SOME EXPERIENCE FROM THE DEVELOPMENT OF A SIMULATOR FOR A TELECOM CLUSTER (CPPemu)

Mikael Bergqvist, Jakob Engblom
Virtutech AB
Norrtullsgatan 15
SE-113 27 Stockholm, Sweden
+46 8 6900746
mikael/jakob@virtutech.com

Mikael Patel
Ericsson AB
Datalinjen 4
SE-581 12 Linköping, Sweden
+46 13 284665
mikael.patel@ericsson.com

Lars Lundegård
TietoEnator
Kanikenäsbanken 12
SE-651 15 Karlstad, Sweden
+46 54 294185
lars.lundegard@tietoenator.com

Abstract

This paper presents lessons learned when developing and implementing a simulator for Ericsson's Connectivity Packet Platform, the basis for most Ericsson 3G systems. CPP is a heterogeneous cluster with more than 20 types of processor boards with 1-5 MLOC of code running on each board, and several processors per board.

The simulator, called the CPP Emulator (CPPemu), has been used mainly for embedded software development. CPPemu runs the unmodified software from the real CPP. Thus, CPPemu can replace a real CPP node.

Designing and implementing CPPemu was a major undertaking, which required a simulation engine with high performance supporting roughly the same scalability as the platform itself. Multiple hosts are to simulate large configurations and several improvements to simulation technology were necessary to obtain a useable simulator.

The project lead-time was approximately one year and CPPemu is now being deployed to testing and verification departments as well as application development projects.

CPPemu has brought several benefits and spin-offs. Concurrent software behavior in the cluster is much easier to study and understand. The simulator has been used to support development of new boards, reducing time to market. There has also been a reduction in test system hardware costs.

Keywords

Industry Experience, Simulation Technology, Embedded Software, Heterogeneous Cluster Simulator, Integration & Verification, Cluster Platform Software Design.

1. Introduction

The Ericsson Connectivity Packet Platform (CPP) is a platform for high availability 2G and 3G applications, intended for ATM- as well as IP-based nodes. It is used as a basis for several key components such as Radio Base Stations (RBS), Radio Network Controllers (RNC), and Media Gateways (MgW) in the 3G mobile networks, both WCDMA and CDMA2000.. [1][2].

As for all telecommunications equipment the cost of developing and testing software for CPP-based products is high, Some of the major cost factors are:

- The hardware itself
- Managing the hardware and different hardware configurations
- Hardware availability
- Time-consuming and complex trouble-shooting
- The systems are of telecom grade

Simulation offers a way to mitigate several of these cost factors, and Ericsson has good experience in using simulation and emulation technology in the development and testing of telecommunications equipment.

The initial cost to buy test equipment for a telecom development project is high, since a large amount of custom telecom grade hardware has to be acquired Simulation enables the use of regular development workstations instead of actual hardware, which can give significant cost savings.

Managing numerous hardware configurations is costly, especially for a product intended for many platform versions and developed in parallell projects.. With a simulator, the hardware configurations are expressed using software and can thus be managed in the same way and at the same time, as the target software configurations.

Telecom verification projects often suffer from not having hardware available on time. It is not unusual that software and hardware are developed in parallel, and when the hardware becomes available, it is early series hardware available in limited quantities and carrying extra costs. The simulator can both be used to enable the parallel development, as well as to provide early hardware at no extra cost compared to regular deployment.

Testing projects often have peaks where they need to execute a large number of test cases and as a consequence they need access to many “test channels”.

This peak is often during the functional testing stage and the availability problem situation get worse for every new release of a product.

Sharing test equipment between testers is typically not feasible, since the testers will interfere with each other. Our experience is that using test equipment in work shifts does not solve the problem either, primarily because of user specific test site preparations. It could also be the case that a tester can not continue debugging the next day because someone else has reconfigured the site. With simulation, each tester can be given their own test system, avoiding interference and facilitating efficient work.

The trend to use a more iterative development model increases the demands on having easy access to test equipment since regression tests are supposed to be executed frequently. These availability issues have a direct impact on the development cost and on time to market.

Debugging and trouble shooting is cumbersome on con current, multi-tasking, real-time systems like our telecom systems.

Reproducibility is essential for testing and debugging, but the system is by its nature non-deterministic. If the system is modified with extra logging or debug information, probe effects are likely to occur that change the behavior of the system, often masking the problem.

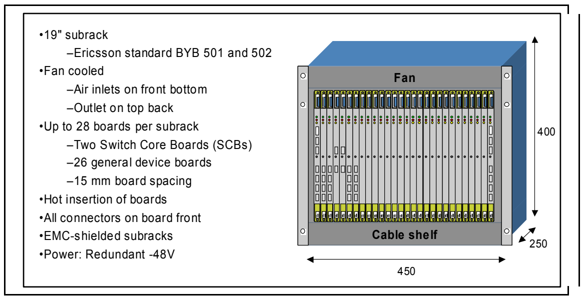
The time consuming task to reproduce an error is something that every tester and designer has experienced. There are many developers that can testify about debugging sessions where the failure in operation could not be reproduced in his/her environment.

The rest of this paper will present the CPP hardware in more detail, the simulation technology used, and our experience from the development, deployment, and use of the CPPemu simulator.

2. The connectivity packet platform (CPP)

CPP is a platform from which several different products are created. The platform encompasses both custom hardware and a significant software stack. CPP is the base from which it is possible to develop anything from an Asynchronous Transfer Mode (ATM) switching network node, a Radio Base Station (RBS), to a Radio Network Controller node (RNC) for the UMTS network [1][2].

The software side of CPP consists of a distributed, fault-tolerant real-time system based on the ENEA OSE operating system. It includes an element management system built on UML based description of Managed Objects (MO), Web technology, the Common Object Request Broker Architecture (CORBA), and the IP-based inter-ORB protocol (IIOP).



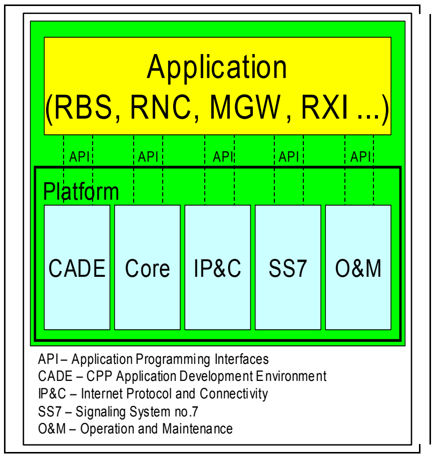
•19" subrack
–Ericsson standard BYB 501 and 502
•Fan cooled
–Air inlets on front bottom
–Outlet on top back
•Up to 28 boards per subrack
–Two Switch Core Boards (SCBs)
–26 general device boards
–15 mm board spacing
•Hot insertion of boards
•All connectors on board front
•EMC-shielded subracks
•Power: Redundant -48V
Fan
Cable shelf
400
250
450

Figure 1. A CPP Sub-rack

Figure 1 shows the basic unit of the CPP hardware, the subrack (three subracks fit in a single cabinet). The boards in the subrack are connected over a redundant ATM backplane, and the switching for the backplane is performed by boards in the subrack (the Switch Control Boards, SCB). The most common processor type is PowerPC, but there are also many DSPs and other processors used.

There are more than twenty different types of boards and most are available in a number of revisions. Old board revisions must be maintained, as they are still in use in customer installations. The lifespan of telecom equipment is quite long and each installation will evolve over time as boards are replaced and added to the configuration.

The CPP platform also provides methods and tools for developing custom node software and hardware, such as radio algorithms, call control programs and device cards. The CADE (CPP Application Development Environment) is a development environment for both application and CPP platform software.



Application
(RBS, RNC, MGW, RXI ...)
API
API
API
API
API
Platform
CADE
Core
IP&C
SS7
O&M
API – Application Programming Interfaces
CADE – CPP Application Development Environment
IP&C – Internet Protocol and Connectivity
SS7 – Signaling System no.7
O&M – Operation and Maintenance

Figure 2. CPP System Areas

2.1 Standard CPP Boards

The CPP platform provides a set of basic CPP board types which are required by all or most applications built on CPP. This set of standard boards can be extended with boards customized for a particular application.

All CPP boards feature a *Board Processor Module* (BPM) and a *Switch Port Interface Module* (SPIM). The BPM module handles basic board status indication functionality and Product Identity (PID) information. The SPIM connects the board to the ATM backplane. The SPIM and BPM together form the *Device Board Module*, DBM, which is a custom ASIC used on most boards.

Standard board types include:

- *Switch Core Board*, SCB. Provides ATM switching within a single subrack in a cabinet.
- *Switch Extension Board,* SXB. Provides switching between subracks in large configurations where the capacity of the SCB is not sufficient.
- *General Processing Board*, GPB. General processing board featuring a single PowerPC processor with memory and Flash disk storage. It has Ethernet and serial communications available for management purposes. It uses a PowerQUICC-series communications processor to connect to the ATM backplane, in lieu of the standard DBM.
- *Timing Unit Board*, TUB. Provides a high-quality clock signal for the system.
- *Exchange Terminal Boards*, ET-XX. This is a large family of boards providing interfaces for traffic over various types of transmission lines. Different versions of Exchange Terminals are needed in order to implement adaptations to different physical media. To simplify this, the Exchange Terminal boards are designed with a modular hardware structure. Examples are ET-M4 for 155 Mbps ATM, SDH, and SONET, and ET-M1 for 2 Mbps ATM or TDM traffic.
- *Special-Purpose Boards*, SPB. These boards provide extra processing power for certain applications. The SPB2 board contains six PowerPC processors, for example.
- *Media Stream Board*, MSB. Contains a large number of DSP processors for high perfomance media and signal processing. The latest generation has 16 DSP processors, plus a PowerPC to manage and set up processing tasks.

Each board is typically available in a number of hardware versions, and different versions can coexist in the same configuration. For example, the GPB is now in version 5.Version 6 is under development, while version 3 and 4 are still in significant use.

2.2 Application-Specific CPP Boards

CPP provides a hardware device design platform containing a basic board design and core support software. The device designer only has to design the software and the hardware specific to his application, and the resulting board can be used in the CPP platform just like the standard boards.

2.3 Hardware Redundancy

All boards are deployed in redundant, master-backup pairs. Thus, all CPP configurations use an even number of boards, and no configuration uses a single board of any type. The software running on CPP ensures that failed boards are diagnosed, possibly restarted, and that system operation continues even as boards are added, removed, swapped, or upgraded.

3. Simulation Technology

CPPemu was built using the Simics simulator from Virtutech as the base simulation infrastructure and component source. Simics is a *full-system simulator,* i.e., a simulator that simulates a computer system at such a level of detail that the complete software stack from real systems can run on the simulator without modification. A full system model includes models of the processors, memories, IO devices, disks, and interconnections of the real system. Figure 3 gives a conceptual view of full-system simulation.

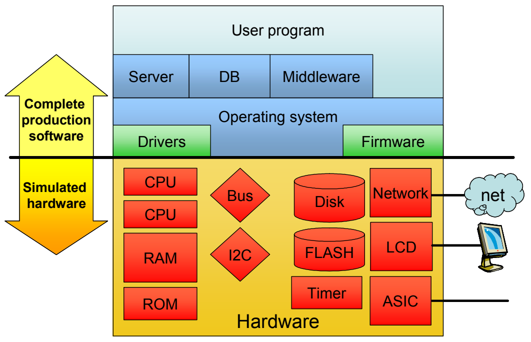


Figure 3. Overview of full-system simulation

At heart, Simics is an event-driven simulator where processors are treated specially for performance reasons. All I/O and other peripheral devices in a system (and thus all interconnects between systems) are simulated in a *transaction-based* style. The main design goal of Simics is high simulation speed, in order to scale to simulating systems with many processors and to still run the target software sufficiently.

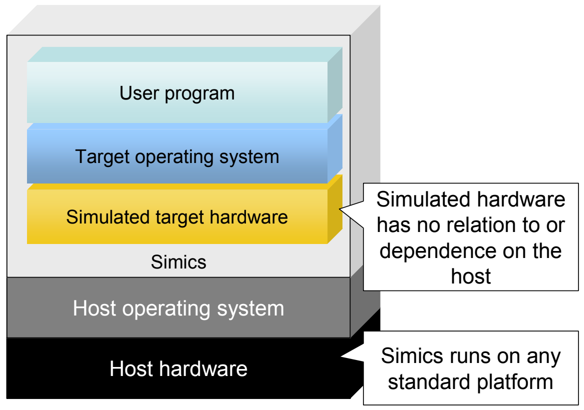

User program
Target operating system
Simulated target hardware
Simics
Host operating system
Host hardware
Simulated hardware has no relation to or dependence on the host
Simics runs on any standard platform

Figure 4. Simics provides total virtualization

As illustrated in Figure 4, Simics is designed to provide total virtualization of the target system, with no dependence on the host machine. For CPPemu, this was important as it ensured that CPPemu executed code in the same way on Windows, Linux, and Solaris hosts. It also meant that network addresses for the cluster cards could be set arbitrarily and that many instances of the same configuration could be run at the same time on the same host or network of hosts.

Simics can execute any number of simulated processors and boards within a single Simics process. When simulating multiple processors Simics uses *round-robin time slicing*. Each node gets to execute a certain (simulated) time before switching to the next node.

Each Simics simulation can contain any number of computation nodes, each containing any number of processors, connected over any number of networks. The precise network topology of the simulated system is mirrored in Simics, including multiple types of networks (ATM mixed with Ethernet, for example). As shown in figure 5, this works just like the real system from a software perspective.

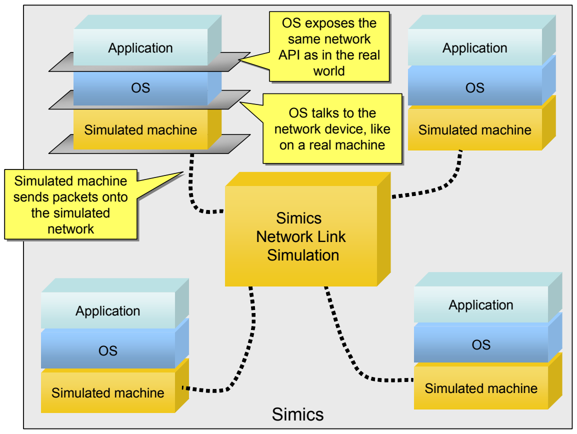

Application
OS
Simulated machine
OS exposes the same network API as in the real world
Application
OS
Simulated machine
OS talks to the network device, like on a real machine
Simulated machine sends packets onto the simulated network
Simics Network Link Simulation
Application
OS
Simulated machine
Application
OS
Simulated machine
Simics

Figure 5. Principle sketch of Simics network simulation

Simics uses a completely virtual time base, which means that the relative execution speed of network nodes is respected in simulation. For example, in a scenario mixing 50 MHz board controller processors with 800 MHz main processors, the main processors nodes will execute code 16 times faster (assuming that both set the same instructions-per-cycle parameter).

This requires that a common time based can be found across all processors, and this computation had to be updated as a result of the work on CPPemu. Mixing processors with frequencies like 738 MHz, 638.10 MHz, and 77.76 MHz did reveal a problem in Simics which had not been detected before the development of CPPemu.

3.1 Simulation Configuration

The CPP emulator follows the structure of the real CPP system, with simulation models for boards connected over a simulated ATM backplane.

A large number of boards are simulated in CPPemu, each board built from the components as discussed below.

When configuring a CPPemu simulation, users select the types of boards to use, the board's positions in the rack, and the software to run on each board. The CPPemu configuration tool then sets up a series of Simics scripts to create and start the simulation.

CPPemu allows for simulation of both standard and application-specific boards. In order to provide a working system, a necessary set of standard boards had to be simulated.

3.2 Simulated Components

The CPPemu simulator is built up using a library of simulated hardware components. There are today more than 150 different simulated hardware elements used in CPPemu, in categories like processors, integrated communications processors, memories, disks, flash disks, Ethernet, back-plane, and custom FPGAs.

The heterogeneity in terms of processors is greater than any other simulated system we have encountered. The current CPPemu simulates more than 10 processor types in the PowerPC family, together with several communications processors. There are also Texas Instruments C64 DSPs. Most configurations contain most of these processor types in varying numbers.

The number of simulated hardware elements on each board ranges from 20 to 50 with an average of around 30. A large amount of the effort on developing a cluster simulator like CPPemu is spent on integration and verification of the boards.

Approximately 90 of these elements are common-of-the-shelf components provided with Simics. The other components have been developed within the CPPemu project and are in most cases Ericsson-specific.

The reuse of simulated hardware elements is very high within CPPemu, as the real hardware also employs reuse of components. An important example of reuse is the DBM, which is found on almost all boards as discussed above.

3.3 Emulated Boards

Not all board types need to be simulated, by simulated we mean that an actual processor model is executing the actual software for that board. Instead, an emulation of a board can be used, where a simple simulation model provides the same behavior towards the rest of the system as a real board would have, but at a much lower computational cost. The board is emulated at the interface to the ATM backplane.

One such example is the TUB. In most cases, an emulated model is sufficient, with the result that we do not need to simulate the code running on the PowerPC inside the DBM on the TUB. This speeds simulation, simplifies simulator development, and reduces the memory consumption of the simulator.

Note that we have found some test cases where the TUB needed to run its software, and thus a simulation model is also available.

3.4 Simulation Performance Enhancements

On average the simulated cluster runs 3-5 times slower than the real hardware on a single host at a low concurrency load (i.e. all boards are not active). This results from several performance-enhancing techniques used in Simics.

Processors in Simics are simulated using *just-in-time compilation techniques (JIT compilation)*, converting target code to native host instructions. Compared to a simple interpretation of each instructions, this speeds simulation by a factor of 5 to 10 typically, and much more in fortuitous cases.

The simulation time is totally dominated by the execution of code on the processors. The device simulation contributes very little, and can mostly be ignored for optimization purposes.

Simics also takes advantage of the fact that in many cases most boards in a CPPemu run are idle or very lightly loaded. In such circumstances, Simics *skips idle time*. When a processor is idling waiting for the next interrupt to happen (for example, putting a PowerPC processor into doze mode or executing HLT on an x86 processor), Simics will skip ahead in time for that processor until the next interesting event happens. In CPPemu, we also have to perform pattern-recognition on executing code in order to catch operating-system idle loops, which do not explicitly activate sleep mode.

To facilitate very large target system memories, as found in the CPP system, Simics uses *lazy memory allocation*. Only memory actually allocated is represented on the host. Also, when needed, Simics swaps target memory out to the host disk on a least-recently-used basis. This makes it possible to simulate target systems with far larger memories than the host [9], typically this can amount to allowing for simulated target memory of 20GB while running on a host with 2GB DRAM [9].

3.5 Host Machine Considerations

We have found that the performance of the simulator is greatly dependent on the host machine used. SPARC servers typically have much slower processors than contemporary PCs, but are able to better handle large memory configurations. In most cases, however, performance is CPU-bound, and using a fast PC is the best host machine. Within the PC space, the type of the processor has a large impact on simulator performance.

- The microarchitecture of the AMD Athlon64 and Opteron, as well as Intel's Pentium-M and Core/Core 2 processors have proven very suitable for Simics. The Intel Pentium 4 architecture is not to recommend, since its deep pipelines does not handle Simics-like code well.
- The size of the level 2 cache is critical, since the working set of the JIT-generated code can get large when there are many processors involved. The on-chip 4 MB L2 cache of the recent Intel Xeon 5000-series chips has been very beneficial to Simics performance on CPP.

3.6 Distributed Simulation

To increase performance beyond what is possible when using a single host to execute the simulation, we have used *distributed simulation*. As illustrated below in Figure 6, distributed simulation in Simics is implemented using a synchronization service called *Simics Central* [9]. The simulated system is divided up into multiple Simics processes, at points where communication between processors is performed using networks and not shared memory. For CPPemu, this naturally maps to distributing out the boards in the simulated system among the Simics processes.

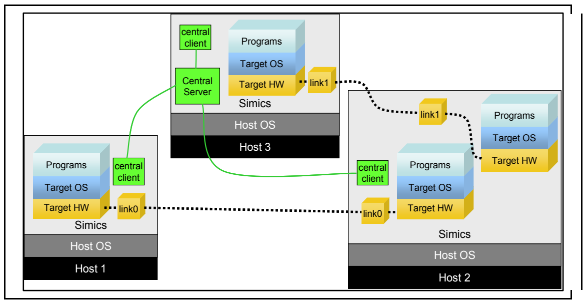


Figure 6. Distributed simulation using Simics

Achieving performance with distributed simulation is non trivial.. At this point (Summer 2006) we have not yet reached linear scalability.

Typically, an additional host provides about 20% performance improvement compared to running the entire simulation on a single host. This can be explained by the need for communication between the hosts but is also due to the implementation of the simulation of the ATM back-plane. As ATM traffic is essentially implemented on top of IP when using distributed simulation, several layers of message fragmentation is introduced, giving a high level communication overhead. There are also some unnecessary hand-shakes in the ATM simulation, which serialize simulation that could in principle be done in parallel.

Distributed simulation performance also depends on how well the partitioning of the CPP system across the Simics processes is performed. Since the partitioning is currently static, an unbalanced partitioning will make one host do most of the work and keep the other hosts idle. Since the load hot-spots can vary depending on the workload being run, partitioning requires some intuition and trial. In the best cases we have seen, we have managed to use four processors at about 70% host load, i.e., providing a speed up of 2.8 using 4 host processors.

We are also considering using a homogeneous Linux cluster to avoid the static configuration. This will require some additional research to understand how to synchronize the simulators in a dynamically distributed environment.

4. CPPemu Development History

In early 2004, Ericsson took a decision to initiate a “proof of concept” project with the goal of investigate whether board-level simulation was possible and could deliver the necessary performance to allow replacement of real hardware during System Testing, a.k.a. Integration & Verification (I&V) within Ericsson.

One of the CPP general processor boards (GPB4) was selected as the candidate for prototyping. After evaluating the simulator for the GPB4 board, the CPPemu project was initiated with the goal to be able to simulate a node (cabinet) with all necessary boards.

After approximately a year, the first full system/cluster simulator was delivered and could be used as a replacement of a hardware node. More than 20 different board types were simulated at this time. The software platform, run-time system, real-time operating system with drivers and services could run, unchanged, directly on the cluster simulator.

The goal of the initial project was to increase the testing capacity at I&V departments within Ericsson, but many new uses for the simulator have appeared over time. They range from HW-SW co-design with early SW builds during the design of new boards to early Unit Testing by design organizations. The simulator has also started to support changes in ways-of-working allowing more incremental feature oriented processes and continues development.

5. Benefits of Simulation

CPPemu has let us realize many well-known benefits of simulation, at a scale never before reported in the literature.

5.1 Easier Debug and Trouble Shooting

All telecom systems are fault-tolerant distributed real-time systems, and the majority of current testing and debugging techniques have been developed for sequential (non real-time) programs. These techniques are not directly applic-able to real-time systems, since they disregard issues of timing and concurrency. This means that special real-time debugging techniques and tools need to be used. Our experience is that these debugging techniques and tools are not used in the real world. The major reason is that the threshold to learn them is too high.

A common approach to debugging when a problem is found in testing is to either skip it altogether and just send a trouble report to the design organization, or to ask the design organization for an updated software version that prints more extensive logging information. Once the new version has been obtained, the test is reexecuted and the log sent back to the design organization. This type of trouble-shooting tends to be very time consuming.

With a simulator, debug can be performed in a more straight-forward manner. Since the simulator provides deterministic, reproducible, and controlled behavior along with interfaces to standard debuggers like gdb, a program can be debugged in a manner much more like traditional single-task debug. The main enabler is synchronous stop, where the entire massively concurrent system is stopped, while the user investigates the behavior of some particular program.

In addition, the whole system with all its subparts can be single stepped, without risk for communication pile-up or time-out between subsystems. Execution of the full system can at any time continue at full simulation speed.

It was discovered that logging using the mechanisms used in a real CPP node blocks subsequent logs to appear while examining or retrieving a log. Using the simulator, a special smart breakpoint, that traps the logging mechanism, was developed. This enables the user to view all logs that would be issued, something that is not possible on real hardware. In essence, logs are taken out through a invisible back door in the system.

5.2 Single Build Version

With the increasing size of code in the cluster, with many different sub-systems and load modules, it becomes very difficult to manage and allow multiple build versions for testing and measurements, in addition to the real production build. Thus, using instrumentation for profiling and code coverage becomes virtually impossible since it creates configuration management nightmares.

With CPPemu, we can run the real production build and still gather profiles and code coverage information, without modifying the code or requiring any particular build variants.

The sheer scale of CPP and its applications makes a simulator, which simply models the hardware and runs the regular software, much easier to use and apply than the various host-based simulation schemes and instrumented compilation tools commonly used in developing embedded systems [10][11].

5.3 Non-intrusive Instrumentation & Profiling

Another well-known disadvantage of instrumentation is that it changes the real-time properties and the code size, thus disturbing the properties being observed.

The simulator is by definition non-intrusive, as it runs the production code. It thus avoids the code size change. The real-time properties are simulated with sufficient precision to provide useful data, even if CPPemu does not attempt to provide precise timing information at the clock-cycle level.

Profiling with the simulator can be done on any level from instruction (address) to message passing. This allows replacement of a large set of tooling with the single CPPemu system. We have implemented sampled profiling, where the simulator periodically records the current program counter.

5.4 Increased Observability and Controllability

CPPemu provides much better observability into the workings of a CPP cluster than the real hardware does. There is a lack of good tools for observing multiple boards, processors, and threads. To realize this potential however, we had to implement special scripts and tool connections for the simulator.

Interestingly, the slowdown of the simulator compared to the real system has helped system understanding. Slowing down by a factor of 5 makes it easier to spot problems and understand the performance and timing bottlenecks in the cluster. These had not been studied in detail before due to lack of tooling..

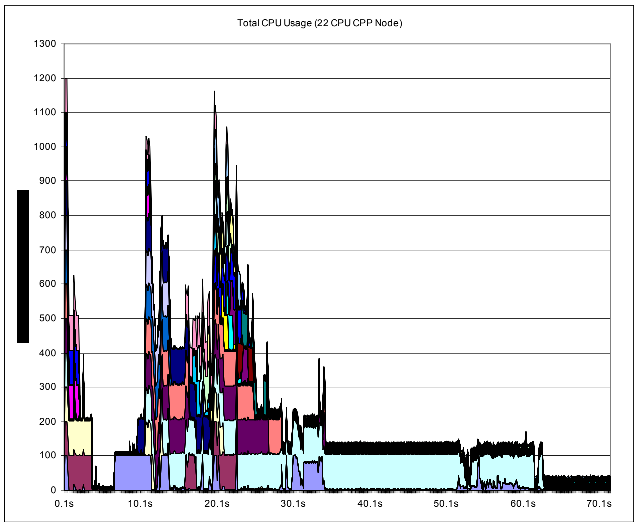


Figure X: Example load measurement

The figure above shows an example of the insight gained using the simulator: it shows the total load on a 22-CPU CPPemu system when booting a base system without any applications loaded. Measuring this on real hardware is impossible, as no load measurements tools can be loaded before the operating system has finished loading.

6. Deployment Issues

Some interesting issues surfaced when we deployed CPPemu to the developers at Ericsson. These are worth discussing, since they are general issues that any other similar simulation solution would suffer from.

6.1 Test System Time-Outs

To system test CPP applications a number of test case generators are used, e.g., TTCN and 3G simulators. Many of these tools have built in real-time time-outs, based on the clock on the computer they are running on. If the system under test does not reply within a certain time, it is considered to have failed the test.

When CPPemu instead of real hardware, the handling of the deadlines needs to be adjusted since the simulator does not run at the same speed as real hardware. It can run faster, when skipping idle time, as well asslower, under heavy load, and the relationship between simulated and real time is not constant. The best solution that we have found is to have the simulator expose a heartbeat signal that external tools can listen to, and express deadlines in terms of number of heartbeats rather than absolute time.

An important observation is that the simulator maintains the same order of events as the real hardware. With the simulator, debug and profiling tools become easier to use for I&V departments.

6.2 Connection to the Real World

Connecting simulated boards inside CPPemu to the real world is a necessary feature of CPPemu. In terms of network interconnectivity, this implies running software with system access to network devices.

Within large organizations System Administrator privileges of the host computers can not be given to all developers, thus special real-network connectivity methods had to be developed for the simulator that would allow connections to the real network without sysadmin privileges. The solution created was based on Network Address Translation over non-privileged communication ports.

6.3 Deployment within Large Organizations

A special challenge is to deploy a tool such as the CPPemu in a large organization (1000+ developers and testers). Both training and support becomes a hard issue and effects ROI and is necessary for achieving the ultimate goal of increasing quality and decreasing project lead-time.

TietoEnator, the main contractor, has put together a special Change Control Board (CCB) and Steering Committee for this Ericsson-internal product. Extensive training material and on-site support is necessary in order to make users accept and become productive with the simulator.

6.4 Adapting Work Methods to CPPemu

CPPemu was designed to provide a transparent replacement for development hardware and test systems. From a technological viewpoint, we have achieved this goal, and CPPemu is being used instead of hardware. However, using a simulator in the same way as you use hardware does not necessarily make the best use of the simulator. If work methods can be adapted to take advantage of some of the idiosyncrasies and unique features of the simulator, much more efficient use is possible.

The OS of the CPP can currently recognize if it is running on the simulator or on real hardware. In this way time consuming hardware tests can be skipped, without compromising the quality of the simulation.

We are still in the progress of trying to take advantage of the possible new ways of working. A typical example of this is to share initiated simulator state using check-pointing instead of having every developer boot their simulator from scratch. This quickly reduces the turn-around time from change to uploading and re-running the test cases.

7. Related Work

Simics is designed primarily to be a development tool for software and hardware designers, and not an execution environment for code. There is a lot of related work.

The use of Simics described in this paper is typical for the industrial use of the simulator, but it has also been used in a very different way as the basis for detailed computer-architecture research simulation solutions such as GEMS[5], SimFLEX[6], Vasa [7], and RASE[8]. Nothing prevents the combination of a large cluster simulation like CPPemu with such detailed studies, allowing computer architecture to be studied in a different context than is usually the case.

Virtual Network User-Mode Linux (VNUML) [12] is a system built on top of User-Mode Linux (UML) to allow the creation of large simulated networks consisting of x86-based Linux nodes. For CPPemu, this solution does not work, as it does not support PowerPC and DSP processors. It also lacks determinism and other software-development features.

Virtualization systems like VmWare [13] or *paravirtualization systems* like Xen [14] can be used to create virtual networks consisting of copies of the host machine, but with potentially different operating systems. Just like VNUML, they are limited to copies of the host processor, and cannot simulate PowerPC processors on an x86 host or vice versa. They are also focused on code execution rather than software development, which was not appropriate for CPPemu.

The *M5* simulation framework [15] is similar to Simics in that it uses full-system simulation to set up networks. However, M5 is intended for computer architecture studies rather than software development. Simics executes code much faster than M5, and M5 is currently limited to a single Alpha target system and the PISA research instruction set.

8. Acknowledgement

Our thanks to Tommy Hill and Johan Preuss, Ericsson AB for their valuable comments and contributes to this work.

9. References

[1] Kling, LG., Lindholm Å., Marklund, L., and Nilsson, G.B. CPP – Cello Packet Platform, *Ericsson Review*, No 2, 2002.

[2] Zhang Z., Heiser F., Lerzer J., and Leuschner H. Advanced baseband technology in third-generation radio base stations. *Ericsson Review*, No 1, 2003.

[3] Lundegård, L. Using a Telecom Platform Emulator, Technical Paper, EuroSTAR 2005, 2005.

[4] Engblom, J., Girard, G., and Werner, G. Testing Embedded Software using Simulated Hardware, *Embedded Real-Time Software (ERTS) 2006*, Toulouse, January 2006.

[5] Martin, M.M.K., Sorin, D. J., Beckmann, B. M., Marty, M.R., Min Xu, Alameldeen, A.R. , Moore, K.E., Hill, M.D., and Wood, D.A. Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset, Computer Architecture News (CAN), September 2005.

[6] Wenisch, T.F., and Wunderlich, R.E. SimFlex: Fast, Accurate and Flexible Simulation of Computer Systems, Tutorial in the International Symposium on Microarchitecture (MICRO-38), November 2005.

[7] Wallin, D., Zeffer, H., Karlsson, M., and Hagersten, E. Vasa: A Simulator Infrastructure with Adjustable Fidelity, Proc. 17th International Conference on Parallel and Distributed Computing and Systems, November 2005.

[8] Davis J., Fu C., and Laudon J.: The RASE (Rapid, Accurate Simulation Environment) for Chip Multiprocessors, Proc. of dasCMP, September 2005.

[9] Engblom, J., Kågedal, D., Moestedt, A., and Runeson, J. Developing Embedded Networked Products using the Simics Full-System Simulator, Proc. 16th IEEE International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC 2005), Berlin, Germany, September 2005.

[10] Möller, A. "A Simulation Technology for CAN-based Systems", *CAN Newsletter*, Dec 2004.

[11] OSE SoftKernel Environment Datasheet, ENEA Embedded Technology, 2004.

[12] Galán,, F., Fernández, D., and de Miguel, T.. "Study and Emulation of IPv6 Internet-Exchange-Based Addressing Models", *IEEE Communications Magazine*, Jan 2004.

[13] VmWare: www.vmware.com

[14] P Barham, B Dragovic, K Fraser, S Hand, T Harris, A Ho, R Neugebauer, I Pratt, and A Warfield. Xen and the Art of Virtualization, *Proceedings of 19th ACM Symposium on Operating Systems Principles (SOSP2003)*, October 2003.

[15] Nathan L. Binkert, Erik G. Hallnor, and Steven K. Reinhardt. Network-Oriented Full-System Simulation using M5, *Proceedings of the Sixth Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW)*, Feb 2003.

[16] Jonas Myhrman and Pierre Svärd: *Studying Fault Injection in WCDMA Base Station Processors Using Simics Simulator*, MSc Thesis, Chalmers Institute of Technology, Department of Computer Science and Engineering, Göteborg, Sweden, 2005