TLM messages through inter process communication (IPC), the SOC level simulation computing load is distributed and simulation performance gets boosted. In the proposed verification framework, IP-level simulation environment can be fully leveraged at SOC level without significant change. A two-node prototyping system is constructed in which IPC adapter is implemented in SystemC** [1]**, a popular networking library of ZeroMQ** [2] **is used to deliver messages between adapters, and the Mentor's UVMC**[3] **package is used to integrate these adapters into existing UVM based IP/SOC testbench. The simulation result of this prototyping system shows the simulation performance is good enough to exercise the chip-level test sequence.**

I. INTRODUCTION

To handle design complexity modern SOC design usually follows a component-based design and verification methodology. Third party IP's and in-house developed IP's are used as building blocks to be integrated at SOC level. The total design verification task is partitioned into multiple levels that typically include IP level and SOC level. The goal of IP level simulation is to verify IP functionality exhaustively, and the goal of chip-level simulation is to cover the interconnectivity and interoperability between connected IP's. Lots of verification frameworks have been invented to promote the verification reuse from IP-level to SOC level. UVM [4] is the one of the most popular in which the verification facilities in IP level testbench can be regrouped and reconfigured to reuse them at the SOC level. However extra workload is involved in both IP and SOC testbench development to support this kind of reuse. Firstly the IP's verification component needs to support configuration options what are not really needed at the IP level. For example at IP level, a UVM component works in active mode so it can drive and react with the IP DUT. At SOC level, since the IP's boundary interfaces are driven by peer IP instead of the verification component, the UVM component has to support an alternative passive mode to avoid multiple drivers. Secondly, instantiation, configuration and connection of IP-level UVM component at the SOC level needs significant work effort and debug time. Also to be noted is during the SOC-level simulation, even the IP UVM components are compiled and simulated, this results in huge memory usage for the SOC simulation and considerable degradation in the simulation speed. These performance factors prevent the full leverage of the verification component from IP level to SOC level.

The proposed distributed simulation reuses IP-level testbenches directly. Instead of integrating the IP level UVM component into SOC level, IP-level testbenches are connected by inter process communication agents and simulated simultaneously in parallel. Each IP-level simulation node simulates the compiled image of IP DUT and the IP's verification components. With customized IPC adapters transactions crossing the IPs' boundary can be sent or received across simulation nodes. In order to accomplish this, a centralized phase alignment server is used to align the simulation phases of all the parallel simulation processes to reach at same time. Under this framework, the whole SOC level simulation are partitioned into IP-level simulation and the effort of IP level simulation is fully reused at SOC level.

There are earlier papers that propose the idea of distributed simulation. In [5] and [6], simulations architecture named SCGPSim was invented to use the GPU as computing engine to accelerate SystemC based simulation. In [7] ideas are proposed to make SystemC simulation distributed among multiple threads in a single computer to accelerate the simulation model execution. In [8] the parallel discrete event simulation algorithms are compared and discussed. In addition to that there are also few articles released from EDA industry about the simulation engine to support multiple thread mode. There are major differences between our proposal and the earlier related research, the architecture proposed here is to be used in design verification

phase instead of design exploration phase for complex chip design, and work in a mix language simulation environment. We also promote IP level simulation reuse to SOC level.

The major features of our framework are discussed in section II. The detail of the distributed simulation architecture and implementation are described in section III. The prototyping system and the test results of the simulation performance are shown in section IV. Finally, we come to a conclusion in section V.

II. BENEFIT AND LIMITATION

The distributed simulation system has the benefits of simulation performance at the cost of extra latency at the IP boundary. The simulation model's behavior at the IP boundary is not the same with traditional single image simulation. This section clarifies the benefits and limitations of our proposed framework to help the user identify its proper use.

A. Benefit of the distributed simulation system

1) Simulation performance is scalable: SOC-level testbench is partitioned and distributed into multiple IP level testbenches, where the computing load and memory size required by SOC level testbench is reduced. Each IP-level simulation requires less computing performance and memory size. The framework leverages the multi-thread capability of a modern simulation server, where each IP has a separate simulation instance and the number of simulation processes can be allocated based on the required simulation performance.

2) Fully reuse IP-level testbench at SOC level: SOC-level simulation testbench is composed of a simulation cluster of IP-level simulation testbenches. The IPC adapters are extra verification components that can be added to UVM-based testbench without changing the existing testbench. The original verification components such as stimulus driver's and checkers are still functional as usual. The distributed simulation framework can also support an IP testbench implemented in different simulation language, for example a

UVM-based simulation can be connected with a SystemC based simulation.

3) IPC adapter design is generic and can be used by various interfaces and in multiple projects: The IPC adapters for standard interfaces such as SDP (AMD proprietary system bus) and AMBA AXI [9] can be reused for multiple projects. The communication carrier can be either inter process communication or network socket provided by computer operation system, the internal FIFO size is parameterized, and the payload type of the communication follow generic payload defined in standard SystemC TLM2. Since it is based on welldefined industrial standard it is easy to develop an IPC adapter for other interface protocols. Besides connecting other simulation nodes, the IPC adapter can provide extra simulation control API's (E.g., as to drive stimulus or to monitor output status). By using these API's, a script in high level language can be used as a testcase of the simulation system.

B. Limitation

The proposed simulation framework doesn't have cycle accurate simulation timing at the IP boundaries. Since there is no uniform simulation clock across different simulation nodes and the communications between nodes take place asynchronously, buffers are deployed inside IPC adapter to hold the TLM transaction before it is received by the remote node. The transaction buffer and inter-node communication causes extra transaction latency between IP's. As a result, the simulation behavior at the IP boundary is different between the proposed distributed simulation and the traditional single image simulation. The simulation framework simulates cross IP interoperation at the TLM abstraction level. It cannot replace the traditional SOC level simulation which requires simulation to be performed at signal level or bus protocol level. But by partitioning the SOC DUT at standard interfaces, the interface protocol compliance verification is already covered at IP level. Many standard interfaces such as AXI can tolerate extra latency by inserting wait cycles to a transaction, but not all SOCs incorporate such kind of interface as the chip-level interconnection to connect IPs. The simulation framework only applies to SOCs which use latency-tolerate interface for IP connectivity. And as it involves extra latency at IP boundaries, the simulation framework is not suitable to run performance measurement tests.

III. DISTRIBUTED SIMULATION ARCHITECTURE AND IMPLEMENTATION

The architecture of the proposed distributed testbench is demonstrated in Fig.1. SOC-level simulation is composed of a cluster of simulation instances among which one of the instance simulates the SOC DUT and its testbench. Each of the other instances simulates an IP DUT together with its testbench. IPC adapters are connected in point-to-point style in order to relay the TLM transaction from interface UVC in SOC simulation node to interface UVC residing at the distributed simulation node. All of the IPC connections are between

SOC testbench and IP testbenches, and there is no direct connection between the various IPs' simulation nodes. The IP testbenches are wrapped by a stub module at SOC testbench, the stub module and the distributed IP testbench simulation instances add together to perform the IP's functionality in SOC-level simulation.

A. Topology mapping

In the proposed distributed simulation framework, if an IP is selected to be simulated in a distributed simulation instance, the SOC DUT will instantiate its stub module. The stub module doesn't include the internal logic of that IP, but only the top-level module ports. The SOC DUT maintains the connectivity between IPs and the connection doesn't change either when IP is in real logic view or in the stub module view. SOC level can create different simulation configuration and for each configuration the view of IP is selected either as full logic module or as stub module. When an IP is configured in stub module the IP level testbench is required to run in parallel with SOC level testbench to perform the IP level functionality. To connect IP level testbench with SOC level testbench, the interface UVC and the corresponding IPC adapters are instanced under the IP's stub module at the SOC level. When an IP is configured in rtl view, the IP logic is included by SOC DUT and simulated at SOC level testbench. The IP level testbench of the IP is not required to be simulated in distributed instance. With this topology mapping methodology, we can flexibly select either to distribute an IP simulation to a separate simulation instance or include the IP into SOC-level simulation instance. A same test case can be run under different SOC configuration based on the tradeoff between simulation performance and simulation model's accuracy.

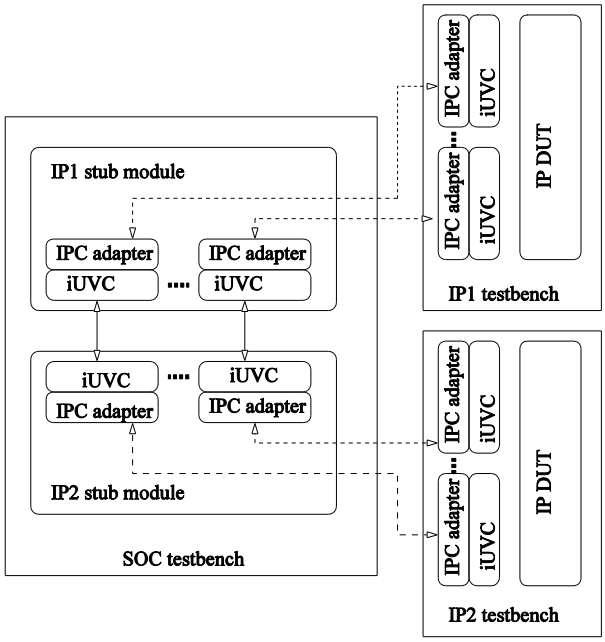

Figure 1. Block diagram of proposed distributed simulation framework

B. Traffic pattern

An IP has an interface through which it sends out request or receives the request from the external module, now there are two ways to it. For instance, if IP is the traffic originator for the interface then it requires a traffic completer to be connected at the other side of the interface, Similarly if the IP accepts traffic request and response for completion from an interface, then the IP is an traffic completer and it require a traffic originator being connected at the other side of the interface. Now in IP level testbench, since we have no peer IP connected external to the IP boundary, the corresponding interface UVC's are used to behave like the peer IP either as a traffic originator or traffic completer. In IP level testcase we will be using the verification API's provided by these interface UVC's to drive the request to DUT or to receive the response request from the DUT. When each IP testbench is constructed with this uniform structure then it is possible to connect all the IP testbenches through those interface UVC's to construct the SOC level simulation. In SOC level testbench, the interface UVC's are configured to accept the request from the peer IP testbench instead of sequences from the testcase. When the response is returned from DUT, the interface UVC will drive the response to the peer

IP. Thus the interface UVC acts like transaction repeaters to relay transactions across IP boundary at SOC level.

Since both IP level testbenches and SOC level testbench are simulated with their own simulation instance simultaneously, each of these testbenches uses one or multiple IPC connections which are in-turn connected in a point-to-point style to construct a simulation cluster and this simulation cluster helps in performing the SOC-level functionality. The IPC adapters are built to connect peer interface UVC through point-to-point IPC communication using ZeroMQ. Each IPC adapter forks two background processes, one is a service process that binds to a ZeroMQ endpoint and a client process that connects to a second ZeroMQ endpoind. For interface UVC as an originator, the attached IPC adapter receives transaction request and sends response to its remote interface UVC and for interface UVC as a completer, the attached IPC adapter receives transaction response and sends transaction request. The interface UVC are of two different types, they both have a sending port and a receiving port. The IPC adapter can be unified as a single type as the IPC adapter don't interpret the transaction content's but only serves as a network agent to send or receive data stream for networking.

ZeroMQ is a high-performance asynchronous message communication library which can be used in distributed applications. ZeroMQ system can run without a dedicated message broker, the message source and message sink are connected in a point-to-point style. It has a built-in *"Request-reply"* traffic pattern to connect a set of clients to a set of services which also match with our distributed simulation requirement. A very handy feature of ZeroMQ is that it can support the message carrier for both IPC and TLP. The switch between IPC communication and TLP communication is achieved by just changing the name of endpoint. This feature makes it quite easy to run the distributed simulation on a multiple computer cluster instead of single computer. However our current experiments are done on a single computer sever using IPC communication carrier.

C. IPC adapter

Fig.2 shows how an IPC adapter is connected with interface UVC and extends its function. When interface UVC is a transaction completer, it accepts a transaction request from DUT's originator interface and sends the request to IPC adapter. The IPC adapter is responsible to buffer the request and send it to external components through IPC networking endpoint. The IPC adapter then takes charge to receive the response of the transaction from IPC endpoint and returns it to the connected interface UVC. And then the interface UVC will send the response back to DUT. When the DUT is a transaction completer, the IPC adapter receives the transaction request by external IP through the IPC endpoint and forwards the request to interface UVC which will perform as a traffic originator. When the transaction response is returned to the interface UVC, it will be forwarded to the IPC adapter. And then the IPC adapter sends the response to peer IP simulated in distributed node through the IPC endpoint. Two IPC endpoints are used by an IPC adapter, one is used to send out transaction to network and another is used to receive transaction from the network. An egress FIFO and an ingress FIFO are used to buffer the transactions to be sent or to be received. The depth of egress FIFO is set to be the maximum supported outstanding transactions number so that the egress FIFO will never overflow. And the ingress FIFO is single entry in depth, IPC adapter will reject the incoming transaction when ingress FIFO is occupied by blocking the corresponding IPC endpoint. The local IPC adapter works together with the connected remote IPC adapter to move transaction in local egress FIFO to remote ingress FIFO.

The IPC adapter is implemented in SystemC so that we can use generic networking communication library based on C or C++ language, and we select the popular ZeroMQ. The IPC adapter provides DPI routine to be used by System Verilog based interface UVC to send/receive transactions. The IPC adapter forks sending and receiving thread which monitors the IPC FIFO and performs the sending or receiving of the TLM transactions through IPC endpoint. The IPC adapter is designed as a generic building block, different transaction type and FIFO depth are all parameters that can be specified when it is instantiated. For transactions to be sent or received through networking using ZeroMQ, it needs to be converted from the structured transaction packet to a stream of data byte. The C++ *boost* serialization library is used to perform transaction class serialization and de-serialization for our implementation.

Each IPC adapter forks two background processes, an operating system thread to send and receive transactions through ZeroMQ API. The receiving thread is a "*server*" which accepts the connection and request from a "*client'* of the remote connected IPC adapter. And the sending thread is a "*client*" to the remote connected IPC adapter.

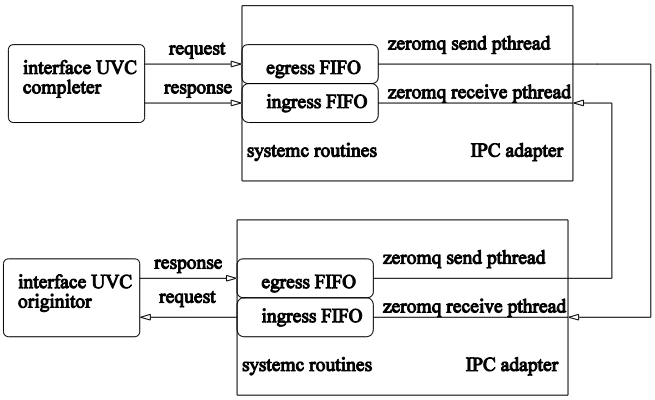

Figure 2. Connection of IPC adapters

The IPC adapter strictly follows the request-reply traffic pattern and guarantees there are no packets lost during the networking even when a "*server*" thread is running. The sending thread gets blocked when the reply packet is not received after sending a request packet. And the receiving thread will block when the ingress FIFO is full and waiting for the interface UVC to consume the transactions received. Since the operation system thread is outside the simulation's scheduler, when the background threads are blocked the simulator instance is not blocked and the simulation time modeled there still runs. The asynchronous communications between the IPC adapters are decoupled with the simulator processes.

The FIFOs used by the IPC adapters are "*tlm_fifo"* in SystemC and the IPC adapter itself is implemented as SystemC module. The FIFOs' TLM ports are exposed at IPC adapter level as module interface connected with the UVM side. UVM Connect [3] library from Mentor Graphics is used for this connection. The library provides TLM1 and TLM2 connectivity between SystemC and System Verilog models and components. It implements a DPI layer which do data type conversion and provides standard connection APIs to be used in the SystemC and System Verilog mixed simulation environment. The SystemC based implementation makes the IPC adapter a modular design used as an add-on component to corresponding interface UVC.

D. Phase alignment

Each IP testbench needs to be setup properly and initialized before it can originate or respond to the transaction at the IPC adapter. For example, the clock and reset needs to be driven properly and IP should be programed to be ready to interact with the peer IP. In other word, all IP testbenches are required to reach the functional mode before the cross node TLM transaction can be exchanged. A central server process is used to sync the simulation phases between simulation nodes. All sub phases under UVM simulation run phases are synchronized between simulation nodes as shown in Fig. 3. When a simulation process wants to exit a specific simulation phase, it will send phase sync message to the phase server. The phase server will acknowledge the simulation phase requestor after the request for all simulation nodes are received. This way the simulation phases across simulation nodes are aligned. After a simulation node sends a phase request message to sync sever, it will get block before it can the phase acknowledge from the sync server. The slower simulation instances keep running, until all the simulation nodes is ready to exit the simulation phase.

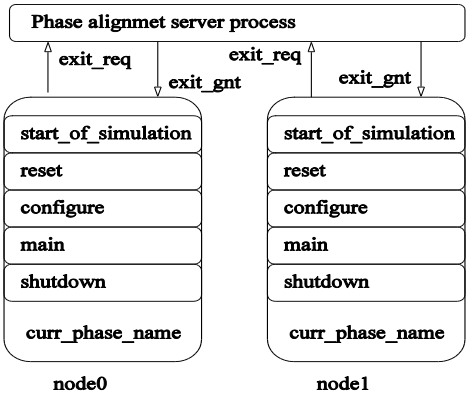

Figure 3. Simulation phase alignment between simulation node

IV. SIMULATION RESULT ANALYSIS

A. Back to back connected IPC adapter test

To measure the performance of IPC adapters to transport transactions with IPC communication, we took several back to back tests. In the first test, two IPC adapters are connected with each other but not attached to an interface UVC. Transactions with different payload size are injected into the egress FIFO of one IPC adapter, and then it is received from the ingress FIFO of another IPC adapter. The transaction sending time from one IPC adapter and arriving time at the other IPC adapter are recorded. Fig.4 shows the distribution of the physical time required for the IPC communication between simulation nodes. The results show that transaction transport time for typical payload size is within a range of 80ms to 200ms even if the payload size becomes quite large such as 1024 bytes. But for packets that are small in size, there is no shrink in the packet transport time.

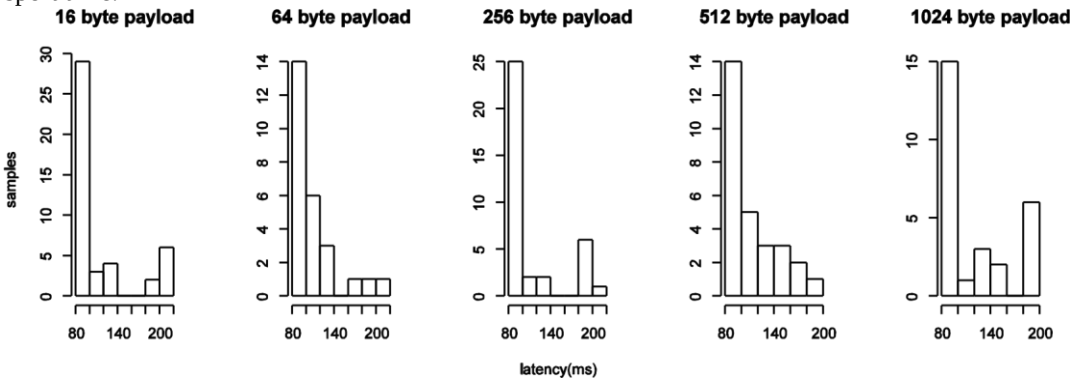

Figure 4. Time required by the IPC communication between simulation nodes

The transaction transport time is the absolute latency involved in the distributed simulation architecture to send transaction from one simulation node to another. It is asynchronous to the simulation time modeled in each simulation node. From a specific simulation node, the latency is presented as the cycle number needed for sending the transaction request at the egress FIFO to the receiving time of the transaction response at ingress FIFO. The value is not a fixed number as the networking performance of the running server varies from time to time. In the second test, we use an IP developed in our project with two standard system bus interfaces. The IP is a transaction completer for one interface and an originator for the other interface. The IP level testbench is based on UVM with interface UVC connected at each SDP interfaces. We extend the testbench and attach an IPC adapter to each interface UVC. The two IPC adapters are connected back to back, when the DUT initializes a transaction, it will be forwarded from the DUT originator interface to the completer interface. The latency caused by both interface UVCs and IPC adapters can be measured in term of numbers of simulation cycle at the interface. The below Fig. 5 is the distribution of the latency in cycles for transaction with different payload size. It shows non-ignorable extra latency is involved at the interface. Even the interface protocol can tolerate the extra latency by adding idle cycles between transaction request and response, the internal buffer of IP may not be stressfully tested as in IP level testbench.

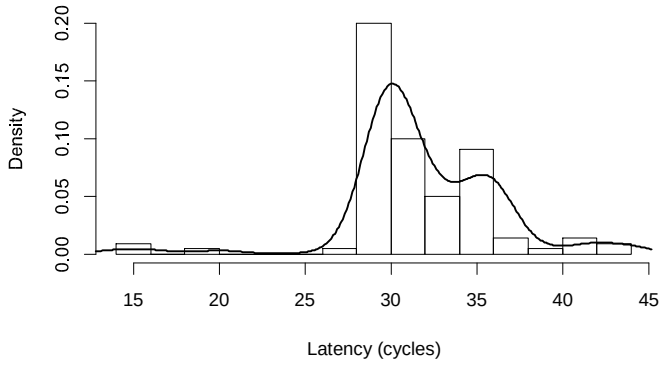

Figure 5. Distribution of latency in simulation cycles for back to back interface UVC

B. Simulation performance of prototyping system

The top level of the prototyping system for the distributed simulation framework contains two simulation nodes, one simulates SOC level testbench and another simulates IP level testbenches. The IP selected is the same one used in above section to measure the back to back performance. The framework depicted in section 3 is followed to construct the distributed testbench. Fig. 6 shows the block diagram of the multiple node prototyping system.

Both the simulation nodes had testbench already coded in the UVM implementation. A special UVM test is developed for each UVM testbench, and they are running separately on each simulation node when the distributed simulation is launched. The test creates transaction request sequence and response sequence and then sent to corresponding interface UVC sequencer. The sequence will manipulate the *"tlm_fifo"* contained in IPC adapter through the API provided by UVMC library.

The IP and SOC all need a set of register programming to be initialized to enter the functional mode. They are initialized separately, the first one which finishes the initialization will be suspended when it tries to sync with the phase sync server until the second node finishes the initialization, then the first one resumes to continue the execution. Several testcase developed in original IP level testbenches have been run in the distributed simulation. The same testcase extends its functional coverage when it runs in the distributed simulation environment. In original IP level single node simulation, the cross boundary IP level transaction is completed by the response handler in the interface UVC and in the distributed simulation, the response handler is changed to forward the transaction to IPC adapter and then to another simulation node, the transaction response is generated by logic simulated in the other simulation node and then sent back. The IP level test case is directly reused at this distributed simulation testbench and its simulation coverage is extended to cover other logic residing outside the IP.

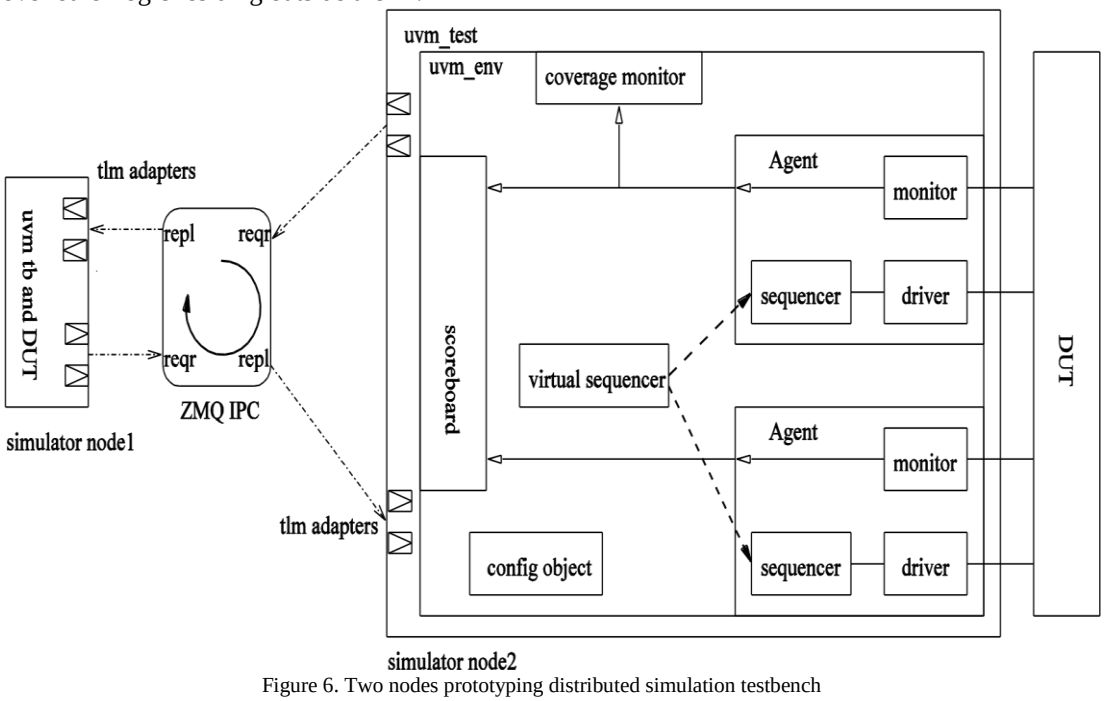

Figure 6. Two nodes prototyping distributed simulation testbench

We compared several performance factors between the distributed simulation system and the original single image simulation system. We use the same server to run all these tests, and for the disturbed simulation test, the two nodes are run in a single server. A suite of testcase which involves cross IP traffic are simulated in both single image simulation and the distributed simulation. The below Table I shows the results including compile time and run time performance factors. Both the memory used to compile the simulation image and the memory used to run the simulation is significantly reduced. The average simulation speed is increased from *5.4 ns* sim speed per seconds to *9.6 ns* sim speed per second and it makes the simulation much faster. In our project, both the IP level and SOC level simulation speed is quite slow and it is only several nanosecond

simulation time progress within a physical second. The simulation behavior deviation resulted from communication latency is relatively alleviated.

TABLE I
PROTOTYPING SYSTEM PERFORMANCE FACTORS

Configuration	Compile Peak Mem(M)	Compile Avg. Mem(M)	Compile Time(S)
Single image sim	13378	8385	5548
DIST sim SOC node	8968	5674	4111.6
DIST sim IP node	4978	4820	5654.9
	Sim Peak Mem(M)	Sim Avg. Mem(M)	Sim Speed(ns/s)
Single image sim	8884	7966	5.4
DIST sim SOC node	5458	4799	9.6
DIST sim IP node	4978	4820	8.1

The simulation result shows that by using the distributed simulation framework, the SOC level test can be run faster when compared with the single process simulation.

B. Bring the script capability to UVM testbench

The distributed simulation frame work can also be used in a single node simulation environment. In the implementation of prototyping system, besides to connecting the two simulation nodes, we use a simulation control script to control the IP simulation to develop debug. Fig.7 demonstrate how it works. A backend service routine in C++ talks with the IPC adapter in the simulation node through the ZeroMQ API. The routine contains a queue to buffer the transaction request or response to be sent to the ingress FIFO of an IPC adapter. And it also contains the list to record the callback routine to be called when a response or a request is received from an IPC adapter. When the service routine is initialized, it creates a ruby interpreter and loads the IPC adapter control script in ruby. Most of other popular script language can also be used as the script language like ruby provided it can support to use C++ routine as its language extension. The service routine takes part in the simulation phase synchronization protocol. When the simulation reaches to the phase that the IP testbench can handle transaction from IPC, the service routine polls for the availability of all the ZeroMQ sockets used IPC adapters. When a socket is available and if it is connected with ingress FIFO of IPC adapter, the service routine will move the transactions from its internal queue to the IPC adapter. And if it is connected with an egress, the service routine will then invoke all of the callback functions which has been registered for that socket. The script currently support generating transaction, registering callback for response handler or response checker function. A special *"wait_all_done"* API is also supported which will wait until all of queued transaction is clean.

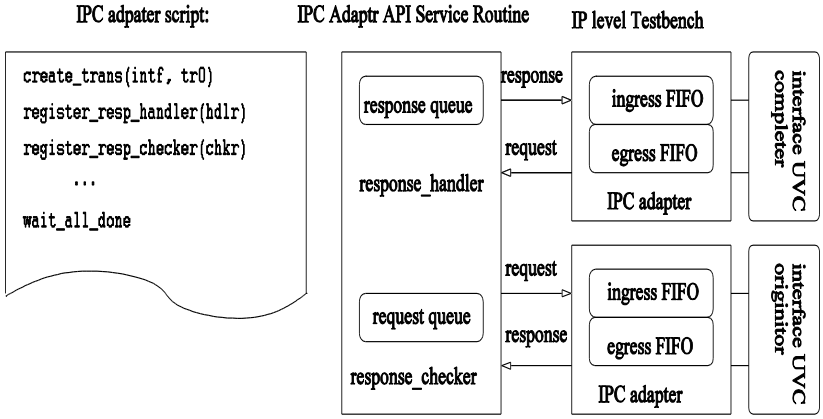

Figure 7. Single node simulation API

IV. CONCLUSION

A new testbench framework is proposed to run SOC simulation as a distributed cluster of IP level simulation. In the framework, chip-level testbench can reuse IP-level testbench directly without bringing IP level simulation component at the SOC-level, Hence, testbench construction effort at SOC level is largely saved. By dividing the chip level simulation into multiple simulation nodes, the simulation limitations such as simulation capacity and performance from a single node simulator are overcome.

The proposed distributed simulation framework is not supposed to replace totally the current single image single instance simulation, but it is a good complementary testbench especially when SOC level testbench is not yet available. By connecting IP standalone testbench together, the SOC level can construct a workable simulation testbench in a very short period and also interoperability issues if any between IP's can be identified in early design phase. When IP's are fully integrated into SOC level, the testcase itself don't need to be changed, but based on the topology maintained methodology a configuration switch can change the IP from stub module to the rtl module. The distributed simulation leverages the multi core servers easily acquired nowadays and provides a better simulation performance compared with the single image simulation. It can also be extended to support multiple computer simulations.

The current prototyping supports only two simulation nodes and runs on a single simulation server. The distributed simulation frame is still far away to be fully qualified and proven the can work for a cutting edge design project and for simulation sign off. But it has proven its value for fast SOC testbench setup and simulation at the early design stage. As the IP testbench and testcase are used directly lots of SOC level rework has been saved, and SOC level simulation can take the already done IP level simulation as reference to ease the simulation debug

There are several areas to be optimized to decrease the cross simulation node communication latency. The phase alignment algorithm can be further extended to support a global slow clock, we can make all simulation instances run synchronously on the slow clock. With this global clock, when a simulation instance tries to run the simulation ahead, it will get suspended and waits for the slower simulation node. And with the global clock, all the cross node traffic can be constrained to complete in the same cycle. With this extension, the simulation time deviation between simulation nodes can be limited to the single cycle of the slow global clock. The middle ware library used in the framework have room to be optimized as well. Even the UVMC connection and ZeroMQ is fast enough in our prototyping work, they can be optimized and tuned in our specific usage scenario. We expect further investigation can take place to make the distributed simulation system used in production level.

ACKNOWLEDGMENT

The author would like to thank Wayne Yun, and David Chen for the review of the initial proposal of this testbench framework and their valuable suggestions. And the author also thanks Vishwath Bhandary for cleaning up all grammatical issues in the draft paper.

REFERENCES

[1] Accellera, "SystemC LRM," available at http://www.accellera.org/downloads/standards/systemc

[2] P Hintjens, "ZeroMQ: The Guide," available at http://zguide.zeromq.org/

[3] Mentor Graphics, "UVM Connect" available at https://verificationacademy.com/

[4] Accellera, "Universal Verification Methodology," available at http://www.accellera.org/downloads/standards/uvm [5] M Nanjundappa, A Kaushik, H D Patel, S K Shukla, "Accelerating SystemC simulations using GPUs," IEEE HLDVT, 2012 [6] H D. Patel, B A. Jose, S K. Shukla, "SCGPSim: A Fast SystemC Simulator on GPUs," in Proceedings of IEEE ASPDAC, 2010.

[7] A Mello, I Maia, A Greiner, F Pecheux, "Parallel simulation of SystemC TLM 2.0 compliant MPSoC on SMP workstations," in DATE, 2010

[8] W. Chen, X. Han, C.W. Chang, R. Dömer, V.O. Topaloglu, "Advances in Parallel Discrete Event Simulation for Electronic System-Level Design," IEEE CEDA, 2013

[9] ARM, "AMBA AXI and ACE protocol specification," available at http://www.arm.com/