Towards Feedback-Based Generation of Hardware Characteristics

Marcus Jägemar
Sigrid Eldh
Andreas Ermedahl
Ericsson AB
first.last at ericsson.com

Björn Lisper
Mälardalen University
bjorn.lisper at mdh.se

ABSTRACT

In large complex server-like computer systems it is difficult to characterise hardware usage in early stages of system development. Many times the applications running on the platform are not ready at the time of platform deployment leading to postponed metrics measurement. In our study we seek answers to the questions: (1) Can we use a feedback-based control system to create a characteristics model of a real production system? (2) Can such a model be sufficiently accurate to detect characteristics changes instead of executing the production application?

The model we have created runs a signalling application, similar to the production application, together with a PID-regulator generating L1 and L2 cache misses to the same extent as the production system. Our measurements indicate that we have managed to mimic a similar environment regarding cache characteristics. Additionally we have applied the model on a software update for a production system and detected characteristics changes using the model. This has later been verified on the complete production system, which in this study is a large scale telecommunication system with a substantial market share.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*performance measures*; B.3.2 [**Memory Structures**]: Design styles—*Cache Memories*; C.4 [**Performance of Systems**]: Measurement techniques

General Terms

Measurement, Performance

Keywords

Control Theory, Feedback computing, Performance Analysis, Characteristics, Cache memories, Simulation, Load Testing and Design Aids

1. INTRODUCTION

Measuring behavioural characteristics for complex large scale computer systems is difficult since this requires either a full production system or advanced test programs with large test systems. After a software update, it is essential to measure behavioural characteristics and check that the nature of the system has not changed. Behavioural changes results in costly and time consuming verification in the development cycle. Late detection of unfulfilled requirements due to characteristics changes leads to increased lead time, since parts of the system must be re-investigated and re-implemented. Such increase in development time may not be accepted since short time-to-market is essential[6]. The system we are investigating is a telecommunication system with a market share of about 38% in 2011 [6]. It consists of 5M SLOC [3] and runs on more than 20 types of boards with different hardware layout and functionality servicing both voice and data communication.

We define one instance of the computer system we are investigating as a *node*. One node can consist of many CPUs but from a system point of view they are grouped as one execution entity. A large scale node has many CPUs, a small scale node may consist of only a single CPU. A node communicates extensively both internally and externally between nodes using signals, i.e., operating system messages. We introduce two concepts central to our investigation. The first; *behavioural characteristics* is in our case CPI, CPU-load or signal turnaround time but can be any metric that describes the behaviour or performance of the system. The second is *load characteristics* which is described by metrics that will affect the behavioural characteristics of the system. In our investigation we have concentrated on cache misses but it can be any other metric such as TLB usage, branch statistics, number of system calls or interrupts etc.

For early detection of behavioural characteristics changes we suggest to create a model of the production system on a small scale node. The benefit of doing so is that we don't have to wait for the availability of large scale nodes which are expensive and difficult to obtain. Additionally, changes in the platform may require modifications in the application software which even more extends the time before characteristics measurements can be made. Our approach is to alter load characteristics, in our case cache miss rate, to change the behavioural characteristics. Our model system consists of a signalling application and a load regulator. The signalling application simulates the production system by communicating extensively between processes. Additionally, it is used to detect performance differences when applying the model, i.e. instead of using IPC/CPI as a metric. The load regulator is implemented as a Proportional-Integral-Derivative (PID) regulator and generates hardware load in the form of cache misses

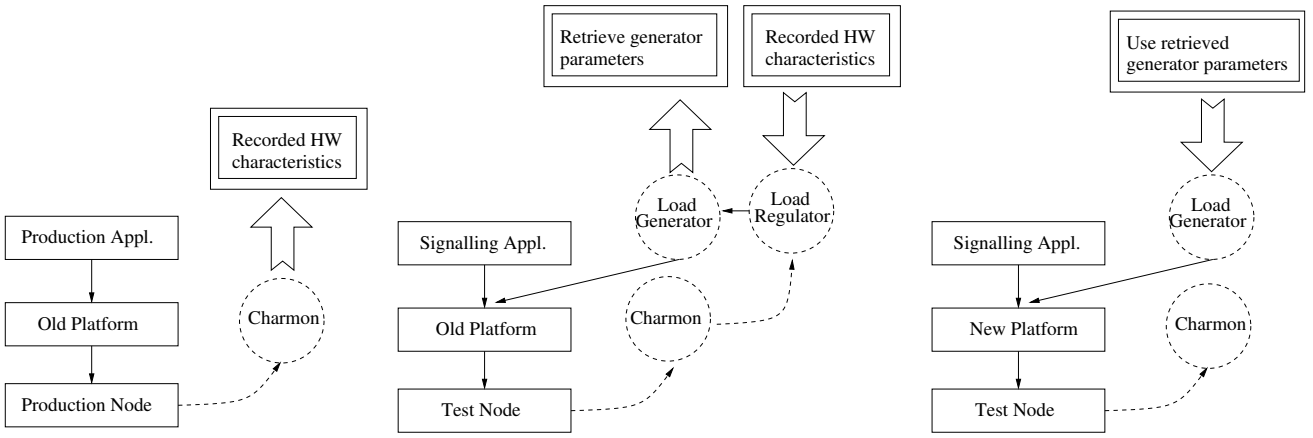

Figure 1: Our procedure in three steps to measure characteristics for a new software release by modelling a production system.

at a rate consistent with a real production system. Changes in the production system can now be tested on the model system with similar behaviour. The model in itself is easily extendable with additional regulators and generators for additional hardware metrics such as L3-cache hit/miss rate, branch prediction statistics etc. The results we have achieved is to PID-regulate L1 Instruction, L1 Data and L2 Data cache misses according to a predefined rate measured on a real production system. Additionally we have succeeded to detect behavioural characteristics changes in the production environment by using the model node to test a new software release.

2. PROCESS

We define the *platform* as the operating system bundled with basic cluster functionality such as program handling, error recovery mechanism, load balancing etc. On top of the platform one or several telecommunication applications run using the platform API. The *system* denotes the complete executing software on the hardware. In our investigation we have followed this procedure to model a large system on a much smaller node.

1. Record characteristics for an existing production system, see Figure 1a.
   (a) Run a complete customer system for which we want to create a hardware (HW) characteristics model.
   (b) Measure load-characteristics, in our case L1 Data, L1 Instruction and L2 Data cache misses.
2. Create an environment that mimics the characteristics, obtained in step 1, on a test node, see Figure 1b.
   (a) Start the platform (same release as in step 1) together with the signalling application.
   (b) Use the HW load regulation algorithm to reach the same load-characteristics ratio as in step 1b.
   (c) Retrieve metrics from the regulation algorithm. In our case the internal counters used to describe the amount of cache-misses generated by the load generation algorithm.

In the scenario above we have sampled behavioural characteristics from a real customer system and then recreated a similar execution environment for a limited platform signalling application. By storing the internal load-generation parameters, in 2c, we can generate the same rate of cache misses without using the regulation algorithm. This allows us to change the platform and then apply the same rate of misses. Investigating the ratio of misses allows us to detect changes in platform behaviour. In the continued procedure below we can measure characteristics for a different release of the platform to get an indication of how it will perform running the complete system.

3. Detecting behavioural characteristics changes, such as signal turn-around time, on small scale hardware, see Figure 1c.
   (a) Start the new platform together with the signalling application.
   (b) Generate HW-load at the same rate as obtained in step 2c.
   (c) Check behavioural characteristics for the benchmarking application, such as signal turnaround time.

3. OUR APPROACH

3.1 The Characteristics Monitor

Implemented within the platform is a continuously running characteristics monitor called charmon. It gathers information about the HW-usage of the complete system by periodically sampling performance monitor counters, PMCs. PMCs can be configured to count different HW-events such as cache misses, TLB misses, branch statistics etc. The charmon stores counted events in a database together with commonly requested key performance indices such as cycles per instruction (CPI), L1-2-3 cache hit/miss rate and ratio, TLB hit/miss rate and ratio, branch statistics, CPU load and others. The probe effect is low since the PMCs are located inside the CPU with low or no performance penalty and the database storage and PMC reprogramming occurs infrequently. Furthermore, using PMCs gives us the opportunity to measure non-instrumented code reducing the intrusiveness of the monitor. For more information regarding the characteristics monitor and load generator see [9].

3.2 The Load Regulator

The load regulator operates in two modes. The first mode is a client to charmon, subscribing to metrics for certain HW-properties. With a user supplied ratio as reference the regulator tries to generate HW-load reaching the reference value. The second mode operates in a stand-alone manner generating cache misses at a specific rate without any feedback of current metrics. To generate a specific HW-load ratio a PID-regulator is used to control each parameter. Each HW-property, such

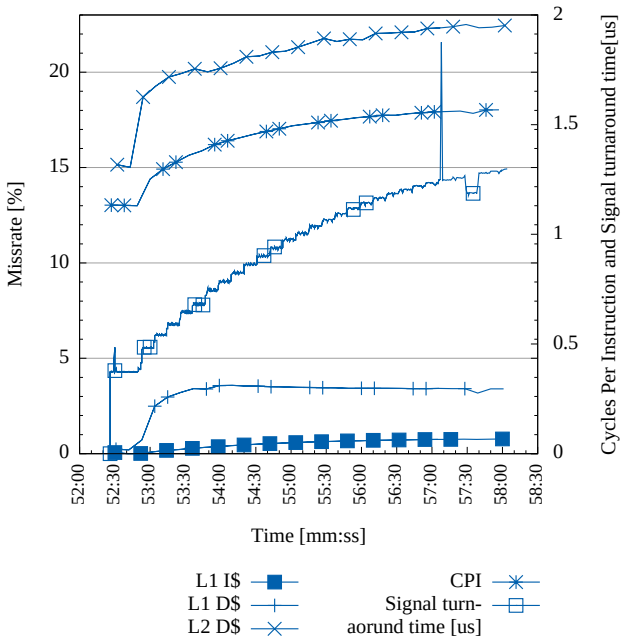

Figure 2: Cache misses and CPI when running a simple signalling application bouncing signals between two processes located on the same core. The load regulation application strives to have a system miss rate of 0.74% L1 I-cache, 3.3% L1 D-cache and 22% L2 D-cache which is similar to a production system. The regulation scenario described in this figure was started after the system was completely stable, in our case after about 52:00 minutes

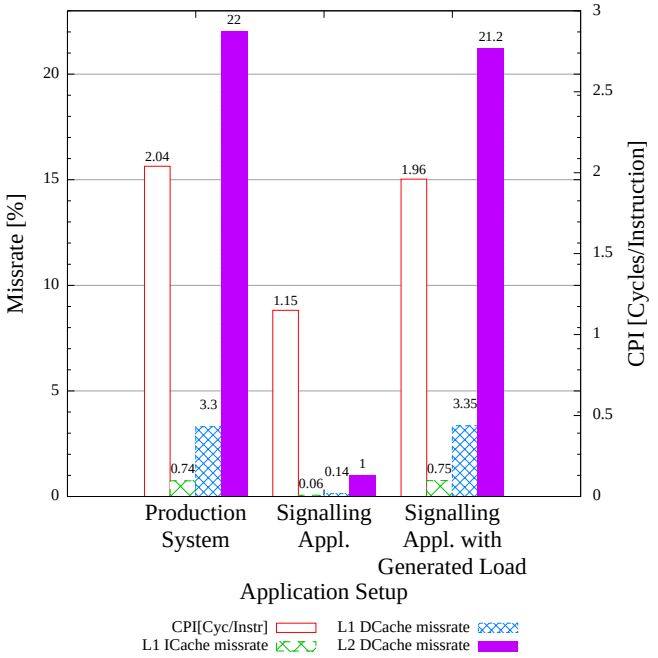

Figure 3: Measured cache usage and CPI for the reference system, test application and a recreated execution environment with the test application. Characteristics for the reference system is similar to the mimicked scenario.

as cache miss-rate, is controlled by its own PID-regulator. See Figure 2 for a typical regulation scenario. The test application has an initial characteristics (to the left) that differs from the final characteristics reached after the regulation has started converging. The effect of increasing number of cache misses can be observed by looking at the CPI and signal turn-around time that increases as the cache usage is increased. The spike (57:00) in the graph is difficult to explain but there are many services running on the system being monitored. It can be any periodically running signalling service synchronising information with other boards generating a burst of load causing a spike. However, the regulation algorithm keeps converging after the spike has occurred. The initial control loop parameters for each of the properties were empirically discovered to provide stability rather than quick convergence. See [9, 12] for a detailed description of how cache misses are generated.

4. RESULTS

4.1 Mimicking a Production Node Environment

The assumption is that introducing cache misses will cause the application to execute in a less efficient way. According to Doucette and Federova [4] an introduction of L2 D-cache load causes a significant slowdown of simultaneously running applications. We have used cache misses to mimic the execution environment of a real customer based system within the constraints of a much smaller test suite. A first test of the procedure outlined in Section 2 reveals that a load regulator can add a hardware load yielding a similar load profile as the real application. As can be seen in Figure 3 the cache usage by the test applications itself does not mimic the real application. When running the load regulator in parallel to the test application the hardware usage becomes almost identical. Additionally CPI has increased from 1.15 to 1.96 which is close to the original real world measurements of 2.04. We can conclude that introducing a load-generator together with an already present application-benchmarking suite improves the result.

4.2 Characteristics Evaluation in Daily Production Test Environment

We have evaluated our approach described in Section 2 on a real scenario where a software update was implemented for the platform. All applications running on the platform were left unchanged. The platform change relates to incorrect behaviour of cache handling in some rare circumstances. Before implementing the suggested solution it was suspected that it could affect the behavioural characteristics of the platform therefore leading to increased load when running the production system. We have been using a signalling application together with a load generator to simulate the much larger and complex telecommunication system. The application creates two processes per core bouncing signals between them at a certain rate. As a behavioural characteristics we measure the signal turnaround time, which increases by 1,43% (average of 312 samples on 7 cores) with the software update running the signalling application alone. When altering the load characteristics to mimic the cache usage for a real production system (with respect to L1 Instruction, L1 Data and L2 Data cache usage) the difference is much larger and easily detectable. With the software update, the signal turnaround time increases by 9,13% (average of 74 samples on 7 cores) compared to running without the software change. As a reference running the same software update on the complete production system the CPU load change is 7.57% (average of 606 samples on 6 cores). We should be careful making a direct comparison between signal turnaround time and CPU load since they do not quantify the same metric. They should however be related since a higher CPU-load gives an increased signal turnaround time due to longer processing time at the

sending and receiving processes. Measuring CPU load for the signalling application is not easily done since the cache-miss generators also cause some CPU load. For the production system signal turnaround time is not possible to measure since the production code lack this mechanism. We have not yet found a fully comparable metric that is easily measurable among both systems. Interpreting the figures from the evaluation shows, in this particular case, a relationship between the CPU-load on the production system and signal turnaround-time on the model system. How to quantify this relationship is something that needs to be further studied.

5. RELATED WORK

Bell and John [2] describes a similar approach to ours. They define a method to model an application by synthesizing vital metrics. The model is then used to automatically create a representative test application with similar characteristics to the original one. They have applied this method on the SPEC2000 benchmark suite and the result shows that IPC differs on average 2.4% between the original applications and the model applications. Other metrics differ a degree slightly higher than ours, I-Cache 8.6% and L2 cache misses not explicitly written but to a large degree. Starting with the synthesizing procedure we use a feedback control loop to model the system while Bell and John [2] use statistical simulation with instruction traces, described by Nussbaum and Smith [10]. Bell and John states that the synthesis procedure is semi-automatic and an average of ten passes with some manual intervention is needed to tune the synthesis parameters. As a comparison feedback control allows the synthesis procedure to converge with no user interaction. Additionally, the model in our case is described by configuration parameters fed to a generic application. For Bell and John this is done at compile time requiring recompile to change its configuration. Another difference in our approaches is that we use a signalling application to detect any performance changes between releases while Bell and John uses IPC.

Doucette and Fedorova [4] have implemented a similar functionality to ours when generating cache misses to determine application sensitiveness for different architectures. For example if an application is sensitive on one particular resource and another architecture has different amount of that resource the application performance is to some extent related to the hardware in the same way as the generator functions. One can in other words to some extent predict the performance of an application without actually running it on the target platform. As in Cache Pirating by Eklöv et al. [5] our application steals hardware-resources from other applications thus starving them. Our approach is to use the a cache miss generator to mimic a certain environment, while the cache pirate is used to reduce the available hardware-cache to determine the application demand for cache and memory bandwidth. We also work on a core-private cache instead of a shared cache. Saavedra and Smith [11] explain how to understand cache memory structure and how to generate misses, associativity etc. In the area of continuous system monitoring we can find interesting relations, such as Anderson et al. [1]. In their approach they implement a low intrusive (1%-3%) sample based mechanism to gather system wide information.

6. CONCLUSIONS AND FUTURE WORK

We have managed to synthesise the behavioural characteristics of a real-life production system without using any code instrumentation. The obtained characteristics have been used to create a model that can reproduce the behaviour on a small test system detecting characteristics changes without running the large and complex production system. The feedback control system used in the synthesis phase works well and converges in a stable manner. A natural way to proceed further with this investigation is to decrease the time to converge without sacrificing stability. Also introducing additional metrics, such as L3 cache and branch statistics, would improve the accuracy of the model.

Acknowledgment

This work is funded by Ericsson AB and by the Swedish Knowledge Foundation (KK stiftelsen) through the ITS-EASY program.

7. REFERENCES

[1] J. Anderson, L. Berc and J. Dean. Continuous profiling: where have all the cycles gone?. In *ACM Transactions on Computer Systems*, pp. 357–390, Vol. 15, No. 4, November 1997

[2] R. Bell and L. K. John Improved automatic testcase synthesis for performance model validation. In *Proceedings of International Conference on Supercomputing (ICS)*, pp. 111–120, 2005.

[3] M. Bergqvist, J. Engblom, M. Patel and L. Lundegard. Some experience from the development of a simulator for a telecom cluster (CPPemu). In *Proceedings of the 10th International Association of Science and Technology for Development*, pp. 13–15, Nov. 2006.

[4] D. Doucette and A. Fedorova. Base vectors: A potential technique for microarchitectural classification of applications. In *Proceedings of the Workshop on the Interaction between Operating Systems and Computer Architecture (WIOSCA), in conjunction with ISCA-34*, 2007

[5] D. Eklöv, N. Nikoleris, D. Black-Schaffer and E. Hagersten. Cache Pirating: Measuring the Curse of the Shared Cache. In *Proceedings of Parallel Processing (ICPP), 2011 40th International Conference*, Sept. 2011.

[6] Ericsson. Ericsson unveils new products, partnerships and increased market share at mwc 2012. `www.ericsson.com/thecompany/press/releases/2012/02/1589097c`, 2012.

[7] S. Eyerman, L. Eeckhout, T. Karkhanis and J. E. Smith. A top-down approach to architecting CPI component performance counters. In *IEEE micro*, pp. 84–93, Vol 27, Jan-Feb, 2007.

[8] A. Joshi, L. Eeckhout, R. H. Bell Jr. and L. K. John Distilling the essence of proprietary workloads into miniature benchmarks. In *ACM Transactions on Architecture and Code Optimization*, pp. 1–33, Vol 5, 2008.

[9] M. Jägemar, S. Eldh, A. Ermedahl, B. Lisper Feedback-Based Generation of Hardware Characteristics - Technical Report. *http://www.mrtc.mdh.se/index.php?choice=publications&id=3078*, 2012

[10] S. Nussbaum and J.E. Smith. Modeling superscalar processors via statistical simulation. In *Proceedings of Parallel Architectures and Compilation Techniques*, pp. 15–24, 2001

[11] R.H. Saavedra and A.J. Smith. Measuring cache and TLB performance and their effect on benchmark runtimes. In *IEEE Transactions on Computers*, pp. 1223–1235, October 1995.

[12] Stackoverflow. Generate Instruction Cache misses. In *Stackoverflow forum*, `stackoverflow.com/questions/9793660/what-are-the-causes-for-instruction-cache-miss`, 2012.