Computer Architecture News, 10, 6,
December, 1982.

On the Use of Benchmarks for Measuring System Performance

Henry M. Levy and Douglas W. Clark
Digital Equipment Corporation
Littleton, Mass. 01460

Two recent CAN articles [1,2] have presented the results of experiments to measure the performance of various computer systems, including the Motorola 68000, the DEC VAX-11/780 and the Intel 432, 8086, and 80286. Comparisons of the systems were based on the execution time of four small benchmarks coded in the C and Pascal languages. The findings were summarized by a graph comparing the average performance of the benchmarks on each system relative to their average performance using VAX/VMS Pascal on the VAX-11/780.

Although Hansen et al[1] state that "...a high-level language system consists of the compiler and the machine, so we are not measuring just the architecture and hardware implementation", this observation is lost as the paper moves from results to conclusions. Comparison of architectures through the use of benchmarks requires careful attribution of performance to the contributions of many factors, including architecture, compiler quality, technology, memory speed, language semantics, and benchmark implementation. The benchmarks used for these measurements raise a number of significant questions, some of which are addressed below.

1. The benchmarks. What do these benchmarks measure? Overall, the benchmarks used in the Berkeley measurements are extremely simple and test only a small set of 16-bit integer arithmetic, compare, and branch instructions. In programs this small, the addition of one instruction within a loop can make a difference of 30% in performance. There are no programs to exercise large integer arithmetic, floating point, memory addressing of large arrays, input/output capabilities, etc. Features of more sophisticated architectures, such as the VAX and 432, are not exercised at all.

2. The implementations. What is the effect of the benchmark implementation on the results? Benchmarks were written in both Pascal and C languages. The Pascal and C implementations are sufficiently different in some cases to make the comparison not meaningful. For example, the Pascal string search program includes a procedure call to a search routine within an inner loop, while the C implementation does not. Seemingly small implementation differences can have dramatic effects. Steve Hobbs has demonstrated that the addition of local pointer variables to one of the C string search routines can reduce the VAX execution time by 50%.

3. The language semantics. How do the semantics of the languages used bias the results of the measurement? For example, the C string search uses pointers to access characters while the Pascal program uses arrays and index variables. On any compilers that implement a substring function, such as PL/I or Bliss, the 70-line Pascal and 35-line C programs could be replaced by a program only several lines long. On VAX, such a program would generate the MATCHC instruction to perform the search (our test benchmark in Bliss using the substring function ran almost 5 times faster than the VMS C benchmark).

4. The compilers. What is the effect of compiler quality? The two VAX compilers used by the authors of [1,2] (VMS Pascal Version 1 and Unix C) were both ported to VAX from other architectures. Neither was originally intended for VAX. If we rerun the benchmarks using the VAX/VMS C, Pascal Version 2, and Bliss compilers we get significantly different results. These compilers were developed specifically for the VAX; the C compiler uses a code generator originally written for the VAX PL/I compiler [3]. Table 1 shows the performance of the Berkeley benchmarks on the VAX-11/780 when run with the various compilers. The difference in performance between VMS C and Unix C, for example, is as high as a factor of 2.3 on these benchmarks, while VMS Pascal V2 programs run about 1.6 times faster than Pascal V1 programs.

Table 1. VAX-11/780 Execution Times

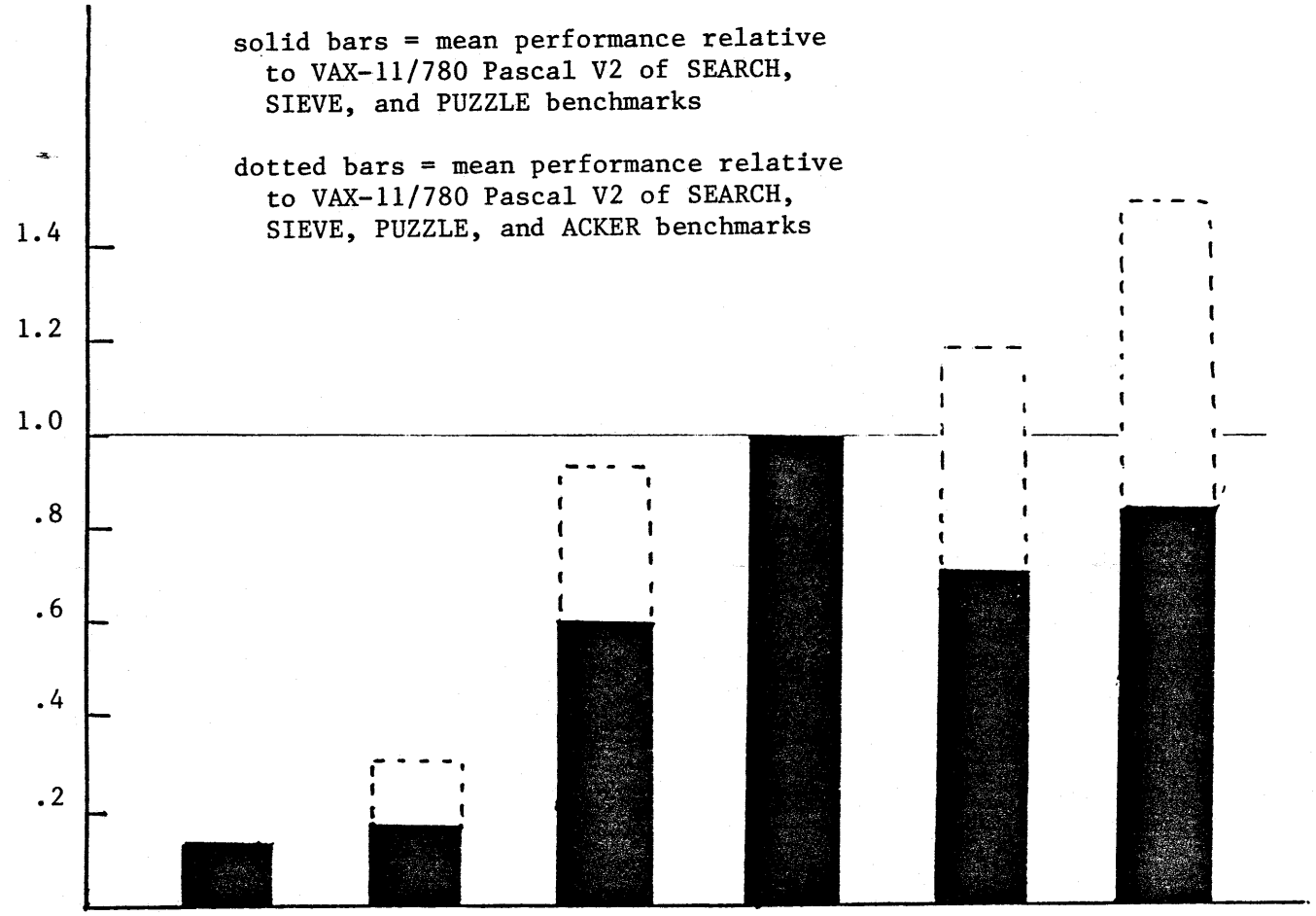
Language	Word Size	Benchmark Time (milliseconds) Search	Sieve	Puzzle	Acker
Bliss(see note)	32	.12			3128
Bliss (VMS)	32	.49	184	5818	4222
C (VMS)	32	.61	116	4625	5048
C (Unix)	32	1.4	250	9400	4600
Pascal (Unix)	32	1.6	220	11900	7800
Pascal V1 (VMS)	32	1.4	259	11530	9850
Pascal V2 (VMS)	32	.88	144	6130	6135

note: Fast Bliss version of Search uses Bliss substring function (MATCHC instruction). Fast Bliss version of Acker uses fast routine linkage (JSB instruction). Unix C and Pascal, and VMS Pascal V1 numbers were taken from [1].

5. The measurement methodology. What is the effect of measurement methodology in the reported results? In the measurements cited in [1,2], some programs were run on an operating system, some on a hardware simulator, and some were measured through use of a logic analyzer to count the number of cycles. Programs run on an operating system may incur the costs of program loading, including accessing the program file, initializing the memory environment, etc.

6. The use of averages. How are the results dependent on the statistics chosen? In the measurements cited, the statistical mean of the relative performance of the four benchmarks is used to compare the systems. One unusually small or large value can cause the mean to be unrepresentative, particularly with a small sample size. This problem is illustrated in Figure 1, which shows mean performance of the benchmarks relative to the performance on VAX-11/780 Pascal Version 2. The solid bars show mean relative performance of Search, Sieve, and Puzzle benchmarks. The dotted lines show the mean relative performance when the Acker benchmark, which measures only procedure call cost, is averaged with the other three. In several of the machines measured in [1,2], the relative performance of the Acker program is two or three times greater than the relative performance of any of the other benchmarks.

Figure 1 - Mean Relative Performance



SYSTEM	Intel 432 8MHz	Motorola 68000 8MHz	Motorola 68000 16MHz	Digital VAX-11/780	Intel 80286 8MHz	Intel 80286 10MHz
LANGUAGE	Ada	C	Pascal	Pascal	Pascal	Pascal
BENCHMARK WORD SIZE (bits)	16	32	32	32	16	16

note: Benchmark times for the 432, 68000, and 80286 are taken from the Berkeley measurements (1,2).

In conclusion, we believe that one should be extremely cautious in using benchmarks such as these to compare the performance of computer systems, hardware or software. It is nearly impossible to attribute the performance of these benchmarks to any one factor. Of the factors involved, however, the effect of architecture is probably one of the smallest for these benchmarks. Finally, it is especially misleading, although certainly tempting, to use a single scalar measure of performance such as the average execution time of a set of benchmarks to compare computer systems.

Acknowledgments

We would like to thank Bob Supnik who measured the performance of these benchmarks using several VAX compilers. Steve Hobbs provided a detailed analysis of the benchmarks, pointing out numerous difficulties in using them to compare compiler quality. Dave Patterson supplied copies of the benchmarks used in the Berkeley measurements.

References

[1] P.M. Hansen, M.A. Linto, R.N. Mayo, M. Murphy, and D.A. Patterson, "A Performance Evaluation of the Intel iAPX 432", Computer Architecture News, 10(4), June 1982, pages 17-26.

[2] D.A. Patterson, "A Performance Evaluation of the Intel 80286", Computer Architecture News, 10(5), September 1982, pages 16-18.

[3] P. Anklam, D. Cutler, R. Heinen, Jr., and M.D. MacLaren, Engineering a Compiler: VAX-11 Code Generation and Optimization, Digital Press, 1982.