Automatic Multi-Core Cache Characteristics Modelling

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

Björn Lisper
Mälardalen University
bjorn.lisper at mdh.se

ABSTRACT

When updating low-level software for large computer systems it is difficult to verify whether performance requirements are met or not. Common practice is to measure the performance only when the new software is fully developed and has reached system verification. Since this gives long lead-times it becomes costly to remedy performance problems. Our contribution is that we have deployed a new method to synthesise production workload. We have, using this method, created a multi-core cache characteristics model. We have validated our method by deploying it in a production system as a case study. The result shows that the method is sufficiently accurate to detect changes and mimic cache characteristics and performance, and thus giving early characteristics feedback to engineers.We have also applied the model to a real software update detecting changes in performance characteristics similar to the real system.

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

When making program change during the early development phases of large complex computer systems it may be difficult to determine the resulting characteristics effects. In many cases there are inadequate test suites mainly testing functionality and not the final performance. Large-scale performance test suites are often run at the end of the development cycle thus yielding a long lead-time between the implementation of low-level functionality and the final test [13]. In this paper we describe a new method to synthesise production workload and then modelling it on a test system providing earlier characteristics feedback to engineers when making design choices. This complies with the opinion stated by Bar et al[3]. of the importance of getting early service-level feedback when making low-level architectural design choices. We have investigated a telecommunication system [5, 9] where the platform consists of approximately 5M Source Lines Of Code (SLOC) and runs on more than 20 types of boards. They have different functionality and hardware layout servicing both voice and data communication.

Our contribution in this paper is a method to reduce the total system development time by making it possible to perform early characteristics tests. We suggest to synthesise a model of the production system and replaying it on a test node. The benefit of doing so is for example; 1) We do not have to wait for the availability of large-scale production nodes that are expensive and difficult to obtain. 2) Changes in the platform may require modifications in higher-level application software that even more extends the lead-time before characteristics measurements can be made. Our approach is to alter load characteristics of a test system according to the synthesised model to mimic the behavioural characteristics of the production system.

The model system consists of a test application and a load controller. The test application simulates the functionality of the production system by communicating extensively between processes. The load controller is implemented as a Proportional-Integral-Derivative (PID) controller and generates hardware load currently in the form of cache misses at a rate consistent with a real production system. Platform changes can now be tested on the model system with similar behavioural changes as it will have on the production system. The model in itself is easily extendable with additional controllers for other hardware events such as L3-cache misses, branch prediction misses, TLB misses etc.

The main contributions presented here are. 1) a novel use of the PID feedback control algorithm to synthesise a characteristics model from a production system. 2) The synthesis procedure is very lightweight and has negligible probe effect due to the use of infrequently reprogrammed hardware counters. 3) The synthesis procedure is autonomic and does not

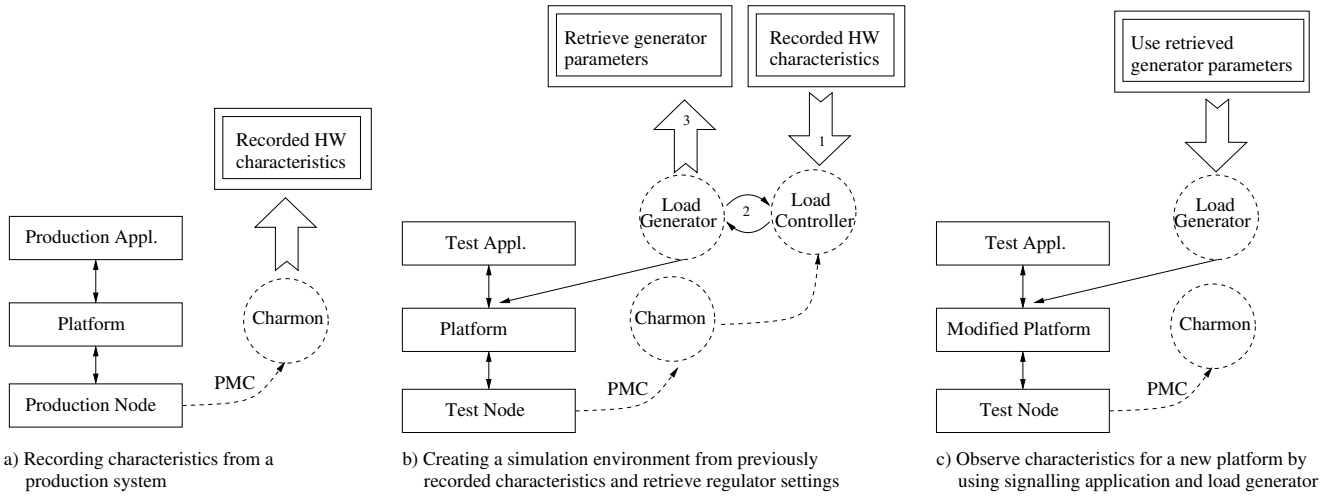

Figure 1: The three-step procedure when a) measuring characteristics for new software releases by monitoring hardware counters (PMU/PMC). In b) we use the characteristics values obtained in a) to have the same hardware utilisation using a function-test application together with a load generator. The convergence of the load generator is handled by a PID-controller. Finally in c) is is possible to detect characteristics deviations in the modified platform by running the same setup as in b).

require any user interaction during the convergence time. 4) The resulting model system mimics the real production environment and have been used in real platform development to find performance related bugs. The results we have achieved is to PID-control 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

The process to obtain characteristics from a production application and mimicking it on a much smaller test environment can be described by three steps. The assumption is that the same type of hardware is used both in the production system and in the test system. Each of them is shortly described in the list below.

1. We start the procedure by sampling characteristics for a production system running in its designated target environment, see Figure 1a.
2. The second step, see Figure 1b, is to create a simulated environment on a test node substituting the production application for a test application paired with a load generator so that they together mimic the characteristics obtained in Step 1.
   (a) Run the test application on the same platform as in Step 1, i.e. exactly the same software release of the platform.
   (b) Use the PID-control algorithm to reach the same hardware load characteristics as in Step 1.
   (c) Retrieve metrics from the control 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 mimicked

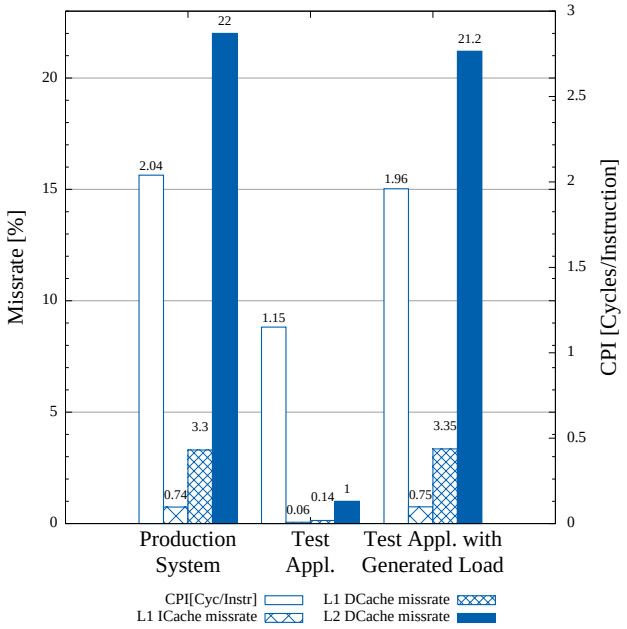

Figure 2: Measured cache usage and CPI for the reference system, test application and a mimicked execution environment with the test application. Characteristics for the reference system is similar to the mimicked scenario.

a similar execution environment for a test application that performs functional test. By storing the internal load-generation parameters, in Step 2c, we can generate the same rate of cache misses without using the control algorithm. This allows us to change the platform and then apply the same rate of cache 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

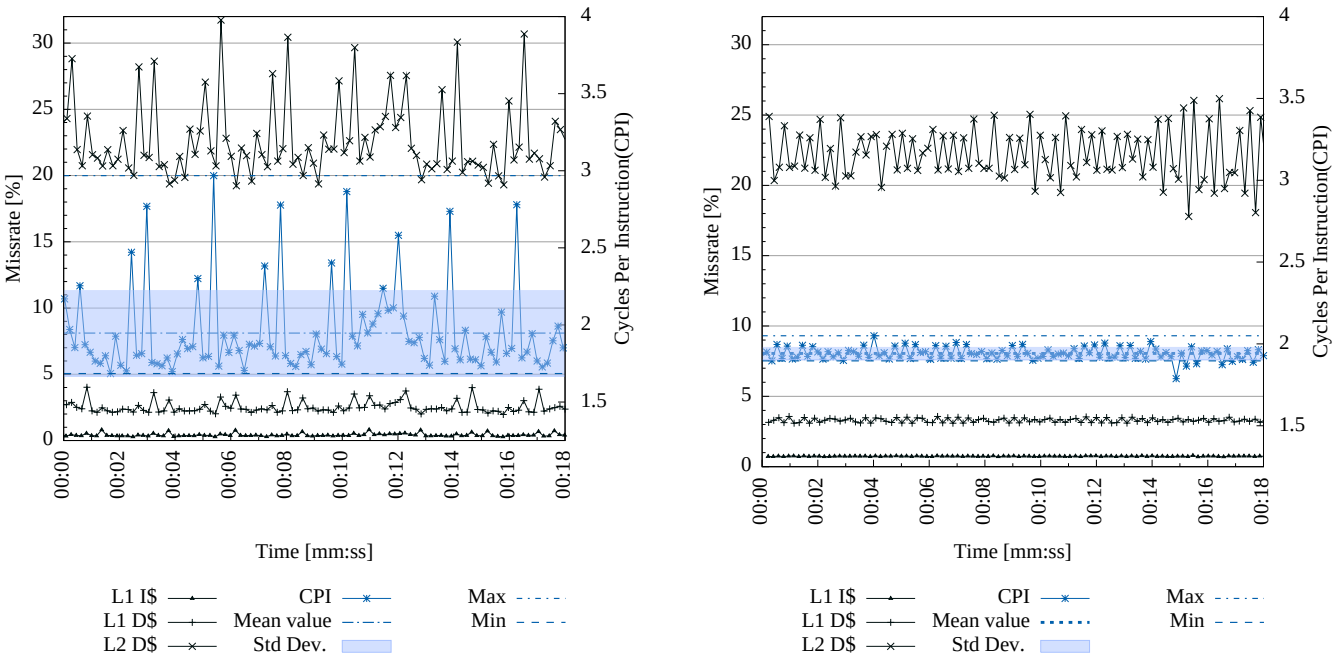

Figure 3: a) Production system characteristics measurements over time. The CPI mean value is 1.95 and the standard deviation is 0.28. b) The model system metrics measured over time using a node substantially smaller than the production system. The CPI mean value is 1.94 and the standard deviation is 0.040 which is narrower than the production system.

perform running the production system.

3. In this last step we can detect behavioural characteristics changes, such as signal turn-around time, for a modified platform without running the production application. see Figure 1c.
   (a) Start the modified platform together with the test 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.

If the signalling turn-around time has changed in Step 3c the characteristics of the modified platform is different from the original one. Low level changes to the operating system can influence the overall performance of the applications drastically if there are problems in cache handling or memory footprint.

3. RESULTS

We have verified the techniques presented in this paper on a telecommunication system by first obtaining the cache characteristics from a production system. Then running the existing test application both with and without a feedback controlled cache load generator making the test environment similar to the production system. As can be seen in Figure 2 there are differences in cache characteristics for the production system and the original test system. When adding the load controller to the test environment the cache characteristics becomes similar to the production environment.

The next test is to see if it is possible to detect performance changes for a specific bugfix. As a behavioural characteristics we measure the signal turnaround time for the test application. For a particular software update the mean signalling roundtrip time increases by 0,75% when running only the test application. Adding the load generator to alter the load characteristics mimicking the cache usage for the production system makes the signalling round-trip time much higher and easily detectable. It increased by 10,8% compared to without the software change. The same update on the production system increases CPU usage by 8,4%.

Finding a comparable metric between the production system and the test environment is difficult. One reason is that the cache-miss generator by itself causes additional CPU load, it is therefore difficult to use this metric. Signal turnaround time is on the other hand not possible to measure for the production system since it is currently not implemented and being a real customer product it is very difficult from a product manager perspective to add new functionality.

We should be careful making a direct comparison between signal turnaround time on the test system, t_s, and CPU load on the production system but we estimate that $t_s = t_p + t_t$ where t_p is the time it takes to process a message and t_t is the time in transit between nodes. In the system we are investigating $t_t \ll t_p$ since the available bandwidth is very high and the communication path is relatively short. Thus t_s is proportional to t_p and we conclude that CPU load is a major contributor to the signal turnaround time. A higher CPU-load results in an increased signal turnaround time due to longer processing time.

As a conclusion, we have shown an improvement to detect characteristics changes in the early stages of development. It is easier to detect a change of 10.8% in signalling turnaround time compared to 0.75%.

4. RELATED WORK

Bell and John [4] describes a similar approach to ours. They define a method to model an application by synthesising vital metrics. The model is then used to automatically create a representative test application with similar characteristics to the original one. Starting with the synthesising procedure we use a feedback control loop to model the system while Bell and John [4] use statistical simulation with instruction traces, described by Nussbaum and Smith [12]. 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.Joshi et al. [10] have formulated a concept called performance cloning that can be used to synthesise characteristics from a proprietary application and create a model that mimics a similar behaviour. In effect Joshi et al. implements a similar methodology as Bell and John in [4] but have refined the memory and branching model to be hardware agnostic.Doucette and Fedorova [6] 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. [7] 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. Alameldeen et al. [1] investigate server platforms and come to an interesting conclusion that it is quite difficult to create simulations of production systems. In their work they mimic the desired characteristics by using a tailored work-load suite. Our approach is similar but since they have shown some difficulties to recreate a similar hardware-load profile we use feedback-based load generator to achieve an approximation of the production application. In the area of continuous system monitoring we can find interesting relations, such as Anderson et al. [2]. In their approach they implement a low intrusive sample based mechanism. The sampling is implemented by means of periodically executing sampling interrupts generated by performance counters. In our work this is done by a periodically executing process gathering performance counters in a ring buffer. One of the standard work when monitoring or measuring systems is the LM-Bench suite by Mcvoy and Staelin [11]. It is useful to measure and calculate cache and memory timings. Our platform is unfortunately not supported by the tool so a change to Linux was necessary to give insight into the characteristics. Regarding measurements with performance monitor counters Eranian [8] claims that they are a vital part in performance measurements and evaluation. Their investigation is done on x86 hardware but the basic concept is the same.

Acknowledgment

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

References

[1] A. R. Alameldeen, M. Martin, C. J. Mauer, K. E. Moore, and M. Xu. Simulating a $2M Commercial Server on a $2K PC. *IEEE Computer*, 36(2):50–57, 2003.

[2] J. Anderson, L. Berc, and J. Dean. Continuous profiling: where have all the cycles gone? *ACM SIGOPS*, 15(4):357–390, 1997.

[3] P. Bar, R. Benfredj, and J. Marks. Towards a monitoring feedback loop for cloud applications. In *Proceedings of the International Conference on Performance Engineering*, pages 43–44, 2013.

[4] R. H. Bell and L. K. John. Improved automatic testcase synthesis for performance model validation. In *Proceedings of International Conference on Supercomputing*, pages 111–120, 2005.

[5] 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 International Association of Science and Technology for Development*, pages 13–21, 2006.

[6] 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*, 2007.

[7] D. Eklov, N. Nikoleris, D. Black-Schaffer, and E. Hagersten. Cache Pirating: Measuring the Curse of the Shared Cache. In *Proceedings of International Conference on Parallel Processing*, pages 165–175, 2011.

[8] S. Eranian. What can performance counters do for memory subsystem analysis? In *Proceedings of the ACM SIGPLAN workshop on Memory Systems Performance and Correctness*, pages 26–30, 2008.

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

[10] A. Joshi, L. Eeckhout, R. H. Bell, and L. K. John. Distilling the essence of proprietary workloads into miniature benchmarks. *ACM Transactions on Architecture and Code Optimization*, 5(2):1–33, Aug. 2008.

[11] L. Mcvoy and C. Staelin. lmbench : Portable Tools for Performance Analysis lmbench : Portable tools for performance analysis. In *Proceedings of the USENIX Annual Technical Conference*, pages 279–294, 1996.

[12] S. Nussbaum and J. Smith. Modeling superscalar processors via statistical simulation. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, pages 15–24, 2001.

[13] Telecomasia. Faster time to market with next-gen OSS, http://www.telecomasia.net/content/faster-time-market-next-gen-oss.