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

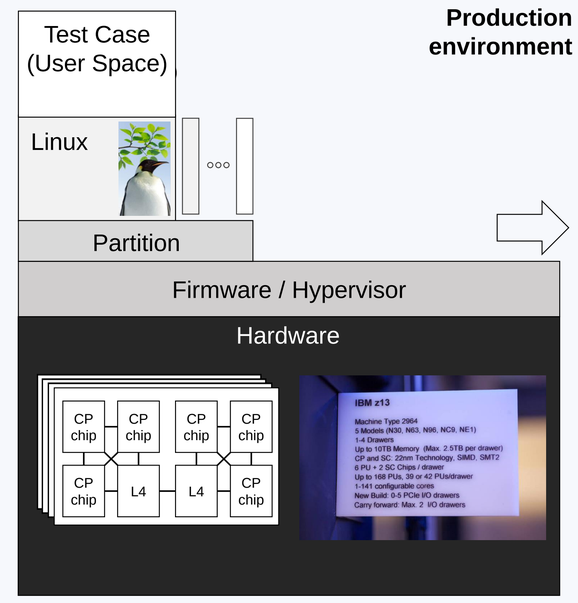
Abstract

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 Mainframe systems creating a benchmarking history, thereby providing insights for the creation of significant system level hardware 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 constraint 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 level of abstraction.

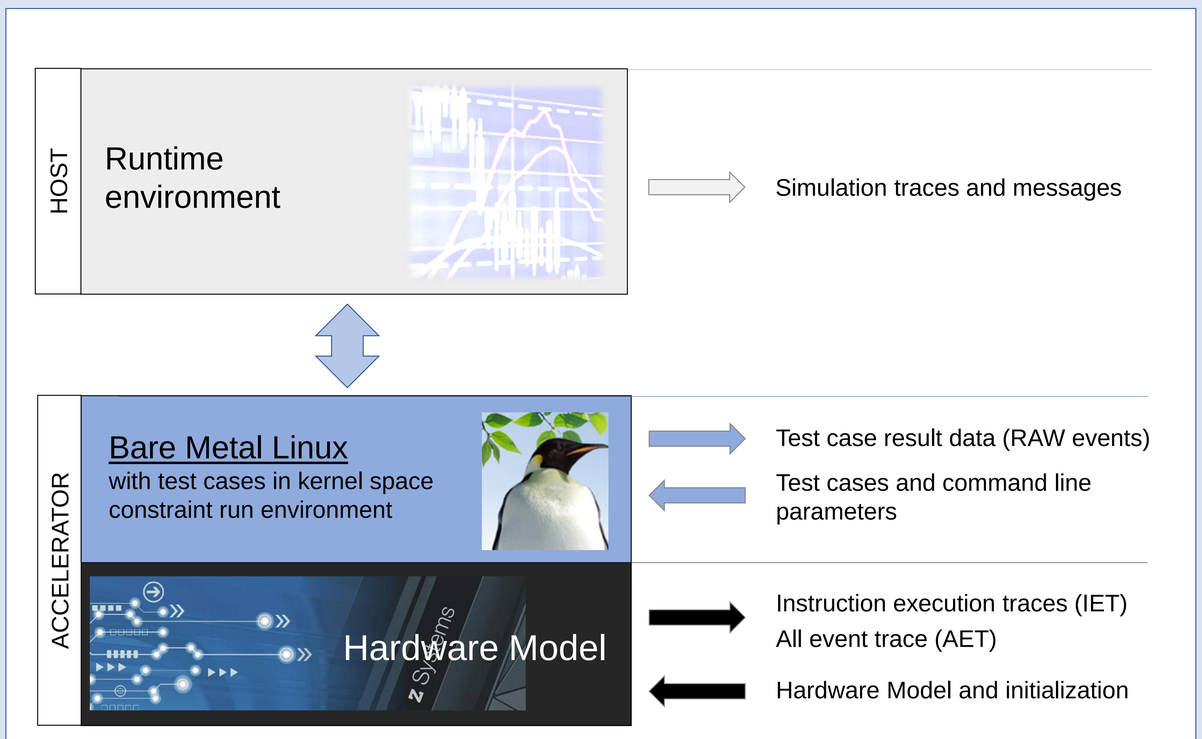
Production vs. Simulation Environment



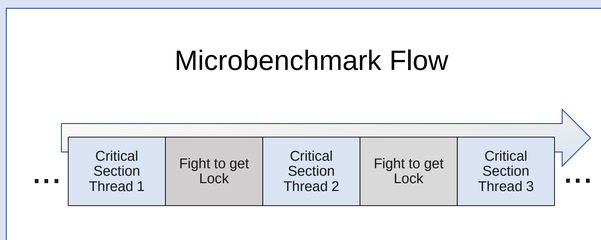
Bare Metal Linux environment
- Reduce complexity
- Reduce model size
- Avoid measurement artifacts

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.

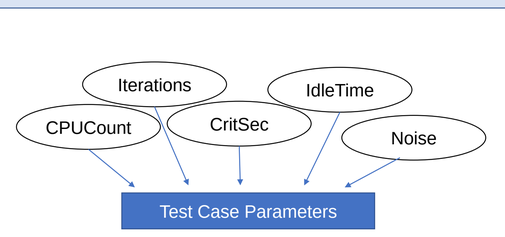
Pre-Silicon Environment



Microbenchmark Example (Semaphore Locks)



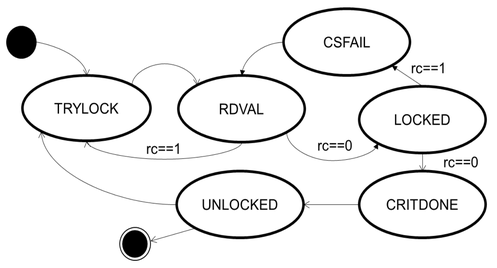
- Critical sections are executed in sequence
- Fight to get lock in between



Microbenchmark is parametrizable

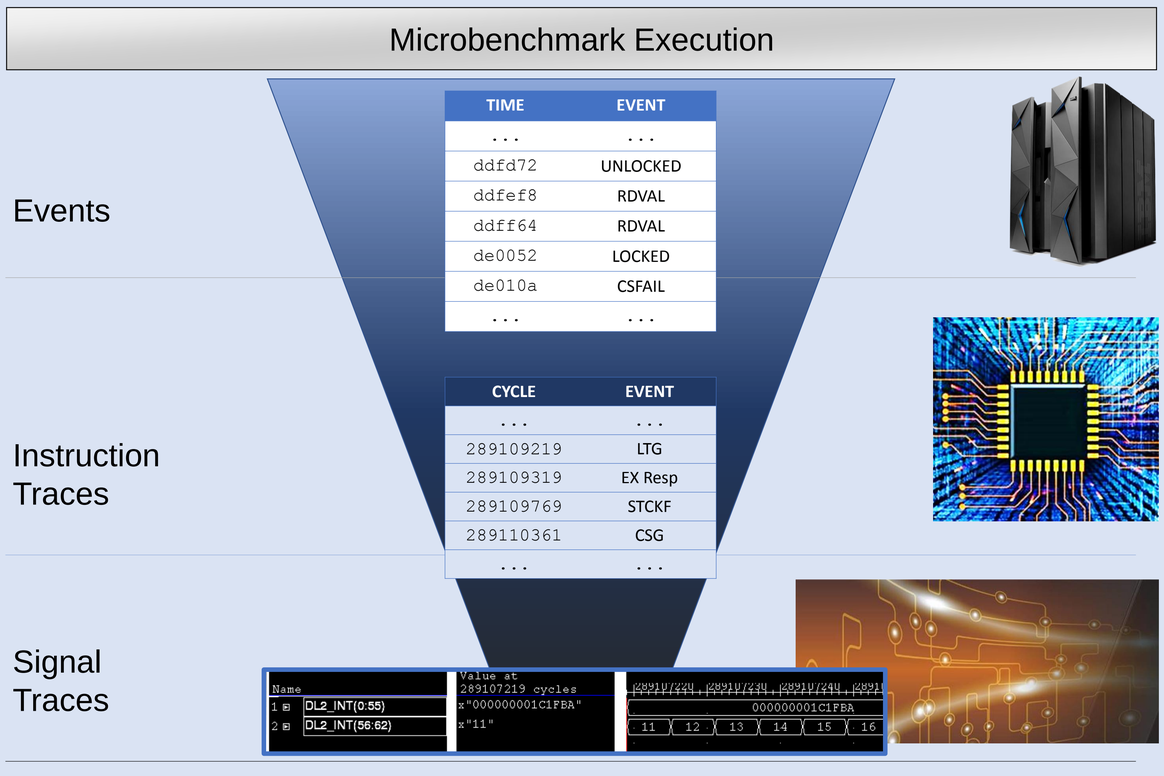
```
Require
  CRITSEC                  Duration of critical section
  ITERATIONCOUNT           Count of iterations
  IDLETIME                 Idle duration after critical section
  NOISE_LOAD, NOISE_STORE  Specifies the noise
  lock                     Variable which represents the lock
  n_start, tmp             Temporary variables

1    for i=0 to ITERATIONCOUNT do
2          rc=LTG;                          // RDVAL
3          if rc==1 goto 2;
4          rc=CSG;                          // LOCKED
5          if rc==1 goto 2;                 // CSFAIL
6          wait(CRITSEC);
7          lock=0;                          // UNLOCKED
8          n_start=time();
9          if NOISE_LOAD:
10               while (time()-n_start<IDLETIME) tmp=lock;
11         else if NOISE_STORE:
12               while (time()-n_start<IDLETIME) lock=tmp;
13         else:
14               wait(IDLETIME);
```

Algorithm of test case



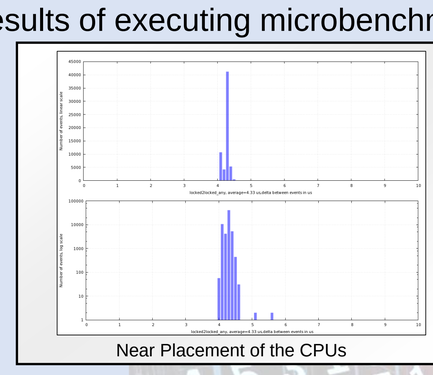
Statemachine transitions of events

Hierarchical Debugging

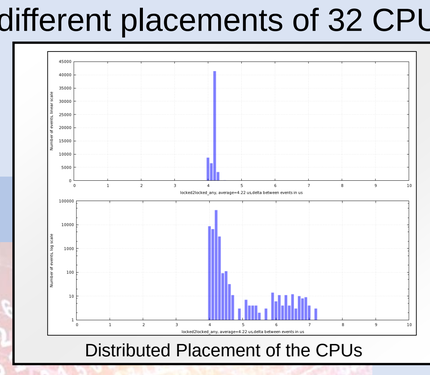


Results

Results of executing microbenchmark using different placements of 32 CPUs

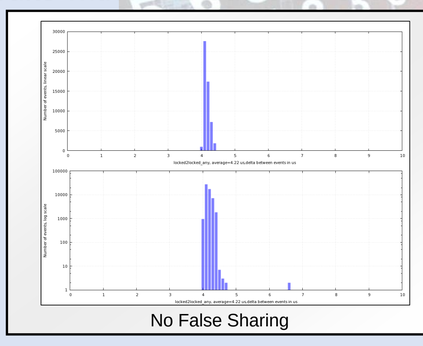


Near Placement of the CPUs

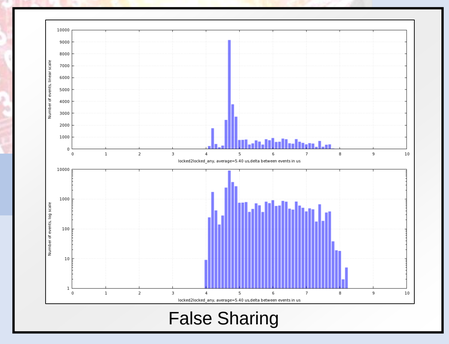


Distributed Placement of the CPUs

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



No False Sharing



False Sharing

*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.