Multimedia SPECmarks: A Performance Comparison of Multimedia Programs on Different Architectures

Hari Balakrishnan **Rahul Garg**

{hari,rahul}@CS.Berkeley.EDU

Computer Science Division,

Department of Electrical Engineering and Computer Science,

University of California at Berkeley,

Berkeley, CA 94720.

Abstract

The field of multimedia systems is becoming increasingly important in both the research and commercial worlds. Multimedia programs impose demanding and stringent requirements on all aspects of a computer system. However, there is a dearth of good benchmarks for such software, and there has been no rigorous performance comparison of different systems for such programs. In this paper, we describe a multimedia benchmark suite consisting of six programs and present the results of running these programs on five different systems. Our results show that in general, integer performance is not a good metric to use to predict multimedia performance. For these programs, performance on our DEC 5240 and HP 9000/730 had a high degree of correlation with SPECInt92, but varied between 48% and 118% for the DEC Alpha 3000/400 and between and 60% and 85% for the SPARCStation 20/51 we measured, relative to SPECInt92. Extensive cache simulations of three of these programs showed miss rates between less than 1% and 20% for different programs. However, based on a few microbenchmarks, we show that it is not cache behavior that causes degraded performance, but load delay cycles (on the Alpha), and inferior branch performance and software multiplication (on the Sparc).

1.0 Introduction

The field of multimedia systems is a new one and is becoming an increasingly important one in both the research and commercial worlds[CACM91]. Multimedia software tests all aspects of a computer system -- the processor, the memory subsystem, and the I/O subsystem (both disks and display). Since it is a relatively new field, there is dearth of good benchmarks and there have been few quantitative and comparative studies of the performance of different computer systems on these programs. These programs are especially interesting not just because of their increasing popularity, but also because they impose fairly severe and demanding requirements on the system to perform acceptably well. In fact, certain compute systems are not equal to the task and this has led to the development of specialized hardware support for some of the tasks performed by many multimedia programs such as decoding video frames. The lack of comprehensive performance studies of multimedia applications has also led people to purchase systems for multimedia software on the basis of integer performance alone, which seems logical since these programs typically don't involve any floating point operations, but which is not substantiated by any real studies (and therefore may not be true at all). One measure of the maturity of a field is the existence of quantitative results and benchmark figures, and we feel that the number of programs in the area of multimedia has become large enough to warrant a comprehensive study and evaluation of different computer systems for such software.

The contributions of our work are two-fold: first, we describe a set of six programs that constitute our multimedia benchmark suite, and justify the inclusion of these programs in the suite. Second, we present the results of running these programs on four different computer systems (a fifth is in progress), and discuss in detail the reasons for the observed relative performance. The systems we studied are summarized in Table 1. In addition, we also identify the different architectural bottlenecks for these programs on the different systems.

Our results show that integer performance, as quantified by SPECInt92[SPEC] performance is in general not a good metric to use in order to predict multimedia performance. Multimedia performance on the HP 730 has a high degree of correlation with SPECInt performance, but this is far from true for the DEC Alpha and the Sparc 20. Multimedia performance on these platforms varies between 41% to 118% of SPECInt92 on the Alpha, and between 60% and 85% on the SPARC. We discuss the reasons for the lack of scaling with integer performance that we observe on these two architectures. Finally, we present the results of extensive cache simulations of three of these programs that show that miss rates vary from less than 1% to 20% for different programs on the same architecture. However, we argue that the reason for degraded

Machine	**SPECInt92**	**MHz**	**Cache parameters**
DEC 3000/400 (Alpha)	74.7	133	L1: 8KB (I) + 8KB (D), d-m L2: 512 KB unified, 2-way
SPARCStation 20/ 51	73.6	50	L1: 20 KB (I) + 16 KB (D), d-m L2: 1 MB unified
HP 9000/730 (PA-RISC)	47.8	66	128 KB (I) + 256 KB (D), d-m
DEC 5000/240	27.9	40	64 KB (I) + 64 KB (D), d-m

TABLE 1.

performance on the Alpha and the Sparc is *not* cache behavior but load delay cycles (on the Alpha) and inferior branch performance coupled with software integer multiplication[1] (on the Sparc). We substantiate these claims by a few microbenchmarks that mirror the critical loops performed in these programs.

The rest of this paper is organized as follows. In Section 2, we present the benchmark suite and discuss our benchmarking methodology. We present the details of our experimental results in Section 3, and explain observed performance on the basis of a few microbenchmarks in Section 4. This section also has the results of several cache simulations of these programs. We conclude in Section 5 with suggestions for future work.

2.0 Benchmarks and Methodology

In this section, we describe the different programs we studied and the benchmarking methodology we used. Before doing so, we define what we mean by a multimedia program, since this is essential to understanding why we measured the programs we did.

2.1 What is Multimedia?

Recent advances in computer technology have resulted in a new class of applications called multimedia applications. Video conferencing, video on demand and hypermedia (such as Mosaic) [CACM91] are examples of such applications. In all these applications, processing of digital video is a very important component and this imposes demanding requirements on computer systems. For example, NTSC quality video with 480x640 dots per frame, transmitted at a rate of 30 frames per second, requires the ability to process 9.2 million pixels per second. The video is almost always compressed before its storage and transmission and is uncompressed while being displayed. The common operations on video data include compression, decompression, display and transmission and they stress all components of a computer system -- the CPU, the memory system, and the I/O system (consisting of both the disks and the display).

There are several algorithms for compressing video, the most popular one is called MPEG (Motion Picture Experts Group) [GAL91] and is being proposed as a standard. Similarly, JPEG (Joint Photograph Experts Group) [WAL91] is a commonly used algorithm and standard for compressing still pictures. Most video playback algorithms involve computing the reverse discrete cosine transform (RDCT), which is a CPU-intensive operation that involves integer additions and multiplications and a CPU- and memory-intensive operation called dithering[AMR94]. Similarly, the encoding of video images into com-

1. This is with the standard compiler options that we used on all the platforms with gcc. With the -mv8 option, gcc compiles using the hardware multiply instruction.

pressed form involves computing the inverse operations (forward DCT). Detailed explanations of these techniques can be found in [GAL91].

2.2 The Benchmark Programs

Armed with the above description of what constitutes multimedia software, we decided to choose the following six programs to measure and study on our platforms. A summary of these programs is shown in Table 2

Number	Program Name	Description
1	CJPEG	Independent Software Group's JPEG encoder. Converts from PPM to JPEG format.
2	DJPEG	Independent Software Group's JPEG decoder. Converts from JPEG to PPM format.
3	MPEGHIGH	Berkeley MPEG player with high quality (dither of fs4). Plays mpeg clips on screen.
4	MPEGLOW	Berkeley MPEG player with low quality (gray dither). Plays MPEG clips on screen.
5	ENCODE-HIGH	Berkeley MPEG encoder with dither of fs4. Converts from PPM to MPEG format.
6	ENCODE-LOW	Berkeley MPEG encoder with gray dither. Converts from PPM to MPEG format.

TABLE 2.

We focused on programs that were common and standardized, rather than on more recent software with possibly better algorithms, since these standard ones are the most commonly used. We measured the performance of both decoding and encoding of video frames. MPEGHIGH and MPEGLOW are the Berkeley MPEG player with very different playback qualities, and show very different program behavior. This is true for the corresponding encoding programs as well. One weakness of this set of programs is the bias towards video-based software, and no emphasis on hypermedia applications such as Mosaic. This was mainly because of lack of time; we intend to study the architectural performance and needs of such programs in the future.

2.3 Benchmarking Techniques and Tools

We now describe the way in which we measured the performance of the above programs and the tools we used. We repeated the experiment several times for each program on many different data files to cover the entire spectrum of available frame sizes, encoding, and other relevant parameters. Overall elapsed time was the metric we used as the measure of performance since that is the most unambiguous and precise measure of the performance of a program.. To standardize our operating environment and focus on architectural differences, we standardized the compiler and optimization options; in all experiments, we used the GNU C compiler (*gcc*) at its best machine-independent optimization level (*-O4* and *-funroll-loops*). Our goal was to focus on architectural differences and their impact on performance and we didn't want to be confused by spurious differences in performance introduced by differences in compiler optimizations. In order to measure elapsed times, we used accurate timers on all the systems: we used cycle counters accurate to one clock cycle on the Alpha and the HP 730, that introduced overheads of the order of only a very small number of instructions, while on the DEC 5240 and the Sparc, we used high resolution timers (of the order of a few microseconds, at most) with a system call (*gettimeofday()*) interface.

We started with the idea of doing accurate profiling of programs using a tool like *prof* or *gprof* [GKR82]. However, preliminary results quickly convinced us of the unreliability and inadequacy of these tools for our purposes. Such experiences are also reported in [VAR93]. The main reasons for this are:

- *I/O (disk and display) time:* Most profiling tools work by sampling the program counter at regular intervals (typically once every clock tick, usually 60 times a second) and extrapolating the results of these samples over the run time of a program to produce the profile information. However, when a process is blocked waiting for an I/O action to complete, it relinquishes possession of the CPU, which means that these times are not measured by such sampling-based methods. As we will see in Section 3, there is significant I/O activity in many of our programs, and therefore profiling tools are inappropriate for completely measuring these programs.
- *Short runs:* For many inputs, several of our programs finish executing in just a few seconds. Sampling-based statistical tools are accurate only if the run-time of the profiled program is sufficiently long.

We did, however, use *gprof* to get a broad idea of where each program spends most of its time, and instrumented those functions using accurate timers. On the DEC Alpha, we used a *process cycle counter*, accurate to the granularity of a clock cycle (7.5 *ns* in our system). The cycle counter is a 32-bit register that is updated every clock cycle and can be read in one instruction [ALPHA]. Such an instruction is present in HP's PA-RISC architecture too, where it is called an *interval timer* [HP]. This register can also be read in one instruction. On the Sparc 20 and the DEC 5240, we used high resolution timers accurate to a few microseconds, with a system call interface (a high resolution *gettimeofday()* call). Overall, we found these timers much more accurate and reliable than profilers, and feel that future architectures would do well to include such tools to encourage and facilitate better performance studies.

3.0 Experimental Results

We ran these benchmarks on four different platforms: a DEC Alpha 3000/400, a SPARCStation 20/51, an HP 9000/730 and a DEC Station 5000/240. Measurements on a Pentium 90-based PC are in progress. This section describes the results we obtained and compares multimedia performance with rated SPECInt performance.

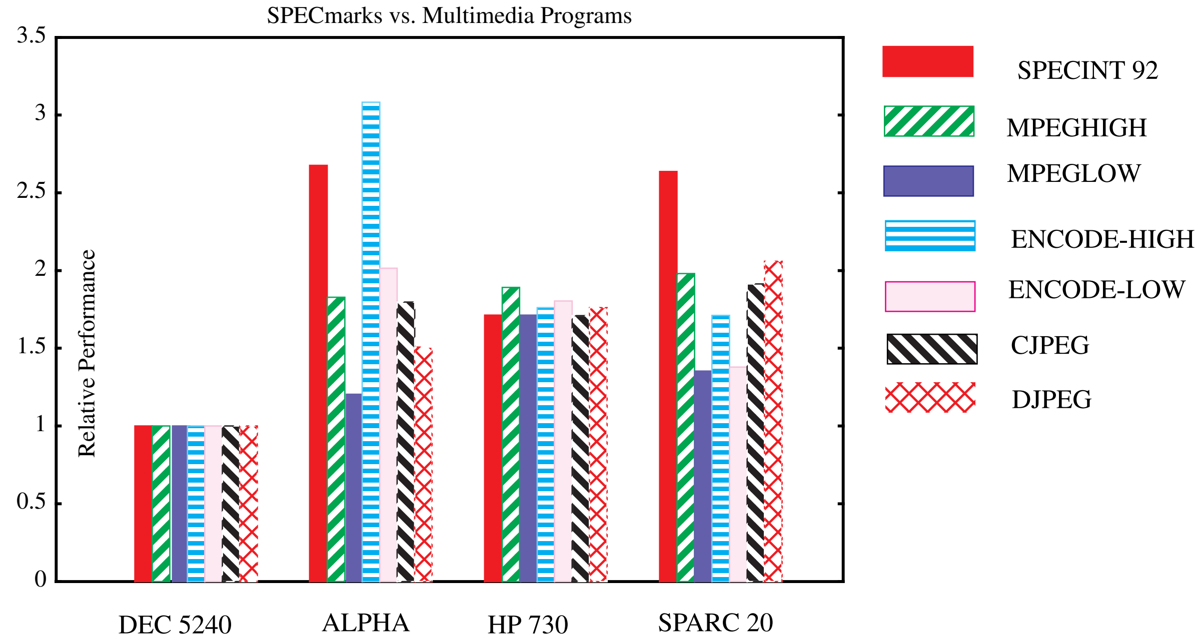
The first set of results, using overall elapsed time as a metric, is shown in Figure 1(a). This figure shows performance normalized with respect to the DEC 5240. As we can see, SPECInt performance on the HP 730 and the DEC 5000/240 is a good predictor of multimedia performance, but this is far from true for the DEC Alpha and the Sparc 20.

One reason for the lack of scaling with SPECInt performance could be the time spent in I/O activity. SPECInt is a measure of the CPU performance, whereas I/O performance is an important component of these multimedia applications. We timed the functions involving I/O activity and calculated the fraction of the time these programs spend doing I/O. The results are summarized in Table 3. For MPEGLOW, the

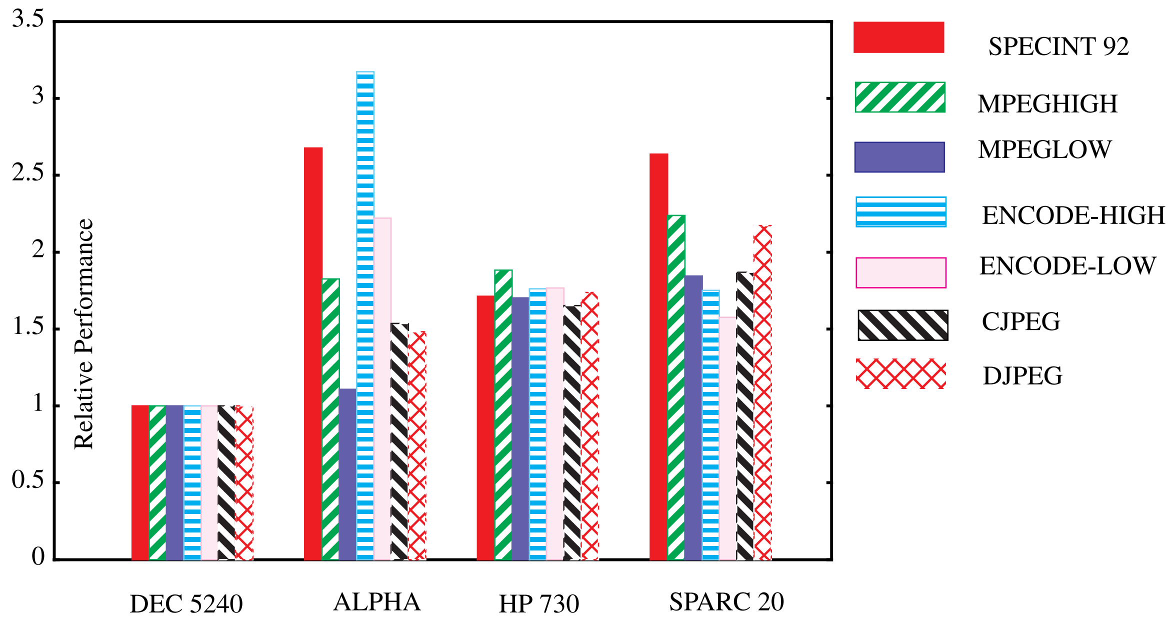
Program	DEC Alpha	HP 730	Sparc 20	DEC 5240
CJPEG	6.58%	17.40%	18.20%	20.16%
DJPEG	13.36%	12.49%	18.19%	13.69%
MPEGHIGH	6.76%	6.75%	17.84%	7.15%
MPEGLOW	**18.61%**	**22.61%**	**43.53%**	**23.06%**
ENCODE-HIGH	*0.91%*	*1.57%*	*3.67%*	*1.53%*
ENCODE-LOW	4.14%	11.92%	24.52%	13.75%

fraction of total time spent in I/O varies from 18.62% to 43.53%; for ENCODE-HIGH the I/O activity is relatively unimportant, since the fraction of time spent in I/O varies only between 0.91% and 3.67%.

Since SPECInt doesn't reflect the I/O performance of the computer systems, the total execution time of the programs may be very different from that predicted by SPECInt. We subtracted the I/O times from the total execution time of the programs to compute their CPU times. Figure 1(b) compares the relative performance of these programs excluding I/O time. We again see that the performance of HP 730 scales as predicted by SPECInt whereas the performance of the Alpha and Sparc 20 are different for different programs and show little correlation with SPECInt. The Alpha performs fastest on ENCODE-HIGH and slowest on MPEGLOW. Sparc 20 has poorest performance of ENCODE-LOW and best on



(a) Relative performance based on overall elapsed times



(b) Relative performance excluding I/O (disk/display)

FIGURE 1. Relative performance of the 6 different multimedia programs and SPECInt92 normalized with respect to the DEC 5000/240.

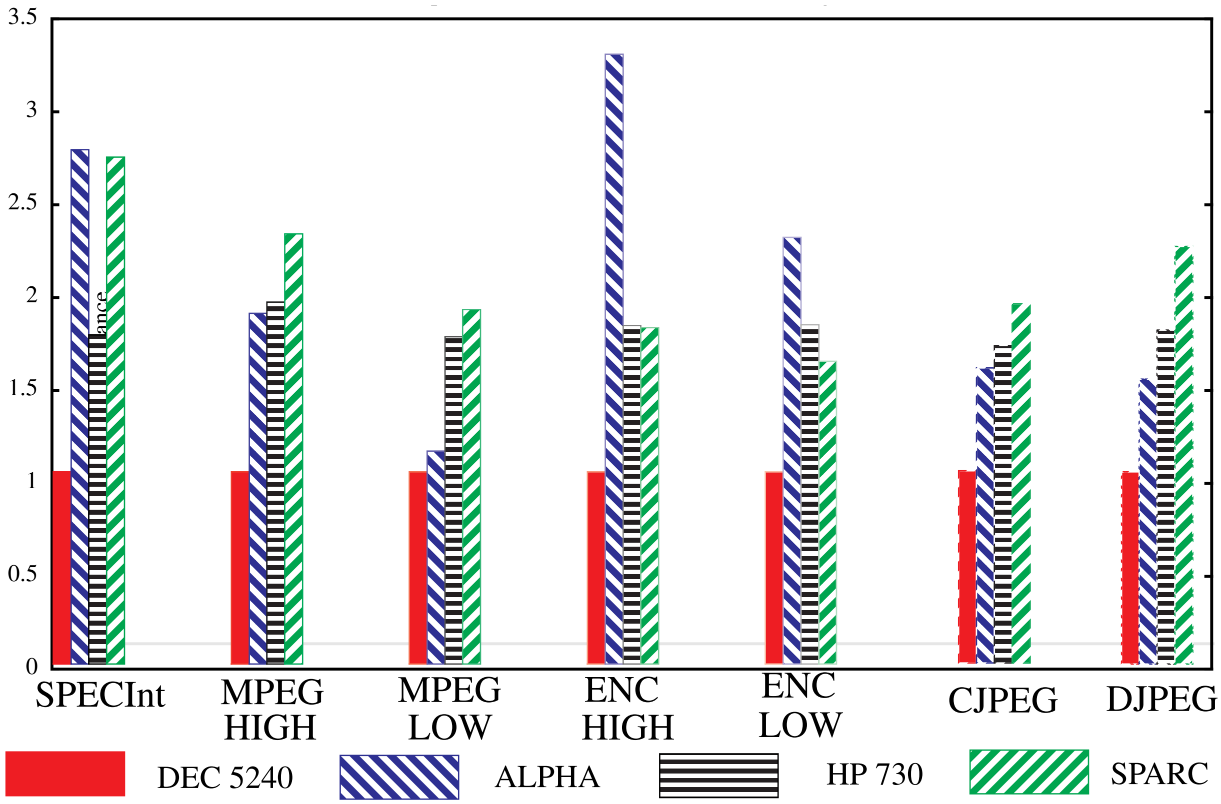


FIGURE 2. Comparative performance of multimedia programs and SPECInt grouped program-wise to highlight differences between different architectures.

MPEGHIGH. We also note that for the Alpha and Sparc 20, the performance of most of the programs is lower than that predicted by SPECInt.

Another informative way of interpreting the performance of these programs on the different platforms is shown in Figure 2. This figure groups together the relative performance for the same program on different machines. For four of the programs -- MPEGHIGH, MPEGLOW, DJPEG and CJPEG, the performance order is SPARC 20 > HP 730 > ALPHA 3400 > DEC 5240, while the Alpha performs well only on the MPEG encode programs.

In the next section, we discuss the results of several cache simulations and microbenchmarks and examine the reasons for these results.

4.0 Cache Simulations, Microbenchmarks, and Explanations

In this section, we explain the reasons for the observed performance on the different platforms. Since all these multimedia programs have no floating point operations, we expected SPECInt performance to be a good predictor of multimedia performance. What we found was that while this hypothesis was fairly accurate (to within 10%) for the DEC 5000/240 and the HP 9000/730, this was far from true for the

DEC Alpha and the SPARCStation 20/51. In other words, for these programs, it seemed like the newer, superscalar architectures with two-level caches and superior memory systems and processors were doing worse than expected.. And as a final twist, the performance for ENCODE-HIGH (high-quality encoding of MPEG frames) on the Alpha was 18% *better* than that predicted by SPECInt!

In the remainder of this section, we explain these results using a couple of different techniques: *cache simulations* and *microbenchmarks*. It is often the case that the performance of the memory system has a lot to do with the performance of a program. This is especially true in our situation since we have excluded I/O activity and are trying to explain the performance of the remainder. One of the important methods of testing this is by measuring the cache performance of the program with the relevant cache parameters, and computing the miss rate and miss penalty, and by determining the average memory access time for the program. The next section describes the results of the cache simulations of these programs for different cache sizes and parameters.

4.1 Cache Simulations

We carried out cache simulations of three programs that showed high deviation in behavior from SPECInt -- MPEGHIGH, MPEGLOW, and DJPEG. We used *pixie* [PIX91] to generate memory access traces for these programs from the runs on the DEC 5000/240. Ideally, we would have liked to use memory traces generated on the Alpha, but the version of pixie on the Alphas available to us didn't produce full memory traces. We then used *dinero* [HILL89] to perform the cache simulations for a whole range of cache sizes. We performed these simulations using two different input data sets (video clips of different sizes) for MPEG, and did not find big differences in the miss rates. The Alpha and Sparc have two-level caches with a large second-level cache (512 KB and 1 MB respectively). For these machines, we did two-level cache simulations with the appropriate cache parameters. The miss rates for the second-level caches were very low, with virtually all the misses being compulsory misses. In addition to finding the miss rates on our machines, we also did the simulations for a range of cache sizes (for a one-level cache). In these simulations, all the parameters except the cache size were held fixed (block size of 32 bytes, direct-mapped and random replacement). The results of these simulations are shown in Figure 3.

For DJPEG the cache miss rates vary from less than 1% (256 KB) to 20% (8 KB). However, the performance of DJPEG on the Alpha and Sparc are explained only in part by these cache simulations. These cache simulations are for a write-back, direct-mapped cache. This is true for the Sparc, but not for the Alpha, which has an 8KB *write-through cache* with *write buffers*. When we did the same simulations for a write-through cache without write buffers, we obtained a miss rate of about 29% for an 8KB cache

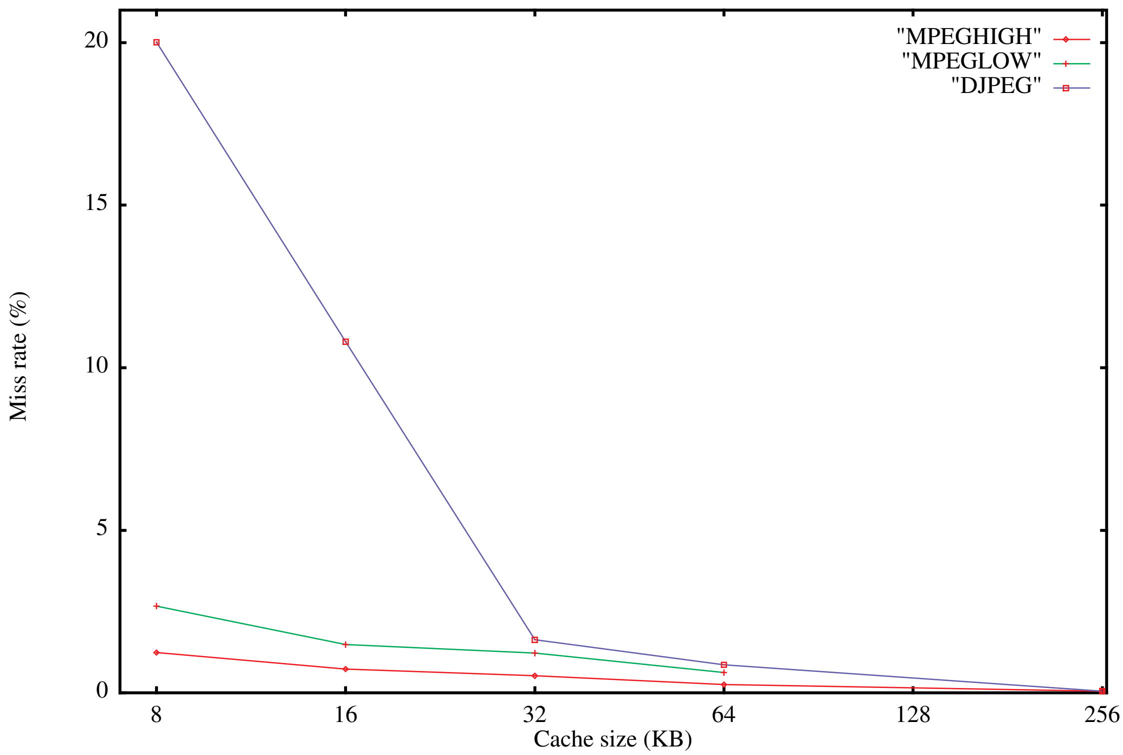


FIGURE 3. Miss rates vs. Cache Size (log scale) for 3 multimedia programs. The cache has a block size of 32 bytes and is direct-mapped.

size. However, this is in the absence of any write buffers, and in reality the miss rates are likely to be about the same as in the write-back case.

For MPEGHIGH and MPEGLOW, the miss rates were quite low for all sizes. Even for a small cache size of 8 KB, the miss rate for MPEGLOW is only about 2.6%. Figure 4 is a scatter plot that shows the miss rates for these three programs and the SPEC programs on the Alpha and the Sparc. The miss rates for the MPEG programs on both machines are *less* than those for the SPEC programs. Yet, the relative performance for the SPEC programs is much *better*! This means that these cache simulations don't fully explain the reason for the degraded performance for these programs on the Alpha and Sparc. In the next section, we describe a few microbenchmarks on the basis of which we explain the performance of these programs.

4.2 Microbenchmarks

Since the cache simulations don't adequately explain observed performance, we decided to take a closer look at the profile information and run a few microbenchmarks to explain these results. We profiled the programs showing most deviation from their rated SPECInt values and wrote microbenchmarks similar to the critical code of these programs. We then measured the performance of these microbenchmarks on all

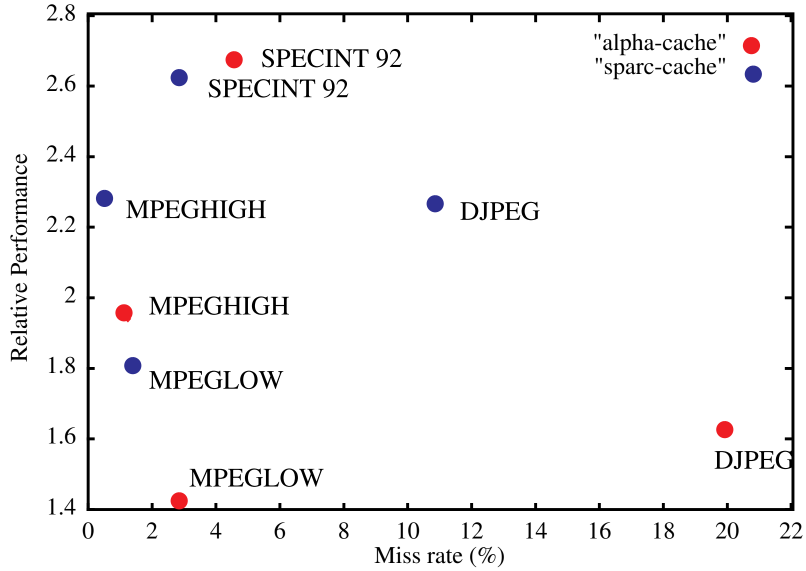


FIGURE 4. Scatter plot showing performance vs. L1 cache miss rates on the Alpha and Sparc for three multimedia programs and SPECInt. The miss rates for MPEG programs are less than for SPECInt, but so is their performance!

the machines. In the remainder of this section, we describe these microbenchmarks, and then present the results of running them on our different machines.

- **Branches, loads and stores:** On the Alpha and Sparc, a significant portion of the execution time (25%) for the program MPEGLOW is spent in a function that checks if the decoded values of the coefficients are within range (if there is an error in the encoding of certain types of frames, the assertion fails). This code has a loop containing a series of array assignments (loads and stores) with several *assert()* statements (that compile to branch instructions). This portion of the code was extracted to get the microbenchmark BR+LD+SW.

- **Branch performance:** In order to evaluate the branch performance on the various architectures we removed the memory accesses from the BR+LD+SW benchmark. The remaining code had only assert statements. This is a common and time-consuming part of all our programs. We call this microbenchmark BRANCH.

- **Loads and stores:** Low quality (gray) dithering is another time-consuming operation in MPEGLOW on the Alpha and the Sparc. The code consists of a series of table lookups (array assignments, loads and stores) in a manually unrolled loop with a small number of integer additions. We based the LOAD-STORE microbenchmark on the basis of this function. In general, this models the performance on loads and stores (e.g, array copying) for any machine.

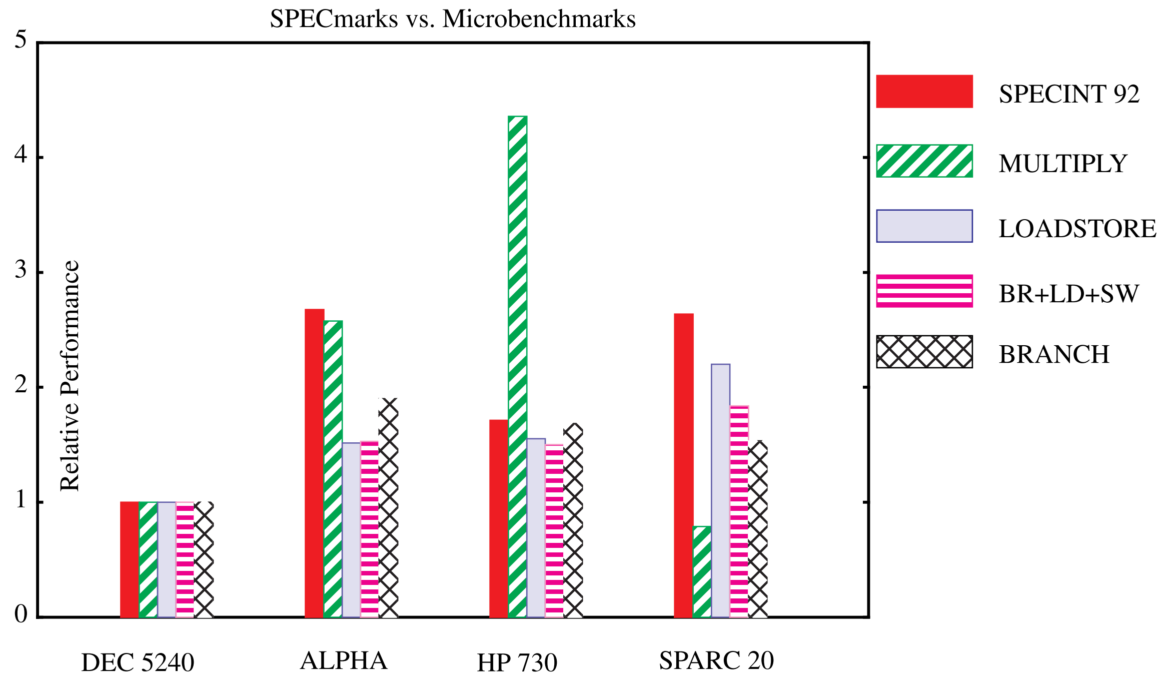


FIGURE 5. Relative performance of microbenchmarks vs. SPECInt performance

- **Integer multiplication:** Computing the reverse discrete cosine transform (RDCT) is another major time consuming operation. On the Alpha, 21% of the time was spent in this function on MPEGLOW. This percentage was much higher (44%) on the DEC 5000/240 that has a much slower CPU. Integer multiplication is an important component of this function. The code uses a macro which typecasts its operands (if possible) to 16 bit numbers before the multiplication. This is done to make the multiply go as fast as possible. The microbenchmark MULT measures integer multiply performance.

Figure 5 shows the relative performance of these microbenchmarks. From the figure we see that performance of the Alpha on MULT scales as predicted by the SPECINT92. HP 730 has extremely good performance on MULT whereas Sparc20's performance is very poor. On the Sparc20, we used standard compiler options to gcc (*-O4 -funroll-loops*), and this defaults to the software multiply routine which in turn results in poor performance of the system on RDCT. We modified the program to have fewer multiply instructions in the loop and found that the relative performance of the modified program was worse. The HP730 has a pipelined integer multiply unit, and this was evident in the increasing relative performance as the number of successive multiply operations in the loop increases (until the depth of the pipeline).

From Figure 5, we find that the relative performance of the Alpha on LOADSTORE is 1.51 (normalized with respect to the DEC 5000/240), whereas according to SPECInt92 it should be 2.68. The assembly code for this microbenchmark has two dependent loads in succession followed by a store in the critical loop. ALPHA has a 2 cycle delay slot, which is the reason it performs so badly on LOADSTORE. Even two successive loads operating on independent data requires one intervening delay slot. All the other machines scale according to SPECInt92.

ALPHA doesn't perform well on LD+SW+BR benchmark. Its relative performance is only 1.53. The assembly code of this benchmark consists of two dependent load instructions and a store instruction as in the LOADSTORE benchmark, followed by add, shift, compare and branch instructions. Each instruction depends on the result of the previous instruction. The superscalar design of ALPHA doesn't help in speeding up this program as the instructions show high dependency. HP730's performance scales well as predicted by SPECInt92, whereas SPARC20's performance is only 1.84 as opposed to the predicted value of 2.64. The reason for this is that the program is unable to efficiently utilize the SPARC20's superscalar design; in addition, the branch performance of the Sparc20 is a little poor.

The assembly listing of the BRANCH program in very similar to that of LD+SW+BR program except that in the two dependent loads and one store are absent in the loop. We see that relative performance of ALPHA is better as compared to LD+SW+BR. Since the dependent load and store instructions are absent in this benchmark, the performance of ALPHA is slightly better because now the processor doesn't have to stall for one or two load delay cycles. HP730's performance scales as expected, but the SPARC20's performance is bad on this benchmark. The processor may have a poor branch prediction scheme, or it might just be the case that for this particular program the branch prediction scheme doesn't work well.

We can explain the performance of these programs on the basis of these microbenchmarks. We see that three of the four microbenchmarks for MEPGLOW perform badly on ALPHA explaining why MPEGLOW doesn't perform very well on ALPHA. On the HP 730, three of these benchmarks have performance as predicted by SPECINT92 and one of them performs better than SPECINT92 (there isn't as much exploitation of the multiplication pipeline possible in MPEGLOW as in the microbenchmark modeling multiplication). The net result is that the performance of MPEGLOW is very close to the SPECINT92 performance on the HP machine. On SPARC20 all the microbenchmarks have relatively poor performance. The MULT benchmark has the worst performance of 0.79 whereas SPECINT92 has a performance of 2.64. This can explain the poor performance of SPARC20 on MPEGLOW program.

We can quantitatively explain the performance of most of the programs on the Alpha and Sparc20. In this section, we show these calculations for both MPEG programs on these machines.

Program	*RevDct (int)*	*LDSWBranch*	*Branch*	*Dither*	*Multiply*	***Performance***
MPEGLOW	24.3%	21.8%	16.5%	8.4%	10.9%	**1.96X (pre)**
	2.28X	1.84X	1.53X	2.20X	0.90X	**1.81X (obs)**
MPEGHIGH	9.4%	8.6%	8.7%	60.8%	4.1%	**2.36X (pre)**
	2.28X	1.84X	1.53X	2.64X	0.9	**2.29X (obs)**

TABLE 3.

Program	*RevDct (int)*	*LDSWBranch*	*Branch*	*Dither*	***Performance***
MPEGLOW	20.6%	23.4%	13.4%	25.2%	**1.95X (pre)**
	2.28X	1.84X	1.53X	1.51X	**1.35X (obs)**
MPEGHIGH	11.0%	14.9%	7.10%	59.6%	**2.04X (pre)**
	2.28X	1.84X	1.53X	2.04X	**1.88X (obs)**

TABLE 4.

Table 4 and 5 show the percentages of time spent by the two programs in different functions on the Sparc20 and Alpha respectively. In each case, the top number in any cell gives the percentage of time spent in the function and the lower number the relative performance based on our microbenchmarks. The last column has our predicted performance (pre) and the observed performance (obs). For the Sparc the predicted performance of both programs is fairly close to the observed one. The reason MPEGHIGH is predicted better than MPEGLOW is because the former has a much lower cache miss rate. On the Alpha, the prediction for MPEGHIGH is within 10% of the observed performance, but the prediction for MPEGLOW is far from the observed one. The cache miss rate is only about 2.6%, and is not significant enough to cover this difference. However, we feel that our microbenchmark approach to explaining performance is by and large correct in predicting performance to within 10 or 15% in most cases.

Finally, an interesting observation is the extremely good performance of ENCODE-HIGH on the Alpha. We profiled the program and found that more than 90% of the total execution time was spent in one function. This function contained a small computation (one addition and one absolute value computation) inside a loop that had been unrolled manually in the C code to get better performance. From the assembly listing of that function we discovered that the compiler was able to fill the load delay slot with two independent instructions (that often executed in parallel on the superscalar architecture) and used the conditional move instruction to compute the absolute value (rather than a more conventional branch). This resulted in very good performance on this machine (118% of the SPECInt performance). However, the assembly listing of the same code on SPARC20 had branches to compute the absolute value, resulting in relatively poor performance.

5.0 Conclusions and Future Work

Our contributions in this paper are two-fold: first, we described a set of six multimedia programs that constitute a benchmark suite and justified the inclusion of these programs on the basis of their being standards and their popularity in the field. We then presented the results of running these programs on four different hardware platforms -- a DEC 5000/240, an HP 9000/730, a DEC Alpha 3000/400, and a SPARC-Station 20/51 (measurements on a Pentium 90-based PC are in progress). Our results show that in general, integer performance as approximated by SPECInt92 is not a good predictor of multimedia performance. On the DEC 5000/240 and HP 9000/730, SPECInt ratings seem to predict multimedia performance quite well, but this is far from true for the newer, superscalar architectures like the Alpha and the Sparc 20. We performed several cache simulations and found highly variable miss rates from less than 1% to 20% depending on the program for an 8KB cache. However, we also found that cache behavior did not adequately explain observed performance, and measured the performance of a few microbenchmarks. These showed that the primary reason for poor performance on the Alpha was load delay stalls even when the data was present in the cache. On the Sparc, the reasons were poor branch performance, as well as software integer multiplication (this was the default option chosen by the compiler; later measurements for 2 of the programs with an option to use the hardware instruction showed performance improvements between 5 and 10%). Finally, one of the programs (MPEG encoding with high quality) showed extremely good performance (118% of SPECInt) because of aggressive use of superscalar instructions and conditional moves rather than branches.

Based on these results an experience with the Alpha and Sparc 20 architectures, we conclude that it is much harder to predict the performance of general programs on the newer, superscalar architectures. In addition, the promise of high performance isn't always achieved, since high clock rates may have to be traded off for load delay slots (like for the Alpha), and this could mean delays even on cache hits. This also implies that traditional methods of cache simulations are not sufficient to completely explain the memory-system performance of programs on such architectures since the access time on a cache hit doesn't always remain constant.

6.0 Acknowledgments

We thank Srinivasan Seshan, Dave Patterson, and Remzi Arpaci for several useful discussions and for comments on earlier drafts of this paper.

7.0 References

[ALPHA92] Alpha architecture reference manual. Sites, R.L. (Ed.), Digital Press, Burlington, MA. 1992.

[AMR94] Amir, E., McCanne, S., Razavi, H. Performance impacts of architectural optimizations for software video codecs. CS252 class report, Dec. 1994.

[CACM91] Communications of the ACM, April 1991. Special Issue on Digital Multimedia Systems, 34(4).

[DWA*] Dobberpuhl, et al. A 200 MHz 64-bit dual-issue CMOS microprocessor. Digital Technical Journal, Vol. 4(4), Special Issue 1992.

[GAL91] Gall, D. MPEG: A video compression standard for multimedia applications. CACM 34(4). April 1991, pp. 46-63.

[GKM82] Graham, S., Kessler, P., McKusick, M. Gprof: A call-graph execution profiler. In Proc. SIGPLAN 82 Symp. on Compiler Construction. SIGPLAN Notices, 17(6), pp. 120-126, June 1982.

[HILL89] Hill, M.D. Test driving your next cache. Magazine of Intelligent Personal Systems (MIPS), Aug. 1989, pp. 84-92.

[HP] The PA-RISC 1.1 Architecture Reference Manual. Available on http://www.hp.com/nsa/acd.html.

[PSR93] Patel, K., Smith, B.C., Rowe, L.A. Performance of a software MPEG video decoder. In Proc. ACM Multimedia 93, pp. 75-82.

[SPEC] Benchmarking 2: SPEC: System Performance Evaluation Cooperative. Capacity Management Review, 21(8), Aug 1993.

[VAR93] Varley, D.A. Practical experiences of the limitations of gprof. Software - Practice and Experience, April 93, 23(4).

[WAL91] Wallace, G.K. The JPEG still picture compression standard. CACM 34(4), April 1991, pp. 30-44.