GI-Fachgruppentreffen 2013 - Proposed Talk:

SimuBoost: Scalable Parallelization of Functional System Simulation

Marc Rittinghaus Konrad Miller Marius Hillenbrand Frank Bellosa

System Architecture Group
Karlsruhe Institute of
Technology (KIT)
firstName.lastName@kit.edu

Full system simulation allows simulating an entire physical machine on top of a host operating system (OS) and thus provides a powerful foundation to study the runtime behavior and interaction of computer architecture, operating systems and applications [1, 7, 11]. Since the entire execution environment in such a system is virtual, every operation carried out can be inspected easily. In contrast to the instrumentation of applications or the OS, analysis with a simulator does not influence the simulated machine's state and thus can be arbitrarily complex in time and space without distorting measurements. The focus of *functional simulation* lies in preserving the functional correctness of the executed target system. Detailed processor- and device state models are not subject to functional simulation.

A well-known limitation of full system simulation is the low execution speed offered by current simulators. The slowdown incurred by functional simulation compared to hardware-assisted virtualization is significant (31x-810x on average). That quickly renders functional simulation impractical for long-running workloads (e.g., 50 days for SPEC CPU2006). Moreover, the execution time is very sensitive to additional overhead caused by analysis logic. *Representative sampling* [8] can reduce the run-time overhead by limiting complex analyses to short time frames that are representative for the analyzed workload. However, an initial functional simulation to identify such intervals is still needed and the accuracy achievable with this technique also heavily depends on sufficient phase behavior in the workload, which is not always present [10]. Moreover, in some scenarios (e.g., analysis of memory duplication) limiting the observation window is not an option. An acceleration technique to enable full-length analyses of long-running workloads is thus desirable.

Contribution. SimuBoost strives to close the performance gap between virtualization and functional simulation to make inspecting the full run-time feasible even for long-running workloads. The core idea is to run the workload that is to be inspected in a virtual machine (VM) managed by a hypervisor such as KVM [4]. At regular intervals the hypervisor takes a snapshot of the full system state (i.e., memory content, device states, HDD data, etc.). The checkpoints then serve as starting points for simulations, enabling to simulate and analyze each interval simultaneously in one job per interval. By transferring jobs to multiple nodes (i.e., CPU cores, hosts, etc.), a parallelized and distributed simulation of the target workload can be achieved. Although each simulation is delayed up to the point when the respective interval is reached in the virtualization stage, the execution speed difference between virtualization and functional simulation enables a parallelization of the simulation, thereby reducing the overall simulation time (see Figure 1).

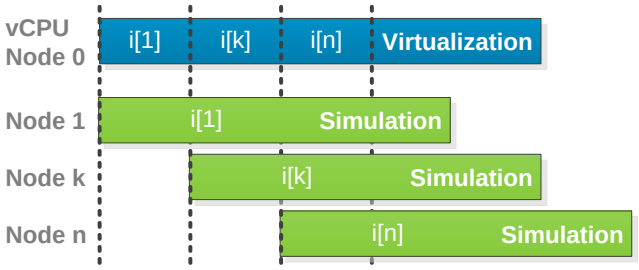

Figure 1: The workload is executed with virtualization. Checkpoints at the interval boundaries serve as starting points for parallel simulations.

The benefit of taking this approach as foundation lies in the fact that it scales with the *run-time* of the simulation: the longer the simulation (including analysis) takes, the more intervals can be extracted and the higher is the degree of parallelization. Opposed to approaches that map simulated CPU cores in a multi-core simulation to real parallelism in the host [2, 5, 9, 12], splitting the simulation into intervals does not limit the degree of parallelization to the number of simulated cores. This way our method is applicable even to single-core simulations. Moreover, since the intervals can be processed independently, it allows distributing the simulation workload across multiple hosts. To match the number of intervals to the available hardware resources the interval length needs to be chosen accordingly.

Results. As we are still working on the implementation of SimuBoost, we cannot provide empirical results, yet. Instead we developed a formal model to describe the speedup and scalability characteristics. SimuBoost can speed up conventional simulation in a realistic scenario (parameter-wise) by a factor of 84, while delivering a parallelization efficiency of 94% according to the model.

Related Work. The division of simulation time employed by our approach has already been proposed for accelerating micro-architectural simulation. Nguyen at al. proposed to use trace-driven simulation and to split the trace into separate time intervals that–in a second step–can be simulated in more detail simultaneously [6]. To generate the underlying instruction trace a preceding recording phase with functional simulation is utilized. Equally targeted at micro-architectural simulation, DiST [3] enhances the approach by providing a robust method to cope with the necessary model warm-up phase at the interval beginnings. SimuBoost differs from these works in that it accelerates *functional simulation*, the mode that is used by all other approaches to generate checkpoints or traces. SimuBoost thus has to cope with new challenges, foremost the non-determinism in the execution with hardware-assisted virtualization.

Talk. The proposed talk will cover the concept behind SimuBoost and provide a more detailed insight into the challenges regarding non-deterministic machine execution that arise when basing functional simulation on checkpoints generated by hardware-assisted virtualization. It will further point out current limitations of SimuBoost and give an outlook on future research directions.

1. REFERENCES

[1] L. Albertsson et al. Using complete system simulation for temporal debugging of general purpose operating systems and workloads. MASCOT, 2000.

[2] J. Ding et al. Pqemu: A parallel system emulator based on qemu. ICPADS. IEEE, 2011.

[3] S. Girbal et al. Dist: A simple, reliable and scalable method to significantly reduce processor architecture simulation time. volume 31. ACM, 2003.

[4] A. Kivity et al. kvm: the linux virtual machine monitor. volume 1. Linux Symposium, 2007.

[5] R. Lantz. Fast functional simulation with parallel embra. Citeseer, 2008.

[6] A. Nguyen et al. Accuracy and speed-up of parallel trace-driven architectural simulation. IPPS. IEEE, 1997.

[7] M. Rosenblum et al. Using the simos machine simulator to study complex computer systems. *TOMACS*, 7(1), 1997.

[8] T. Sherwood et al. Automatically characterizing large scale program behavior. volume 30. ACM, 2002.

[9] K. Wang et al. Parallelization of ibm mambo system simulator in functional modes. *SIGOPS*, 42(1), 2008.

[10] V. Weaver et al. Using dynamic binary instrumentation to generate multi-platform simpoints: Methodology and accuracy. *HiPEAC*, 2008.

[11] C. Won et al. A detailed performance analysis of udp/ip, tcp/ip, and m-via network protocols using linux/simos. *High Speed Networks*, 13(3), 2004.

[12] H. Zeng et al. Mptlsim: A cycle-accurate, full-system simulator for x86-64 multicore architectures with coherent caches. *SIGARCH*, 37(2), 2009.