

Characterization of File I/O Activity for SPEC CPU2006

Dong Ye
Northeastern University
Boston, MA
dye@ece.neu.edu

Joydeep Ray
Advanced Micro Devices
Austin, TX
joydeep.ray@amd.com

David Kaeli
Northeastern University
Boston, MA
kaeli@ece.neu.edu

ABSTRACT

SPEC CPU2006 is a compute-intensive benchmark suite designed to stress a computer system’s processor, memory subsystem, and compiler. To construct this suite, SPEC has selected benchmarks that are derived from real world applications. When run with their reference inputs, these programs place a significant computational burden on today’s mainstream desktops as well as high-end workstations and servers.

For these applications to thoroughly exercise the merits of a particular processor/memory design point, it is necessary to limit the amount of I/O activity generated. Since these applications come from real world applications, the suite developers have considered how best to limit the amount of file-based I/O activity present in these applications. This paper presents the characteristics of file I/O activity in the resulting suite and its overall impact on the performance of these applications. We also report on some of the choices SPEC has made in order to reduce the file I/O activity in some specific programs of the suite.

1. INTRODUCTION

The SPEC CPU benchmark suites are designed to measure the performance of a computer system’s processor and memory subsystem, as well as the platform’s supporting optimizing compiler’s code quality. The SPEC CPU suites are used both in industrial environments [8] and by the academic community [3]. These suites have undergone five major releases since SPEC’s inception in 1989. Its latest release, CPU2006 (V1.0), was released in August 2006 [4].

I/O operations are a necessary part of the execution of any program. There is presently a substantial performance gap between the speed of processor/memory and the supporting I/O subsystem. Therefore I/O operations tend not to enjoy any benefits from further improvements made to the underlying processor core and memory subsystem. Compiler technologies aimed at improving I/O performance are better evaluated with dedicated workloads instead of the SPEC CPU suites [9]. Any significant amount of I/O activity would hinder the SPEC CPU suites from fulfilling their design goals.

The SPEC CPU subcommittee provides a set of guidelines [10] to address this concern when selecting candidate

benchmarks. For CPU2006, SPEC has required that all its constituent programs must spend at least 95% of their execution time in the submitted code and at least 95% of the execution time is compute bound. We chose to look at the user time percentage reported by the *top* utility as a useful metric: a program that spends at least 95% of its execution time in user space is guaranteed to meet the second requirement; and a program that spends less than 95% execution time in user space fails the first requirement. Another design goal of the SPEC CPU suites is to establish a close correlation between the performance measured by these suites and the performance of real world applications. One way SPEC approaches this goal is by selecting real world programs for the suite [8]. However most real world applications contain some file I/O activity. This paper presents a characterization of the file I/O activity present in the CPU2006 suite (V1.0) and reports on its performance implications. We confirm that the file I/O activity of CPU2006 carries little weight in terms of performance impact. We also explain the steps the SPEC CPU subcommittee has taken to reduce the amount of file I/O activity in CPU2006.

The organization of this paper is as follows: Section 2 presents the general characteristics of the file I/O activity present in CPU2006. Section 3 presents the performance impact of file I/O activity. Section 4 briefly reviews some of the effort expended by the SPEC CPU subcommittee to reduce the amount of file I/O in this suite. We then conclude the paper.

Note that the statistics presented in this paper reflect a bugfix that reduces the amount of I/O for the benchmark 453.povray. SPEC plans to include the bugfix in the V1.1 maintenance release for CPU2006.

2. CHARACTERIZATION OF CPU2006 FILE I/O ACTIVITY

First, we present the characteristics of file I/O activity. We have captured a number of statistics related to the file I/O activity, in an attempt to have a first-order understanding of the volume of file I/O activity present in this suite.

The experimental system used to gather this data is described in Table 1. All of the data was collected when running each benchmark program with their reference inputs.

CPU	AMD Athlon™ 64 3400+, 2.2 GHz, single-core, with an integrated single-channel DDR memory controller
Memory	2 DIMMs of 1GB DDR333 memory modules, with peak memory bandwidth 2.66 GB/s
OS	Novell SUSE® Linux Enterprise Server 9 x86-64 Edition [5], Service Pack 3, run level 5, kernel version 2.6.5
I/O	HyperTransport® bus (maximum bandwidth: 4 GB/s), ATA-133 (maximum bandwidth: 133 MB/s)
Compiler	PathScale™ EKOPath™ Compiler Suite 2.4 [7], “-O3” optimization switch turned on, 64-bit binaries built and run for all benchmarks

Table 1: A uniprocessor, single-core system was used to study the overall characteristics of CPU2006’s file I/O activity: its volume and its performance impact.

The reference inputs for some programs are composed of multiple input files. SPEC requires that the same program be executed multiple times over each of these input files in order to be qualified as a reference run. For such benchmarks, we report aggregate statistics of a reference run over all the input files in a reference input.

The Linux tool *strace* was used to collect the information on each system call related to file I/O activity. Using this data, we have calculated the statistics in the Table 2. The middle pane of Table 2 (the six columns next to the column listing benchmark names) presents the following file I/O statistics:

- The number of *open* system calls reported by *strace*,
- The number of *close* system calls reported by *strace*,
- The number of *read* system calls reported by *strace*,
- The number of *write* system calls reported by *strace*,
- The total number of kilo-bytes (1,024 bytes) read from files, calculated by adding up the return values of all the *read* system calls collected by *strace*,
- The total number of kilo-bytes written to files, calculated by adding up the return values of all *write* system calls collected by *strace*.

It is interesting to note that three benchmarks (444.namd, 481.wrf and 482.sphinx3) have some discrepancy between the number of *open* system calls and the number of *close* system calls, which normally suggests a bug exists in the code. After further investigation to all the individual files

upon which mismatched *open* and *close* system calls were observed, we have found that they are all due to the same reason in the source code: an opened file is not closed before the program exits.

3. PERFORMANCE ANALYSIS OF FILE I/O ACTIVITY

Performance characteristics of the file I/O activity are presented in three different studies: (1) Section 3.1 looks at the overall performance impact of file I/O activity during the entire execution of the programs in CPU2006. (2) Section 3.2 investigates the performance impact of file I/O activity under SPECrate runs, where multiple copies of the benchmark are run simultaneously. (3) Section 3.3 takes a closer look at the performance impact of file I/O activity over time for four selected benchmarks (i.e., the histograms of file I/O bandwidth and user time ratio observed during the execution of these programs.)

3.1 Overall performance impact of file I/O activity

The overall performance impact due to file I/O activity is measured using two statistics: 1) the average I/O bandwidth consumed during the entire execution, and 2) the average user time (user space execution time) ratio experienced during the entire execution.

We ran all experiments on the same system as was described in Table 1. The total run time for each benchmark was reported. The average I/O bandwidth during the entire execution for each benchmark was calculated by dividing the total number of bytes read and written by the total execution time. Using the utility *top*, we periodically collected the user time ratio observed through the entire execution. Since a constant sampling frequency was used during the entire run, the average user time ratio for the program’s execution is computed as the arithmetic mean of the ratios collected. The right pane of Table 2 (the rightmost three columns) lists three statistics: 1) total run time, 2) average file I/O bandwidth, and 3) average user time ratio.

In terms of average behavior, the file I/O bandwidth is well below the bandwidth of the hard disk interface. The hard disk interface of the experimental system has a peak bandwidth of 133 MB/s (as shown in Table 1) and is most likely to be limiting factor of filesystem performance. The average user time ratio observed also confirms that the file I/O activity is minimal. We will take a further look at these two characteristic over time in Section 3.3 to gain a better understanding of the performance impact due to file I/O activity.

3.2 Effect of scaling the number of program instances on a multiprocessor versus the file I/O activity.

We are starting to see multi-core and multiprocessing become commonplace in the server and workstation markets [2]. The SPEC CPU suites have been widely used to evaluate the performance of the processor and memory sub-

Benchmark	# of opens	# of closes	# of reads	# of writes	Bytes read (KB)	Bytes written (KB)	Run time (second)	Avg. I/O b/w (KB/s)	User time (%)	
int	400.perlbench	213	213	12,983	42,229	50,951	2,158	1,110	48	98
	401.bzip2	18	18	896	6	112,004	5	1,530	73	98
	403.gcc	72	72	63	5,717	5,535	22,846	1,350	21	99
	429.mcf	8	8	803	59	3,189	232	2,220	2	99
	445.gobmk	569	569	597	4,329	807	10	958	1	99
	456.hmmer	16	16	13,406	13	53,563	40	972	55	99
	458.sjeng	3	3	4	3,376	1	23	1,280	0	99
	462.libquantum	6	6	6	1	3	0	2,740	0	99
	464.h264ref	33	33	1,275	14	50,118	40	1,560	32	99
	471.omnetpp	10	10	10	290	7	1,135	1,310	1	99
	473.astar	24	24	72	2	13,066	2	1,310	10	99
483.xalancbmk	10	10	12	361	75	0	1,540	0	99	
fp	410.bwaves	15	15	13	3	6	4	1,570	0	99
	416.gamess	53	53	14,418	4,022	344,623	158,451	1,880	268	99
	433.milc	6	6	8	19	4	6	1,570	0	99
	434.zeusmp	13	13	13	3	7	7	1,270	0	99
	435.gromacs	16	16	830	1	3,285	0	1,020	3	99
	436.cactusADM	9	9	9	121	5	13	2,160	0	99
	437.leslie3d	13	13	14	17	9	64	1,830	0	99
	444.namd	10	9	1,860	25	7,409	93	878	9	99
	447.dealII	56	56	8	15,690	4	62,472	1,110	56	99
	450.soplex	18	18	37,759	229	301,925	1	1,560	194	99
	453.povray	43	43	85	26,145	258	5,305	489	11	99
	454.calculix	126	126	328	2,151	1,273	8,590	1,520	6	99
	459.GemsFDTD	15	15	228	135	847	537	2,740	1	99
	465.tonto	37	37	213	69	1,844	1,491	1,210	3	99
	470.lbm	8	8	648	3	2,568	10	2,990	1	99
	481.wrf	39	38	2,396	6	21,662	22	1,570	14	99
	482.sphinx3	137	134	3,098	2,972	36,160	3,968	2,420	17	99

Table 2: The I/O activity in CPU2006 and its overall performance impact.

systems of these systems [11, 13]. Given the expanded resources on these systems (multiple processors and/or cores), it is of interest to see how running multiple programs concurrently introduces additional I/O activity and affects the performance of these applications. The system utilized for this study is shown in Table 3.

The experiments involved in this study are as follows: SPECrate [12] runs with 1-4 copies were performed for all the CPU2006 programs (During an n -copy SPECrate run, n processes are run concurrently. Each process runs the same benchmark binary with its own copy of reference input in its own directory.) To reduce the variation introduced by the operating system scheduler, each process was bound to a dedicated processor as recognized by the operating system (The experimental system has four processors, #0-#3, recognized by the operatins system.) The user time ratio on processor #0 was recorded periodically using the same method as described in Section 2. Table 4 lists the

average user time ratio observed on processor #0 during all the four SPECrate runs.

We observed that for the vast majority of the programs in CPU2006, multiple copies of the benchmark program running on a multiprocessor of up to four processors did not increase the I/O activity noticeably. The only exception to this observation was 429.mcf. Due to its large memory footprint (especially when run as a 64-bit binary [14], which is the case for this study), 429.mcf incurs heavy paging activity during both three- and four-copy SPECrate runs. The steady-state memory footprint (resident memory) of a single-copy SPECrate run for 429.mcf with its reference input is about 1.6 GB on this experimental system. In fact the program itself does not incur significant file-based I/O activity, as shown in Table 2.

3.3 Performance impact of file I/O activity over time

CPU	AMD Opteron™ 2218 dual-core processor, 2.6 GHz, two processors on board, dual-channel DDR2 memory controller integrated in each core
Memory	4 DIMMs 1GB DDR2-667 memory modules, with peak memory bandwidth 10.67 GB/s
OS	openSUSE 10.1 x86-64 Edition [6], run level 3, kernel version 2.6.16
I/O	HyperTransport® (maximum bandwidth: 4 GB/s), SATA-I (maximum bandwidth: 150 MB/s)
Compiler	GCC 4.1.1 [1], “-O3” optimization switch turned on to build all benchmarks except 400.perlbench, “-O2” optimization switch turned on to compile 400.perlbench, 64-bit binaries built and run for all benchmarks

Table 3: A multiprocessor (two dual-core processors) system was used to study the performance impact of file I/O activity under SPECrate runs with 1-4 copies.

Real world applications generally incur some I/O activity. Programs commonly read the bulk of their initial memory values from the filesystem upon startup and then write results to files upon completion. Sporadic interactions with the filesystem in the middle of execution may also be encountered. As long as the file I/O activity is concentrated (i.e., lasts only a short period of time) and is infrequent, it has only limited influence on the performance of the application. The SPEC CPU subcommittee has made a deliberate effort to only include programs containing little and infrequent I/O activity.

In this section we investigate the dynamics of the performance impact of file I/O activity over the entire execution for four selected benchmarks. Both file I/O bandwidth and user time ratio were collected using the same method as was described in Section 2. Figures 1- 4 show these two pieces of data plotted over time for the following four benchmarks: 401.bzip2, 416.gamess, 447.dealII, and 450.soplex. These four benchmarks experienced the heaviest average I/O bandwidth usage over their executions (as shown in Table 2).

Each data point in these figures represents the average file I/O bandwidth and user time ratio observed during the preceding 5 seconds of run time. Since different profiling methods were used to collect these two pieces of data, the two figures for each benchmark do not match exactly in terms of its total execution time. However neither of the run times reported under these profiling settings deviates from the native (when no profiling is performed) run time by more than 2%. For all four benchmarks we show that there is infrequent file I/O activity, and we find that the

Benchmark		1	2	3	4
int	400.perlbench	99	98	99	99
	401.bzip2	99	98	98	98
	403.gcc	98	97	98	97
	429.mcf	99	99	82	12
	445.gobmk	98	98	98	98
	456.hmmer	99	99	99	99
	458.sjeng	99	99	99	99
	462.libquantum	99	99	99	99
	464.h264ref	99	99	99	99
	471.omnetpp	99	99	99	99
fp	473.astar	99	99	99	99
	483.xalancbmk	99	99	99	99
	410.bwaves	99	99	99	99
	416.gamess	99	99	99	99
	433.milc	99	99	99	99
	434.zeusmp	99	99	99	99
	435.gromacs	99	99	99	99
	436.cactusADM	99	99	99	99
	437.leslie3d	99	99	99	99
	444.namd	99	99	99	99
	447.dealII	99	99	99	99
	450.soplex	99	99	99	98
	453.povray	99	99	99	99
	454.calculix	99	99	99	99
	459.GemsFDTD	99	99	99	99
	465.tonto	99	99	99	99
	470.lbm	99	99	99	99
	481.wrf	99	99	99	99
482.sphinx3	99	99	99	99	

Table 4: The average user time ratio observed on processor #0 under SPECrate runs with 1-4 copies.

user time ratio observed confirms that file I/O activity has little impact on performance for these programs. In three of four of these benchmarks (416.gamess, 447.dealII, and 450.soplex), we can see that programs or runtime periods that have little file I/O activity tend to spend almost all of their time in user mode.

4. COMMENTS AND CONCLUSIONS

During the development of CPU2006, the SPEC CPU subcommittee measured the amount of file I/O activity in the candidate benchmarks and also took steps to keep I/O under control. For example, in the compression benchmarks like bzip2, the input data is read from memory and the compressed data is written into memory, instead of being read from or written to disk. Another benchmark that performs a significant amount of file I/O is 416.gamess. This benchmark is a computational chemistry program. Some

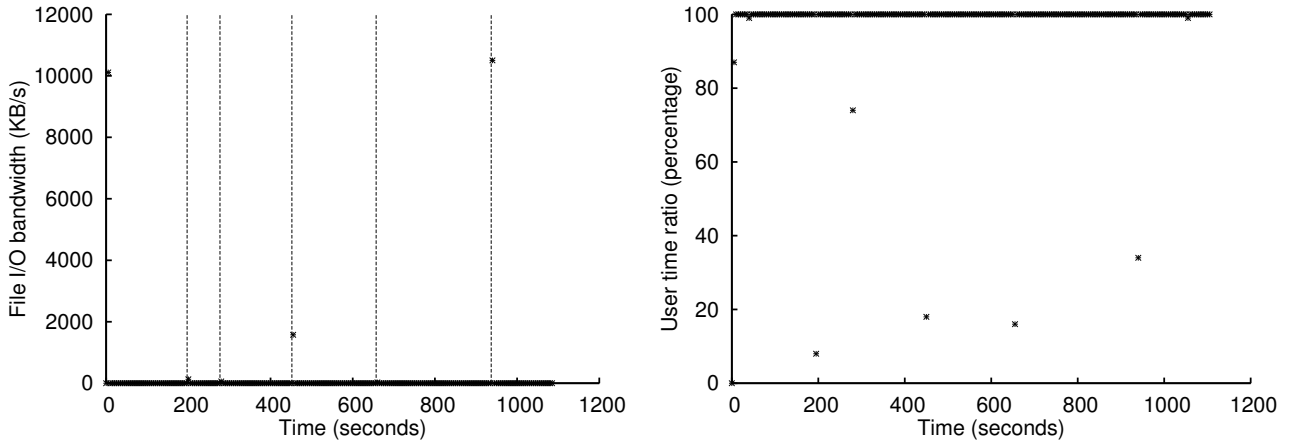


Figure 1: The file I/O bandwidth and user time ratio observed during the execution of 401.bzip2. Note that each vertical dashed line in the left figure indicates that the benchmark changes its input file at that point during the run. (The reference input set for 401.bzip2 is composed of six input files.)

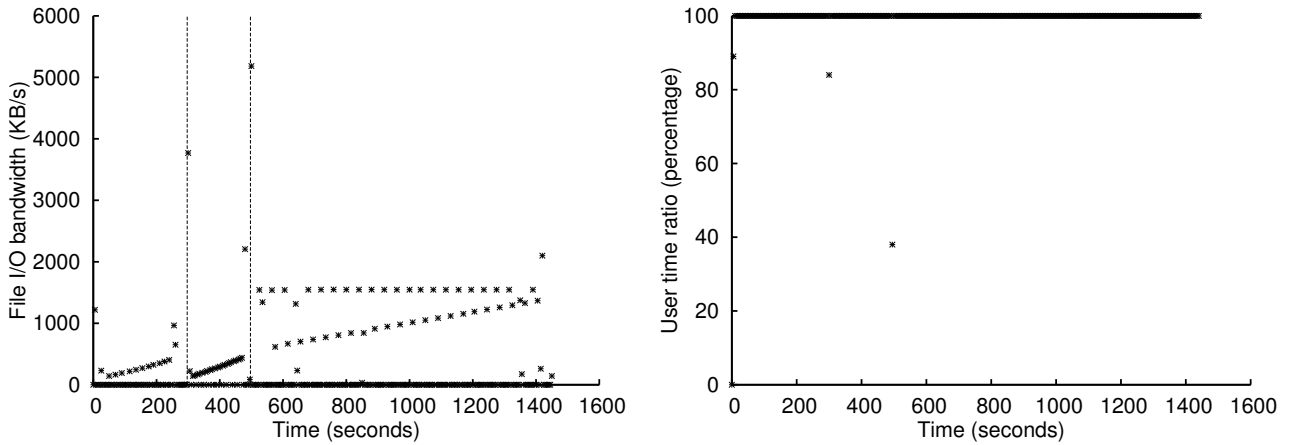


Figure 2: The file I/O bandwidth and user time ratio observed during the execution of 416.gamess.

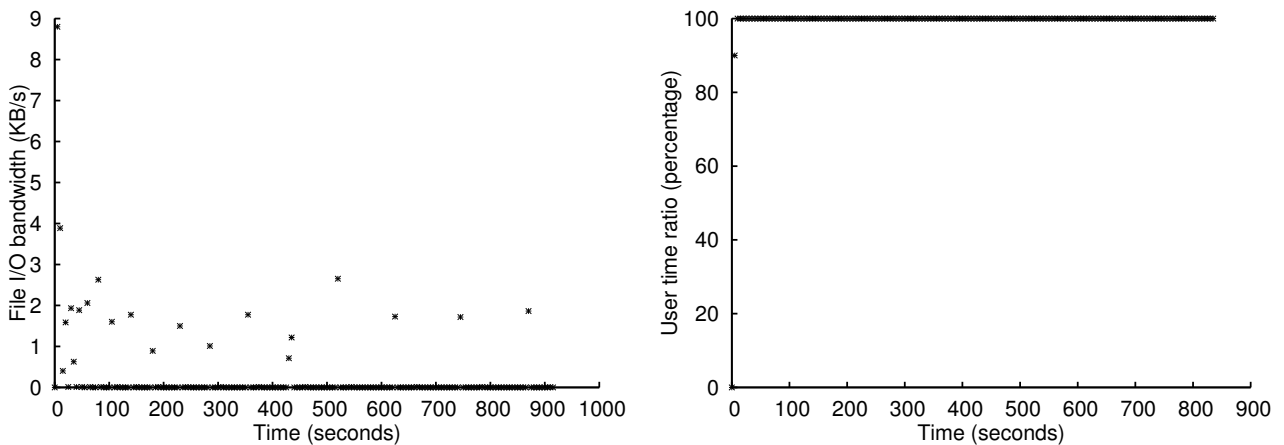


Figure 3: The file I/O bandwidth and user time ratio observed during the execution of 447.dealII.

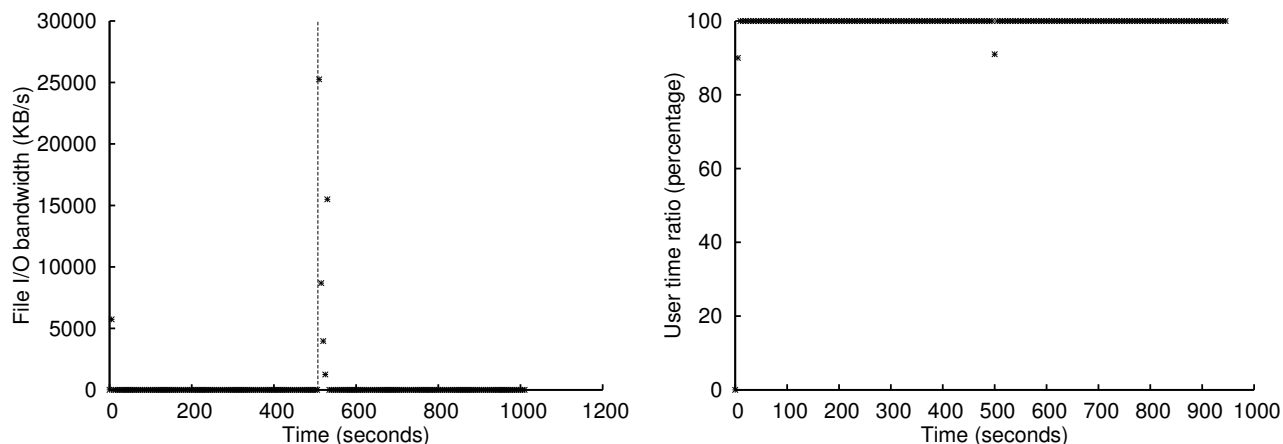


Figure 4: The file I/O bandwidth and user time ratio observed during the execution of 450.soplex.

algorithms used in the original application store intermediate SCF computation results on disk. This causes a relatively heavy amount of file I/O activity. The code used for 416.gamess in the CPU2006 suite only uses the “direct SCF” algorithm for SCF computation, which minimizes the amount of intermediate storage consumed.

In summary, all programs in the CPU2006 suite contain very little file I/O activity. This helps to bolster the position of this suite as a CPU benchmark set to measure the performance of processor, memory subsystem and compiler technology. Our scalability analysis done in this paper indicates that the I/O activity present in this suite and its associated performance impact will remain small on today’s mainstream desktop and workstations systems, but may be an issue in the future when we move to four or more cores.

5. ACKNOWLEDGMENTS

The authors want to thank Darryl Gove, John Henning, and Jeff Reilly for their helpful reviews and/or constructive suggestions. Part of this work was performed on equipments donated by AMD Inc.

AMD, AMD Athlon 64, AMD Opteron, and AMD HyperTransport are trademarks of Advanced Micro Devices, Inc. SUSE is the registered trademark of Novell, Inc. EKOPath and PathScale are trademarks of QLogic, Co. Linux® is the registered trademark of Linus Torvalds.

6. DISCLAIMER

All performance numbers reported in this paper are estimates because they are not fully compliant with SPEC run and reporting rules [12]. It is expected, though not proven, that results from a fully compliant run would be very close.

7. REFERENCES

[1] Free Software Foundation. GCC, GNU Compiler Collection, <http://gcc.gnu.org>.

[2] Ace’s Hardware. Volume MP Systems, <http://www.aceshardware.com/read.jsp?id=45000338>.

[3] John Hennessy, Daniel Citron, David Patterson, and Guri Sohi. The Use and Abuse of SPEC: An ISCA Panel. *IEEE Micro*, 23(4):73–77, July/August 2003.

[4] John L. Henning. SPEC CPU2006 Benchmark Descriptions. *ACM SIGARCH Computer Architecture News*, 34(4):1–17, September 2006.

[5] Novell. SUSE Linux, <http://www.novell.com/products/suselinux>.

[6] openSUSE. openSUSE, <http://www.opensuse.org>.

[7] QLogic. PathScale EKOPath™ Compiler Suite, <http://www.pathscale.com>.

[8] Jeff Reilly. Invited Talk on “Evolve or Die: Making SPEC’s CPU Suite Relevant Today and Tomorrow”. In *2006 IEEE International Symposium on Workload Characterization (IISWC 2006)*, pages 119–119, October 2006.

[9] Kevin Skadron, Margaret Martonosi, David I. August, Mark D. Hill, David J. Lilja, and Vijay S. Pai. Challenges in Computer Architecture Evaluation. *IEEE Computer*, 36(8):30–36, 2003.

[10] SPEC. SPEC CPU Benchmark Search Program, <http://www.spec.org/cpu2005/search>.

[11] SPEC. SPEC CPU2000 published results, <http://www.spec.org/cpu2000/results>.

[12] SPEC. SPEC CPU2006 Documentation, <http://www.spec.org/cpu2006/Docs>.

[13] SPEC. SPEC CPU2006 published results, <http://www.spec.org/cpu2006/results>.

[14] Dong Ye, Joydeep Ray, Christophe Harle, and David Kaeli. The Performance Characterization of SPEC CPU2006 Integer Benchmarks on x86-64 Architecture. In *2006 IEEE International Symposium on Workload Characterization (IISWC 2006)*, pages 120–127, October 2006.