BlackjackBench: Portable Hardware Characterization

Anthony Danalis[*] , Piotr Luszczek and
Jack Dongarra[†]
University of Tennessee
Knoxville, TN, USA
{adanalis, luszczek,
dongarra}@eecs.utk.edu

Gabriel Marin and Jeffrey S. Vetter[‡]
Oak Ridge National Lab.
Oak Ridge, TN, USA
{maring, vetter}@ornl.gov

ABSTRACT

DARPA's *AACE* project aimed to develop Architecture Aware Compiler Environments that automatically characterize the hardware and optimize the application codes accordingly. We present the *BlackjackBench* suite, a collection of portable micro-benchmarks that automate system characterization, plus statistical analysis techniques for interpreting the results. BlackjackBench discovers the effective sizes and speeds of the hardware environment rather than the often unattainable peak values. We aim at hardware features that can be observed by running executables generated by existing compilers from standard C codes. We characterize the memory hierarchy, including cache sharing and NUMA characteristics of the system, properties of the processing cores affecting execution speed, and the length of the OS scheduler time slot. We show how these features of modern multicores can be discovered programmatically. We also show how the features could interfere with each other resulting in incorrect interpretation of the results, and how established classification and statistical analysis techniques reduce experimental noise and aid automatic interpretation of results.

Categories and Subject Descriptors

B.8.2 [**Performance Analysis and Design Aids**]

General Terms

Benchmarking, Multicore, Performance

Keywords

Micro-benchmarks, Hardware characterization, Statistical analysis

1. INTRODUCTION

Compilers, autotuners, numerical libraries, and other performance sensitive software need information about the underlying hardware. If portable performance is a goal, automatic detection of hardware characteristics is necessary given the dramatic changes undergone by computer hardware. Several system benchmarks exist in the literature [1, 2, 4, 5, 6, 7, 8, 9]. However, as hardware becomes more complex, new features need to be characterized and assumptions about hardware behavior revised, or completely redesigned.

[*]Also with Oak Ridge National Laboratory (ORNL).
[†]Also with ORNL and the University of Manchester
[‡]Also with Georgia Institute of Technology

In this paper, we present *BlackjackBench*, a system characterization benchmark suite. The contribution of this work is twofold:
1. Portable micro-benchmarks that can probe the hardware and record its behavior while control variables are varied.
2. A statistical analysis methodology, implemented as a collection of scripts for result parsing, that examines the output of the micro-benchmarks and produces the desired system characterization information, e.g. effective speeds and sizes.

Often, important performance related decisions take into account *effective* values of hardware features, rather than their peak values. In this context, we consider an effective value to be the value of a hardware feature that would be experienced by a user level application written in C (or any other portable, high level, standards-compliant language) running on that hardware. This is in contrast with values that can be found in vendor documents, or through assembler benchmarks, or specialized instructions and system-calls.

BlackjackBench goes beyond the state of the art in system benchmarking by characterizing features of modern multicore systems, taking into account contemporary – complex – hardware characteristics such as modern sophisticated cache prefetchers, and the interaction between the cache and TLB hierarchies, etc. Furthermore, BlackjackBench combines established classification and statistical analysis techniques with heuristics tailored to specific benchmarks, to reduce experimental noise and aid automatic interpretation of the results. As a consequence, BlackjackBench does not merely output large sets of data that require human intervention and comprehension; it shows information about the hardware characteristics of the tested platform. Moreover, BlackjackBench does not rely on assembler code, specialized kernel modules and libraries, nor non-portable system calls. Therefore, it is a portable system characterization tool.

2. STATISTICAL ANALYSIS

The output of the micro-benchmarks is typically a curve consisting of a large number of points representing the performance of the code for different values of the control variable.We have developed analyses that can process this data and output values that correspond to actual hardware characteristics.

2.1 Monotonicity enforcement

Except for the micro-benchmark that detects asymmetries in the memory hierarchy, the output curve is expected to be monotonically increasing. If any data points violate this expectation, it is due to noise, or esoteric hardware details that are beyond the scope of our benchmarks. Therefore, as a first post-processing step we enforce monotonic increase using the formula: $\forall i : X_i = \min_{j \geq i} X_j$.

2.2 Gradient Analysis

Most of our benchmarks result in data that resemble step functions and we aim to detect the location of these steps.

First Step. To extract the number of Execution Contexts from the data, we are interested in that first jump from the straight line. Due to noise, there can be small jumps in the part of the data that is expected to be flat. The challenge is to systematically define what constitutes a small jump versus a large jump. We first calculate the relative value increase in every step $dY_n^r = \frac{Y_{n+1} - Y_n}{Y_n}$ and then compute the average relative increase $\langle dY^r \rangle$. We argue that the data point that corresponds to the jump we want (and thus the number of execution contexts) is the first data point i for which $dY_i^r > \langle dY^r \rangle$. The rationale is that the average of a large number of very small values (noise) and a few much larger values (actual steps) will be a value higher than the noise, but smaller than the steps. Thus $\langle dY^r \rangle$ gives us a good threshold between small and large values.

Biggest Step. The Live Ranges benchmark produces curves that start flat (when all variables fit in registers) then potentially grow slightly (if some registers are unfavorable), then exhibit a large step when the first spill to memory occurs, and then continue growing iregularly. Due to the increase in latency before the first spill, the previous approach for detecting the first step is not appropriate for this type of data. However, the steps caused by the additional spills will be no larger than the step caused by the first spill. Furthermore, since the additional steps have higher starting values than the first step, the relative increase $\frac{Y_{n+1} - Y_n}{Y_n}$ for every n higher than the first spill will be lower than the relative increase of the first spill.

The biggest relative step technique can also be used for processing the results of the cache line size benchmark and the cache associativity benchmark. For the TLB page size, where the desired information is in the last large step, the analysis seeks the biggest scaled step $dY^s = dY \times Y$ (instead of the biggest relative step).

Quality Threshold Clustering Unlike the previous cases, where the analysis aimed to extract a single value from each data set, the benchmark for detecting the cache size, count, and latency has multiple steps that carry useful information. Due to the regular nature of the steps this benchmark generates, we can group the data points into clusters based on their Y value (access latency) with one cluster for one cache level. We use a modified version of the quality threshold clustering algorithm [3]. The modification pertains to the cluster diameter threshold. Unlike regular QT-Clustering with the diameter being a predetermined constant, our version uses 25% of the average value of each cluster as the diameter.

Using QT-Clustering, we can obtain the points that correspond to each cache level. We can extract for each cache level the size information from the maximum X value of each cluster, the latency information from the minimum Y value and the number of cache levels from the number of clusters. An example use of this analysis for a Power7 processor can be seen in Figure 1. QT-Clustering is also used for the levels of TLB.

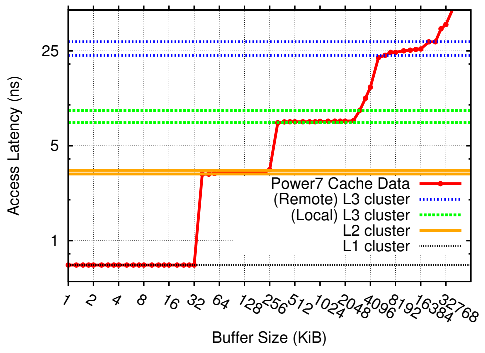

Figure 1: QT-Clustering applied to Power7 Cache Data

3. CONCLUSION

We have presented the *BlackjackBench* system characterization suite, that goes beyond the state of the art in benchmarking by:
1. Offering micro-benchmarks that can exercise a wider set of hardware features than most existing benchmark suites do.
2. Emphasizing portability by avoiding low level primitives, specialized software tools and libraries, or non-portable OS calls.
3. Providing statistical analyses for inferring useful values that describe the hardware from the raw results of the micro-benchmarks.
4. Emphasizing the detection of hardware features through variations in performance. BlackjackBench detects the *effective* values of hardware characteristics, which is what a user level application experiences when running on the hardware.

We described how the micro-benchmarks operate, and their fundamental assumptions. We explained the analysis techniques and demonstrated that our assumptions are valid and portable on a variety of hardware platforms and operating systems. The full paper can be found in [10].

4. REFERENCES

[1] Duchateau, A. X., *et. al* P-ray: A suite of micro-benchmarks for multi-core architectures. In *Proc. 21st Intl. Workshop on Languages and Compilers for Parallel Computing (LCPC'08)* (Edmonton, Canada, 2008), vol. 5335 of Lecture Notes in Computer Science, pp. 187–201.

[2] Gonzalez-Dominguez, *et. al* Servet: A Benchmark Suite for Autotuning on Multicore Clusters. In *IEEE International Symposium on Parallel & Distributed Processing (IPDPS)* (Atlanta, GA, 2010).

[3] Heyer, L. J., Kruglyak, S., and Yooseph, S. Exploring expression data: Identification and analysis of coexpressed genes. *Genome Research 9*, 11 (1999), 1106–1115.

[4] McVoy, L., and Staelin, C. `lmbench`: portable tools for performance analysis. In *ATEC'96: Proceedings of the Annual Technical Conference on USENIX 1996 Annual Technical Conference* (Berkeley, CA, USA, USENIX Association., 1996), pp. 23–23.

[5] Mucci, P., and London, K. The CacheBench Report. Tech. rep., Computer Science Department, University of Tennessee Knoxville, 1998.

[6] Saavedra, R., and Smith, A. Measuring Cache and TLB Performance and Their Effect on Benchmark Runtimes. *IEEE Trans. Computers 44*, 10 (1995), 1223–1235.

[7] Staelin, C., and McVoy, L. mhz: Anatomy of a micro-benchmark. In *USENIX 1998 Annual Technical Conference* (New Orleans, January 15-18 1998), pp. 155–166.

[8] Yotov, K., Jackson, S., Steele, T., Pingali, K., and Stodghill, P. Automatic measurement of instruction cache capacity. In *Proceedings of the 18th Workshop on Languages and Compilers for Parallel Computing (LCPC)* (2005).

[9] Yotov, K., Pingali, K., and Stodghill, P. Automatic measurement of memory hierarchy parameters. *SIGMETRICS Performance Evaluation Review 33(1)* (2005), 181–192.

[10] Danalis, A., Luszczek, P., Dongarra, J., Marin, G. and Vetter, J.S. BlackjackBench: Portable Hardware Characterization. *SIGMETRICS Performance Evaluation Review 40(2)* (2012)