Techniques for Benchmarking of CPU Micro-Architecture for Performance Evaluation

Varad Deshmukh*[1], Nishchay Mhatre*[1], and Shrirang K. Karandikar[2]

[1]College of Engineering, Pune
[2]Computational Research Laboratories, Tata Sons Ltd.

***Abstract*—CPU Micro-architecture has a significant impact on performance and hence is of special importance to the High-Performance Computing industry. In this paper, we describe the development of a suite of benchmark programs which enables us to evaluate and compare processors based on the performance of the micro-architecture itself, independent of workload characteristics. The benchmark suite is comprehensive in its coverage of important hardware features that impact performance. The benchmark programs run directly on the CPU hardware as opposed to processor simulators and use innovative methods that we have developed to stress and benchmark programmer-invisible micro-architectural features. We run the benchmarks on machines representing a variety of micro-architectures and develop a mathematical approach towards analysis of the results. We then verify our approach using data published by CPU manufacturers and the SPEC suite.**

I. Motivation and Previous Work

Evaluation and comparison of micro-architectures is of great importance and utility to the HPC industry, since even minute micro-architecture improvements can make a difference in HPC cluster performance.

The traditional approach, using high-level application suites, fails to relate performance directly to the micro-architectural characteristics, unless it is used in conjunction with the profile of the application. However, hardware should be evaluated on the basis of the performance of its constituent components, independent of application characteristics. Such evaluation requires the use of unconventional benchmark programs, because micro-architecture is inherently invisible to the programmer. Such programs are written for optimisation or tuning of HPC applications and compilers rather than for measurement of hardware performance, for example [1], [2].

Studies that focus on performance evaluation, such as [3], [4], [5], [6], [7] and [8] mostly use CPU simulations. Just like available benchmark packages, such as STREAM [9], Ramspeed [10], 7-bench [11], they lack comprehensiveness in terms of hardware features measured. Some works, like [3], [12] introduce useful mathematical tools and and methods of analysis, but without benchmarks.

Our work brings together all the aspects. We have integrated benchmarks for all the important micro-architecture features in one package, developing new benchmarks wherever warranted. These benchmarks are executed directly on hardware and coupled with an analysis technique, thus characterising the processor based on performance of its micro-architectural features, independent of the application profile.

II. Micro-architecture benchmarking approach:

To make the benchmark comprehensive, we consider the set of micro-architectural features which have the greatest influence on performance. The criteria for choosing these match the quantitative considerations used in CPU micro-

* Student authors. This work was done during their internship with Tata CRL.

architecture design, as explained in [13]. We map each feature to a set of execution 'events' associated with it and associate every event with a quantitative measure of performance. We recognise five features and seventeen quantities in all.

We develop benchmarks containing precise execution sequences that exploit the fundamental limitations of the micro-architectural features and deterministically give rise to the events of interest. Accurate performance measurement counters are used to time these code sequences. The values measured by these counters are proportional to the performance of the corresponding hardware feature.

Using this approach, we developed new techniques for benchmarking the data cache, instruction cache and branch predictors. These are presented in table I. Existing techniques similar to those in [5] and [4] were used for benchmarking the TLB and functional units respectively and are not described here. Also, for brevity, only one of the new techniques - branch predictor benchmarking - is described in detail to exemplify our approach.

A. Branch Predictor Benchmark: Tree-of-Code Technique

In this method, we generate code in the form of basic blocks arranged as a binary tree. This layout is shown in Figure 1. Each block contains a set of arithmetic instructions, followed by a conditional branch. Execution of code in block n of the tree, falls through to the left child block - $2n$ (located sequentially in program memory) for an 'untaken' branch and jumps to the right block - $2n + 1$ in case of a taken branch.

We control the total number of branches and saturate the hardware of the branch predictors by using a large tree with every branch at a distinct address in the code.

We initialise a variable with a particular sequence of bits. Instructions in each block check the value of the bit corresponding to the block number. The jump is taken if the bit is one. Execution falls through to the next block if the bit is a zero. By controlling the sequence of the bits in the pattern, we predetermine the flow of execution. By controlling the randomness of the bits, we control the degree of predictability.

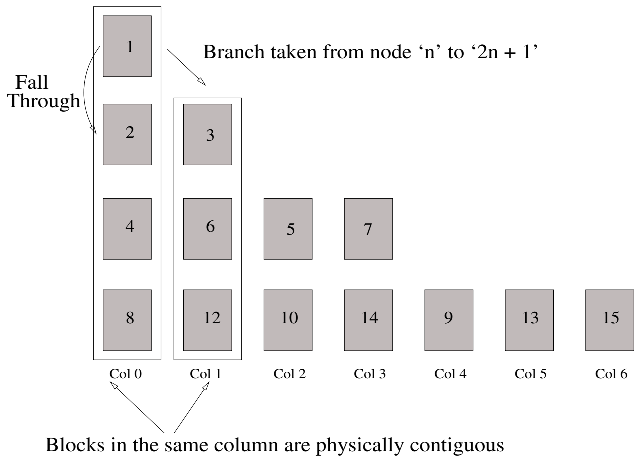
Branch taken from node 'n' to '2n + 1'
Fall Through
1
2
3
4
6
5
7
8
12
10
14
9
13
15
Col 0
Col 1
Col 2
Col 3
Col 4
Col 5
Col 6
Blocks in the same column are physically contiguous

Fig. 1. Branch Predictor benchmark: Layout of code blocks

A pattern of all 1's or all 0's is a highly predictable pattern (best case), and causes very few mispredictions. On the other hand, a random sequence of 1's and 0's, with an equal number of both, is an unpredictable pattern (worst case) and causes more mispredictions.

By timing the execution for both high and low predictability cases and taking their difference, we get a quantity proportional to the penalty of misprediction and to the predictor accuracy.

Using this technique we have obtained the performance of branch predictors. We checked the accuracy of prediction, using on-chip performance counters for both high and low predictability cases. We found that the misprediction rate was close to 0.05% in the former and up to 30% in the latter case. This validates our method of varying predictability. Since a 30% miss rate is unlikely for most real programs, it is sufficiently bad as a 'worst case' condition.

III. Machine comparison using performance vectors

We measure the values of the seventeen quantities, using our benchmarks and measurement techniques for a variety of micro-architectures.

TABLE I
MICRO-ARCHITECTURE BENCHMARKS

Hardware Feature	Event Caused	Quantity Measured	Benchmark Program	Technique
Data cache	Cache miss (capacity)	Miss penalty	FIRA (Forward Initialisation - Reverse Access)	An array larger than last level cache is accessed from first index to last. Accessing different parts of the array in reverse order, we cause a known number of cache misses in each level.
Instruction cache	Cache miss (conflict)	Miss penalty	Cyclic-Jump	The code contains a long, cyclic series of unconditional jumps to addresses in the same set. By saturating the set, we cause a known number of 'conflict' misses.
Branch Predictor	Misprediction of Branch Outcome	Accuracy of prediction Misprediction penalty	Tree of Code	Described in detail in section II-A

We define a 'vector' unique to each machine, characterising its micro-architecture level performance. Each of the seventeen characteristics is a dimension of this vector and its measured value is the component of the vector along that dimension, as shown in the radar-chart in Figure 2.

A one-to-one comparison can be made between micro-architectures using the vectors. A simple comparison of the components along the axes, tells us which micro-architecture is better with respect to which particular characteristic – an impossible task with high-level benchmarks. For example, the Nehalem-EP and Clarkdale micro-architectures are compared in Figure 2.

Figure 3 shows the Kiviat graphs for all the test machines measured. On a visual inspection they can be classified into groups with similar looking shapes. It is interesting to note that these groups exactly correspond to the micro-architecture design families from Intel and AMD. Similar looking graphs are observed to be from the same family while distinctly different graphs represent other design philosophies.

IV. PERSHAPE ANALYSIS OF MACHINE PERFORMANCE VECTORS

We use a quantitative measure of the similarities and differences revealed by the benchmark suite, called the 'pershape distance'. First defined in [3], we adapt this to the micro-

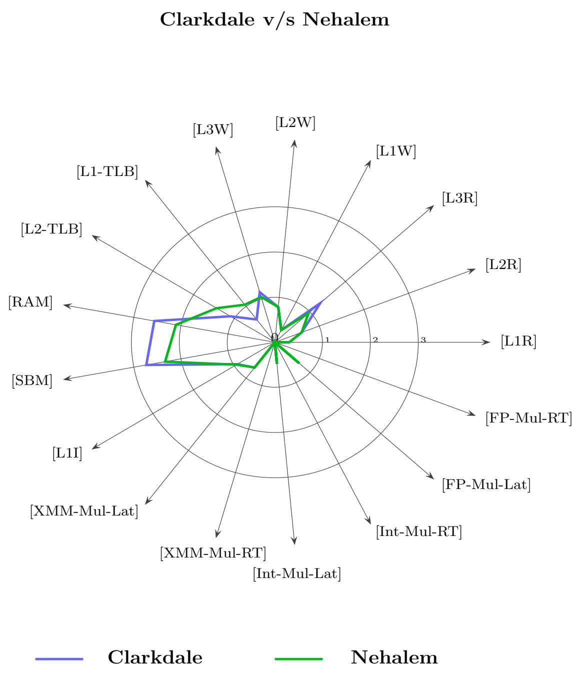

Fig. 2. Comparison of Nehalem and Clarkdale Micro-Architectures. Smaller values indicate better performance. Scale is logarithmic to base 10.

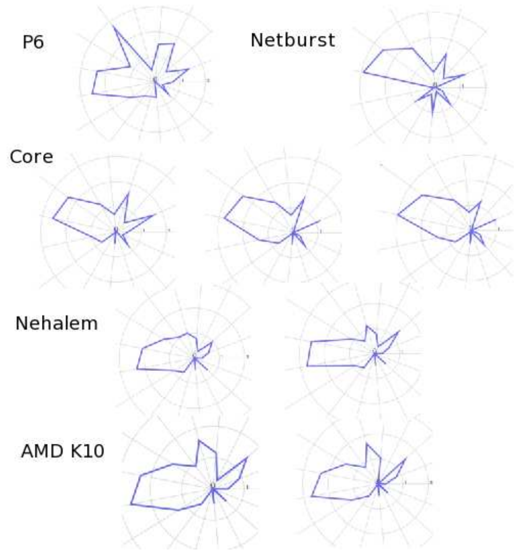

Fig. 3. Micro-Architectures By Family: Top left: Yonah (P6 family), top right: Prescott (Netburst). Middle row: Conroe, Clovertown and Merom (Core). Bottom row: Shanghai and Thuban (AMD K10).

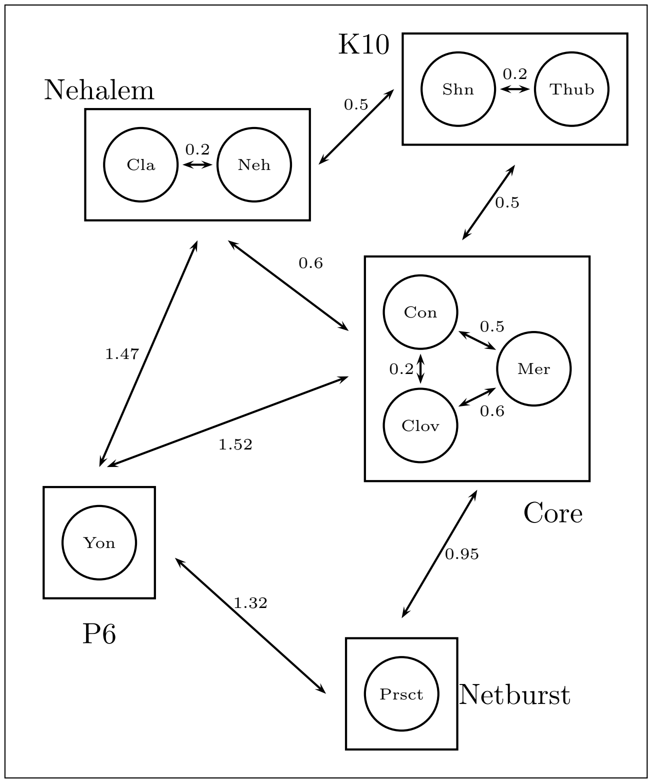

Fig. 4. Comparison of Nehalem and Clarkdale Micro-Architectures

architecture benchmark results. This is a metric which represents the variability of performance between two machines, over the whole spectrum of workload profiles. Even if the clock frequencies of the two machines change, the differences between the overall distribution of their performance parameters does not change. Pershape distance also has this property.

If the pershape distance between two machines is small, their performance across the different types of workloads is more or less similar, else the machines differ significantly. There is no bias introduced by differences in operating frequency.

We calculate the pershape distances between our test processors and draw a graph, where the micro-architectures are the nodes and the pershape distances are edge weights, as shown in Figure 4. We observe that the groups formed in this graph are exactly the same as the family relations that we observed in the previous section. Edge weights between members of the same family are small as compared to those between different families.

V. Comparison with Published Information

We can draw certain inferences about the relative performance of particular features of two micro-architectures after a comparison along the corresponding dimensions of their performance vectors.

Our hypothesis is that if a majority of published performance results for two machines A and B show that A performs better in a program X, and the value of the performance vector dimension related to X is better for A, then we can say that micro-benchmark results correlate with those of the high level benchmarks and the hypothesis is verified.

We use published data from Intel and AMD and the SPEC CPU INT 2006 [14] results for the different micro-architectures with a statistical comparison program, to make such a comparison. We find a correlation between 87 and 99%.

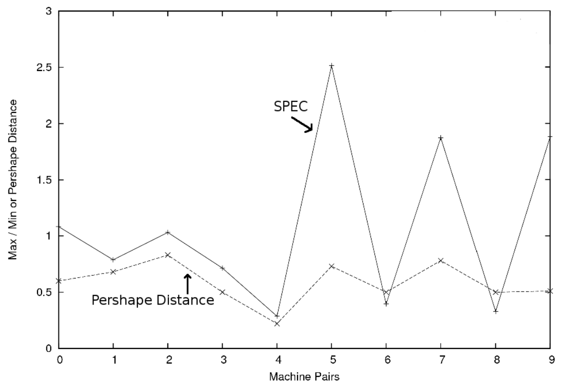

Fig. 5. Correlation of Pershape Distances and SPEC results

The SPEC results are highly workload-dependent. To approximate the overall distribution of performance, we take the maximum value and minimum values among the quotients of the SPEC results of any two processors and calculate the ratio of the two. This 'Max/Min' ratio is the closest approximation we can make to the pershape distance with a high level benchmark. We obtain the ratios for the machine pairs and compare them with the pershape distances.

A comparison of the maximum to minimum ratios with the pershape distances is shown in Figure 5. The X axis represents different pairs of machines and the Y axis shows both the pershape distances as well as the scaled down Max/Min ratios. The upper line depicts the variation of Max/Min ratio across machines, while the lower line represents the pershape distances between the machines.

The trend followed by both is the same. This shows that the same information about overall performance as is provided by the high level benchmarks, can be provided by the micro-architecture benchmarking suite.

VI. Conclusion

We have developed a comprehensive, representative set of benchmarks, which uses some new methods of micro-architecture benchmarking and compares machines independent of workload profile. We have demonstrated that the results provided by this suite are on the expected lines of theory and previously available results. Thus, this suite represents a new, more direct approach toward measuring CPU performance and comparing micro-architectures and a potential improvement over traditional methods.

References

[1] J. Demmel. Single processor machines: Memory hierarchies and processor features; case study: Tuning matrix multiply. [Online]. Available: http://www.cs.berkeley.edu/~demmel/cs267_Spr09/

[2] R. H. Arpaci-Dusseau *et al.*, "Empirical evaluation of the cray-t3d: A compiler perspective." in *ISCA'95*, 1995, pp. 320–331.

[3] R. H. Saavedra-Barrera, "Cpu performance evaluation and execution time prediction using narrow spectrum benchmarking," Ph.D. dissertation, EECS Department, University of California, Berkeley, Feb 1992.

[4] T. Granlund, "Instruction latencies and throughput for amd and intel x86 processors," CSC, KTH, Tech. Rep., 2009.

[5] Y. Yuanhua, "Measurement of data cache and tlb parameters under linux," Ph.D. dissertation, Department of Computer Science, The University of Auckland, 2000.

[6] M. Sakamoto *et al.*, "Microarchitecture and performance analysis of a sparc-v9 microprocessor for enterprise server systems," in *International Symposium on High-Performance Computer Architecture*, 2003, pp. 141–152.

[7] K. Skadron. (1999) Characterizing and removing branch mispredictions.

[8] S. Eyerman *et al.*, "Characterizing the branch misprediction penalty," in *International Symposium on Performance Analysis of Systems and Software*, 2006, pp. 48–58.

[9] [Online]. Available: http://www.cs.virginia.edu/stream/

[10] [Online]. Available: http://alasir.com/software/ramspeed/

[11] [Online]. Available: http://www.7-cpu.com/

[12] U. Krishnaswamy and I. D. Scherson, "Micro-architecture evaluation using performance vectors," in *SIGMETRICS*, 1996, pp. 148–159.

[13] J. Hennessy and D. A. Patterson, *Computer Architecture, A Quantitative Approach, Fourth Edition.* Morgan Kaufmann Publishers, San Francisco, CA, 2007.

[14] [Online]. Available: http://www.spec.org/