System Responsiveness Verification of large Multi-Processor System Configurations using Micro-Benchmarks and a Multi-Level Analysis
Dr. Ralf Winkelmann, Edward Chencinski, Hanno Eichelberger, Michael Fee, Carsten Otte, Christoph Raisch
IBM® Systems

Abstract
Computer system development necessitates design changes to incorporate new technology. These changes can have unintended negative side effects, such as statistically significant system responsiveness issues. We architected, implemented, and executed a flexible environment that enables us to identify and address these issues in pre- and post-silicon simulation environments (software simulator, hardware accelerator, real hardware). A deterministic Linux™ kernel based execution environment has been created to implement and run micro-benchmarks targeting multi-processor coordination test scenarios. Sophisticated multi-hierarchy analysis and predication tooling were invented. We successfully applied this method to multiple generations of IBM Mainsframe systems creating a benchmarking history, thereby providing insights for the creation of significant system level improvements.

Overview
• Multi processor coordination features that are addressed with our methodology strongly depend on system components playing well together in concert.
• Critical behaviour occurs via interaction between the components on the microarchitectural level.
• Therefore, a “full” sized cycle accurate hardware model has been chosen for the pre-silicon environment.
• A complete system running in production mode at customer sites is beyond our pre-silicon capabilities, instead we run Linux bare metal on the underlying hardware.
• Pre-silicon experiments are constrained by runtime and model size – post-silicon allows us to go beyond those limits.
• Running Linux bare metal on our hardware with micro-benchmarks on top is supported for both pre-silicon and post-silicon execution with only minor changes.
• Specific approaches, such as, “stop_machine”, are used to prevent us from measuring unintended artifacts.
• We have a hierarchical approach that enables us to analyse and debug the system efficiently on various levels of abstraction.

Production vs. Simulation Environment
The Figure on the left side illustrates an example of a production environment on a mainframe computer. Real customer environments are usually more complex, but here we use a less complex example to show the difference to our Bare Metal Linux environment. For our low-level analysis and debug of system responsiveness micro-benchmarks we need to run on a cycle accurate model to study the interaction between code and hardware in detail. Reducing the complexity in the environment and in the model size is a key enabler for success. Therefore, the Firmware code and the hypervisor are not part of our environment (see Figure on the right side). Instead, Linux runs directly on the underlying hardware.

Pre-Silicon Environment
Runtime environment
Simulation traces and messages
Test case result data (RAW events)
Test cases and command line parameters
Instruction execution traces (IET)
All event trace (AET)
Hardware Model and initialization

Bare Metal Linux
with test cases in kernel space
constraint run environment

Microbenchmark Example (Semaphore Locks)

Microbenchmark Flow
• Critical sections are executed in sequence
• Fight to get lock in between

Algorithm of test case
Statemachine transitions of events

Hierarchical Debugging

Microbenchmark Execution

Events

Instruction Traces

Signal Traces

Results

Results of executing microbenchmark using different placements of 32 CPUs

Results with/without false sharing from other CPUs in critical section

*Trademark of IBM in USA and/or other countries
**Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.