A Highly Efficient Full-System Virtual Prototype Based on Virtualization-Assisted Approach

Hsin-I Wu[1], Chi-Kang Chen[2], Tsung-Ying Lu[3] and Ren-Song Tsay[4]
[1,3,4]National Tsing Hua University, Department of Computer Science, Hsinchu, Taiwan
[2]Industrial Technology Research Institute, Hsinchu, Taiwan
[1,3,4]{hiwu.dery, iu0987810505, rstsay}@gmail.com [2]{rc}@itri.org.tw

***Abstract*—An effective full-system virtual prototype is critical for early-stage systems design exploration. Generally, however, traditional acceleration approaches of virtual prototypes cannot accurately analyze system performance and model non-deterministic inter-component interactions due to the unpredictability of simulation progress. In this paper, we propose an effective virtualization-assisted approach for modeling and performance analysis. First, we develop a deterministic synchronization process that manages the interactions affecting the data dependency in chronological order to model inter-component interactions consistently. Second, we create accurate timing and bus contention models based on runtime operation statistics for analyzing system performance. We implement the proposed virtualization-assisted approach on an off-the-shelf System-on-Chip (SoC) board to demonstrate the effectiveness of our idea. The experimental results show that the proposed approach runs 12~77 times faster than a commercial virtual prototyping tool and performance estimation is only 3~6% apart from real systems.**

I. Introduction

As designers continue to integrate increasing numbers of components into a system, the growing complexity of inter-component interactions has led to dramatically lower productivity. Since interactions and system performance can only be debugged and verified at the system level, effective and efficient full-system virtual prototypes are critical for system behavior verification and system performance estimation. Unfortunately, the performance of modeling inter-component interactions has failed to satisfy advanced system designers. Moreover, there are no effective models for analyzing system performance accurately during early system design stage. Therefore, in this paper we propose a virtualization-assisted (VIRA) approach in order to provide a practical solution for modeling and performance analysis of full system simulations.

The VIRA approach is an hardware-assisted approach that models target-simulated components as software-modeled components (SMCs) or hardware-assisted components (HACs). SMCs are executed by the host system's Central Processing Units (CPUs) and HACs are offloaded to host devices that are equivalent functions to HACs. As a result, we can save the time of implementations for HACs while improving simulation performance.

However, in addition to increasing simulation performance, HACs (and SMCs) must be synchronized to model deterministic inter-component interactions for debugging purposes. To address this issue, the VIRA approach uses a deterministic synchronization process to manage inter-component interactions, particularly data dependency. The VIRA approach simulates all synchronization points in chronological order so as to produce deterministic inter-component interactions and reproduce any system bugs. Essentially, the VIRA approach provides a deterministic synchronization process that defines synchronization points and implements a non-intrusive trap mechanism to intercept synchronization points. The unique feature of the trap mechanism is enabling host devices to intercept synchronization points precisely automatically. Therefore, the VIRA approach achieves very efficient and effective virtual prototyping.

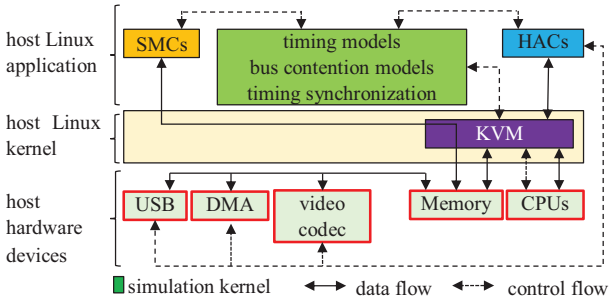

Figure 1: The proposed VIRA virtual prototyping architecture. The interfaces provide controls of the hardware devices and manage runtime operations. SMCs and HACs are simulated on the same SoC board.

To accurately estimate system performance, an analytic timing model is designed to compute the target time, and a quality of service (QoS)-aware bus contention model is built to calculate the bus utilization and contention delay caused by the target-simulated components. The generated estimation results are deterministic because the timing and bus contention calculations are based on runtime operation statistics, which are consistent for simulations repeated under the same input and runtime conditions.

To realize the ideas behind the VIRA approach, the VIRA approach utilizes hardware virtualization techniques to implement all SMCs and HACs on a single SoC platform, as shown in Fig. 1. SMCs, HACs, timing models, bus contention models and timing synchronizations are all executed as host Linux applications. In case a target system component finds no functionally equivalent host devices for HAC offloading, the target component is then modeled as an SMC. We specifically modify the host device drivers in the Linux kernel to provide

controllability for HACs and improve KVM (kernel-based virtual machine) implementation in order to intercept synchronization points easily.

Experimental results show that the full-system VIRA virtual prototype presents only 3~6% performance estimation error rates compared with real systems. Additionally, the VIRA virtual prototype runs 12~77 times faster than ARM® Fast Models, 1.6 times faster than the Field Programmable Gate Array (FPGA) based approach and is 7~47 % slower than the ideal native executions. The VIRA approach is demonstrated to be efficient and can be effective for hardware/software co-simulations during early system design stage.

II. RELATED WORK

Various types of virtual prototyping acceleration approaches have been proposed in the past. We briefly review how existing approaches accelerate simulations and discuss their pros and cons.

A. Hardware-Based Acceleration Approaches

Some hardware acceleration approaches [2][3] attempted to increase the simulation performance of SMCs using hardware threads, which have lower synchronization and communication overheads. Most modern Graphic Process Units (GPUs) support hardware threads. The challenge of these approaches is that, in practice, both converting SMCs for GPU execution and managing inter-component interaction orders between SMCs in hardware threads are complex tasks because GPUs and CPUs adopt quite different thread programming models.

Using FPGA to accelerate the HACs is another popular approach. Nakamura [4] proposed a shared communication registers (SCRs) interface for modeling the inter-component interactions between target-simulated components. However, accessing SCRs requires passing data through a physical bus, such as the Peripheral Component Interconnect (PCI) bus, whose transfer rate is limited. Therefore, this data passing step is a performance bottleneck that limits the approach.

Protoflex [5] proposed a hybrid approach by offloading common functionalities onto FPGAs while running software to model infrequent operations such as I/O operations. This approach is designed to provide functionality similar to that of QEMU but at an FPGA-accelerated speed. However, since it considers no timing models, it cannot deterministically reproduce the interactions of target-simulated components.

On the other hand, FAST [6] proposed an FPGA-based approach with a timing model. FAST computes timing models in FPGAs but executes function models in the host CPU. The issue of this approach is that the heavy communication bandwidth requirement between the CPU and FPGA limits the scalability of modeling inter-component interactions.

In contrast to FAST, approaches such as RAMP [7], and FRAME [8] put function models in FPGAs and compute timing models in the host CPU. In general, these approaches try to fix most function models to avoid re-synthesis and then compute the timing information according to the design architecture, such as cache organization. The simulation speed-up is mainly from the parallel execution of FPGAs. Nevertheless, these FPGA approaches require expert partitioning of the function models in order to work properly. Therefore, they are more suitable for designs with well-structured micro-architecture rather than for system designs with many variations.

Thus, in response to the severe limitations in performance and application scope of existing approaches discussed above, we propose the VIRA approach, a virtualization-assisted approach, in this paper for use in practical virtual prototypes.

III. THE VIRTUALIZATION-ASSISTED APPROACH

In order to efficiently and accurately model inter-component interactions and analyze a target system performance, the VIRA approach incorporates a deterministic synchronization technique, runtime operation statistics-based analytic timing model, and bus contention model. The goal of the VIRA approach is to provide a full-system virtual prototype good for debugging purposes while having performance estimation results that closely match real systems. In the following, we elaborate the key ideas behind the VIRA approach.

A. Deterministic Synchronization

The deterministic synchronization process of the VIRA approach provides the capability to facilitate debugging of inter-component interactions. However, if inter-component interactions are not synchronized properly, the generated results may not be desired. To solve this issue, we develop a deterministic synchronization process to manage the chronological order of HACs and SMCs to produce deterministic inter-component interactions. The deterministic synchronization process defines a transaction as the segment of operations between two consecutive synchronization points. A synchronization point is simply any operation that can affect the SDA (shared data access) order. Examples of synchronization points include *interrupts* and SDAs.

Therefore, the deterministic synchronization process performs synchronization only at necessary locations instead of every cycle or every data access point and hence greatly improves simulation performance and reproduces consistent inter-component interactions good for debugging. However, a proper timing model is required to determine the chronological order and hence is discussed in the following.

B. Runtime Operation Statistics-based Analytic Timing Model

Since in reality the clocking rate of FPGAs or host machines depend heavily on the hardware implementation, building a deterministic timing model based on the local clock count information is not feasible. Nevertheless, we observe that all target designs have definite specifications for basic operations, such as instructions, cache hit/miss, memory access, etc., which always have fixed data processing time and transmission timing behaviors and can be analytically modeled [10].

With the timing models of basic operations, VIRA gathers the runtime operation statistics among target-simulated components and knows the numbers and types of basic operations in each transaction. The information is then applied to compute the time used by the transaction.

Specifically for CPUs, the runtime operation statistics generally refer to the number of different types of instructions in each transaction. Based on the CPI (cycles per instruction) number of each instruction type and data load/store latency statistics of the target cache and memory system, we can derive the data processing and transmission time for the analytic timing model. For other devices, such as video decoder engines, DMA (Direct Memory Access) engines or SMCs, we can adopt the same process to compute the analytic timing model based on the main input/output data operations.

In addition to the analytic timing model above, we also consider bus contention effects for shared buses in real systems. Next, we will explain how to handle the bus contentions of different bus topologies and different priorities.

C. QoS-Aware Bus Contention Model

The VIRA approach extends the activity-sensitive contention delay model approach [1] to support multiple layer buses and QoS. The idea is to use the number of bus requests in each period of time to derive the bus contention probability and compute the probable contention delay based on the required data transmission time instead of using a slow but cycle-accurate model for analysis.

Next, we explain how to compute contention delay for multiple layer buses starting from a single layer bus. In order to simplify the discussion, we assume that the system bus is an out-of-order bus and first consider the case where there are at most two bus access requests at a time, so we can simplify the formulation of the activity-sensitive contention delay model to that of Eq. (1), where p_j is the probability of bus request conflict derived from the bus utilization ratio of a target-simulated component j, b_j is the data transmission time in the clock cycle unit and d_j is the average contention delay.

$$d_j = \frac{p_j}{2}.(b_j + 1) \quad (1)$$

Then, for each transaction, the VIRA approach counts the number of bus requests and computes the bus utilization to compute the contention delay for each request following Eq. (1), based on the data transmission time of all master components. For details, refer to the work in [2].

Originally, we assumed that all target-simulated components are of the same bus access priority so that for the case of n target-simulated components attached on the same bus, the average probability to win the bus contention is $1/n$. When QoS is considered, then each target-simulated component is assigned a certain priority level. If a component has higher priority, it is assigned a higher weight w and its probability of winning bus contention will become higher. The actual probability of bus conflict depends on the priority scheduling algorithm. For example, in the Cubietruck platform we tested, the probability p'_j of a bus request conflict from component j is calculated from Eq. (2), where p_j is the bus conflict probability with no priority setting and w_j is the priority weighting of the component j.

$$p'_j = p_j \times (1 + \frac{1}{n} - \frac{w_j}{\sum_{j=1}^{n} w_j}) \quad (2)$$

Then, we use p'_j to calculate the bus contention delay considering QoS using the following Eq. (3).

$$d_j = \frac{p'_j}{2}.(b_j + 1) \quad (3)$$

IV. EXPERIMENTAL RESULTS

The VIRA virtual prototype is implemented using an entry-level off-the-shelf Cubietruck platform as the host, containing an ARM® SoC with 960 MHz dual-core Cortex®-A7. As a baseline for comparison, we use the ARM® Fast Models and COREMU [9] to simulate the same target designs on a machine equipped with Intel® Xeon® E5-2650. For benchmark cases, the standard SPLASH-2 parallel program test cases are adopted.

A. Modeling Comparison

Before conducting performance comparisons, we confirm that the synchronization point order is deterministically reproducible by checking the log file of the synchronization points from target-simulated components. Then, we compare the performance of various cases listed below on the Fast Models, COREMU, native execution and the VIRA approach.

1) Host CPUs as HACs

Since the target CPUs and the host CPUs are of the same ISA, we use the host CPUs as HACs for the target CPUs. The VIRA virtual prototype performs 12~77 times faster than the Fast Models, 2~47 times faster than COREMU and has only about 7~47% overhead compared to the ideal native execution, as summarized by the test results of the SPLASH-2 in Table 1.

In Table 1, the simulation speed-up is recorded in columns two and three. If converted to measures in term of MIPS (million instructions per second), Fast Models executes at 27 MIPS for the lu case and the VIRA approach performs at 351 MIPS for the same case. Interestingly, the VIRA approach performs 12~47 times faster than the parallel simulator COREMU. The mere 7~47% overhead of the VIRA approach compared to the ideal native execution, which gives the best possible runtime, shows that the VIRA approach is truly efficient and effective for practical use because the VIRA approach still performs at 177 MIPS for the *raytrace* case that has the largest overhead.

Table 1. A performance comparison of Fast Models, COREMU, native execution and the VIRA approach on SPLASH-2 test cases.

	Speed-up (x)		
program	**VIRA/ Fast Models**	**VIRA/ COREMU**	**VIRA/ Native**
barnes	22.59	7.49	0.92
fmm	28.05	13.11	0.94
ocean	27.46	34.83	0.71
raytrace	14.54	20.08	0.53
water	42.53	8.41	0.62
cholesky	17.91	9.26	0.59
fft	77.68	47.89	0.76
lu	11.73	2.10	0.62
radix	37.12	2.55	0.59

2) Host Peripherals as HACs

To test the performance for mapping HACs onto host peripheral devices, we choose a user-level rendering program displaying 1800 frames on a 640 x 480 resolution ARM® PrimeCell® CLCD (Color LCD Controller).

Fast Models uses its own CLCD model, while the VIRA approach simulates the CLCD as an HAC through the hardware display interface of Cubietruck. Nevertheless, due to the fact that

both the rendering code and the HAC on the VIRA approach use the same physical memory and present no extra data passing overhead, the test results show that the VIRA approach still performs 6.2 times faster than the Fast Models (20.88 vs. 129.39 seconds) on this CLCD case.

We also test JPEG decoding using Zynq®-7020 FPGA as a baseline. The Zynq® JPEG decoder works at its maximum operating frequency (synthesized to be 83 MHz) and the VIRA approach offloads the jpeg decoder to the decoder device on Cubietruck. The test results show that Zynq® decodes at 166M pixel/second to memory while the VIRA approach achieves 267.46M pixel/second. In other words, the VIRA approach is 1.6 times faster Zynq®.

B. *Full System Performance Analysis*

To show the effectiveness of the VIRA approach, an MJPEG decoder system is tested to understand how the bus contentions affect the decoding process. The decoder is designed to decode 8 Mega Pixels at 100 frames per second (fps).

We compare the results of two different timing models. The ST (static-trace) model estimates contention delay by pre-computing in static time the average contention delay from previously executed traces. As illustrated in Fig. 2, the ST model simply fails to capture bus contention effects, while the VIRA approach faithfully and accurately reflects the effects on the decoding rate results. Particularly, we observe that when the average bus utilization rate increases to 27% and 72%, the VIRA approach accurately reports a drop of the decoding rate to 82 fps and 56 fps.

In Table 2, we compare the performance estimation error rates with a real system, which consists of ARM® GPU, VPU for video encoder/decoder and a 32-bit memory bus. The GPU writes 3D graphic data to memory and the VPU also decodes JPEG files to memory at the same time. The first column lists the target component measurements for comparison. The second column is the measured real bandwidth of each component. The third and fourth columns are the estimated bandwidth and error rate by the VIRA approach, respectively As expected, the VIRA approach has only a 3~6% error rate and is very accurate compared with the real system.

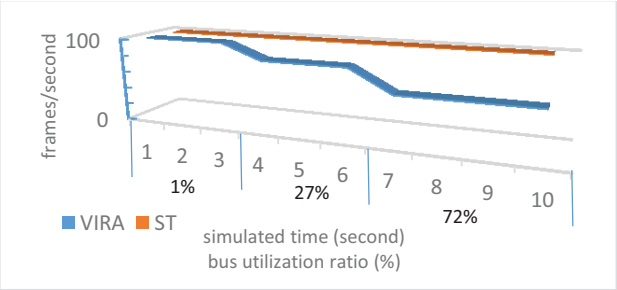

Figure 2: The VIRA approach's dynamic bus contention delay model faithfully captures the bus contention effects, while the ST (static trace) model is insensitive to contention effects.

Note that the VIRA approach has an analytic timing model and does runtime analytic bus contention delay calculations with no need for repeated executions. Therefore, unlike most other non-deterministic approaches, the VIRA approach always produces the same timing results for the same test case and hence does not have statistical variation and standard deviation issues with its test results.

Table 2. Performance estimation compared with a real system.

component	real	VIRA	
GPU	807.9M	784.9M	6.16%
VPU	86.1M	91.4M	-2.95%

V. CONCLUSION

In this paper, we proposed a virtualization-assisted approach—VIRA—for efficient and accurate modeling and performance analysis of virtual prototypes. With the deterministic synchronization process, the analytic timing model and the bus contention model with QoS, the VIRA approach can both deterministically model the inter-component interactions and provide accurate performance estimations for the target system. Although we use virtualization techniques to implement a VIRA virtual prototype, the ideas can also be applied to other hardware-assisted virtual prototypes to increase the accuracy of performance analysis and reduce system modeling effort while keeping the advantages of hardware acceleration.

REFERENCES

[1] Chen, S. Y., Chen, C. H., & Tsay, R. S. "An activity-sensitive contention delay model for highly efficient deterministic full-system simulations." In Design, Automation and Test in Europe Conference and Exhibition. pp. 1-6. 2014.

[2] Vinco, S., Chatterjee, D., Bertacco, V., & Fummi, F. "SAGA: SystemC acceleration on GPU architectures." In Proceedings of the Design Automation Conference. pp. 115-120. 2012.

[3] Sinha, R., Prakash, A., & Patel, H. D. "Parallel simulation of mixed-abstraction SystemC models on GPUs and multicore CPUs." In Design Automation Conference Asia and South Pacific. pp. 455-460. 2012.

[4] Nakamura, Y., Hosokawa, K., Kuroda, I., Yoshikawa, K., & Yoshimura, T. "A fast hardware/software co-verification method for system-on-a-chip by using a C/C++ simulator and FPGA emulator with shared register communication." In Proceedings of the Design Automation Conference. pp. 299-304. 2004.

[5] Chung, E. S., Nurvitadhi, E., Hoe, J. C., Falsafi, B., & Mai, K., "PROToFLEX: FPGA-accelerated hybrid functional simulator." In Parallel and Distributed Processing Symposium. pp.1-6. 2007.

[6] Chiou, D., Sunwoo, D., Kim, J., Patil, N. A., Reinhart, W., Johnson, D. E. & Angepat, H. "FPGA -accelerated simulation technologies (fast): Fast, full-system, cycle-accurate simulators." In Proceedings of the International Symposium on Microarchitecture. pp. 249-261. 2007.

[7] Tan, Z., Waterman, A., Avizienis, R., Lee, Y., Cook, H., Patterson, D., & Asanović, K. "RAMP gold: an FPGA-based architecture simulator for multiprocessors." In Proceedings of the Design Automation Conference. pp. 463-468. 2010.

[8] Tan, Z., Waterman, A., Cook, H., Bird, S., Asanović, K., & Patterson, D. "A case for FAME: FPGA architecture model execution." In ACM SIGARCH Computer Architecture News. pp. 290-301. 2010.

[9] Wang, Z., Liu, R., Chen, Y., Wu, X., Chen, H., Zhang, W., & Zang, B. "COREMU: a scalable and portable parallel full-system emulator." In ACM SIGPLAN Notices, 46(8). pp. 213-222. 2011.

[10] Bammi, J. R., Kruijtzer, W., Lavagno, L., Harcourt, E., & Lazarescu, M. T. "Software performance estimation strategies in a system-level design tool." In Proceedings of the eighth international workshop on Hardware/software codesign. pp. 82-86. 2000.