WPI-CS-TR-98-16 July 1998

Touchstone – A Lightweight Processor Benchmark

by

Mark Claypool

Computer Science
Technical Report
Series

WORCESTER POLYTECHNIC INSTITUTE

Computer Science Department
100 Institute Road, Worcester, Massachusetts 01609-2280

Touchstone – A Lightweight Processor Benchmark

Mark Claypool
claypool@cs.wpi.edu
Worcester Polytechnic Institute
Computer Science Department

August 11, 1998

Abstract

Benchmarks are valuable for comparing processor performance. However, porting and running processor benchmarks on new platforms can be difficult. Touchstone, a simple addition benchmark, is designed to overcome portability problems, while retaining performance measurement accuracy. In this paper, we present experimental results that show Touchstone correlates strongly with processor performance under other benchmarks. Equally important, we present a measure of portability that demonstrates Touchstone is easier to port and run than other benchmarks. We conclude that while Touchstone is not a replacement for more extensive processor benchmarks, it can be used for quick, reasonably accurate estimations of processor performance.

Keywords: benchmark, processor performance, measurement

1 Introduction

> `benchmark` *v. trans* - to subject (a system) to a series of tests in order to obtain prearranged results not available on competitive systems. – *S. Kelly-Bootle, "The Devil's DP Dictionary"*

Standard measures of performance for computers systems provide a basis for comparison, which can lead to system improvements and predictions of effectiveness under other applications. If computer users ran the same programs day in and day out, performance comparisons would be straight-forward. As this will never be the case, systems must rely on other methods to predict performance under estimated workloads. The process of performance comparison for two or more systems by measurements is called *benchmarking*, and the workloads used in the measurements are called *benchmarks*. Broadly, there are four types of benchmarks [10, 11]:

Real programs. Real programs are applications that users will actually run on the system. Observing a system running such programs may be a good indication of performance under normal operations. Examples are language compilers such as `gcc`, `acc` and `f77`, text-processing tools like `TeX` and `Framemaker`, and computer-aided design tools like `Spice`.

Kernels. Kernels are small, key pieces from real programs. Unlike real programs, no user runs kernel programs, for they exist only for performance measurements. They are best used to isolate performance of key system features. Examples are `LINPACK` (see Section 2 for more information) and the `Livermore Loops` [14].

Synthetic benchmarks. Synthetic benchmarks have characteristics similar to those of a set of real programs and can be applied repeatedly in a controlled manner. They require no real-world data files (that may be large and contain sensitive data), and can be easily ported without affecting their basic operation. They often have built-in measurement capabilities. Examples are `Spec` (see Section 2 for more information), `Whetstone` [7] and `Dhrystone` [15].

Toy benchmarks. Toy benchmarks are small pieces of code that produce results known before hand. They are small, easy to implement, and can be run on almost any computer. They may be used in some real programs, but often are not. Examples are the `Sieve of Eratosthenes` and `Quicksort` (see Section 2 for more information).

Although the above benchmarks may provide valuable system performance information, they are often difficult to run properly. Determining the correct mix of real programs is difficult and installing real programs can be time-consuming and resource intensive. `Gcc`, for example, required more disk space than we had available and demanded a signifcant amount of compilation time (see Section 3.2 for more details). Running kernels can be difficult without the necessary compilers and often small changes are required to port kernels between platforms. LINPACK, for example, requires a Fortran compiler and a user-supplied `second()` function, that caused us porting difficulties (see Section 3.2 for more details). Coding toy benchmarks can be surprisingly difficult. Many computer professionals find it difficult to accurately code a `Mergesort` correctly, despite understanding the algorithm. Obtaining reputed synthetic programs can be equally challenging. For instance, although SPEC is non-profit, they charge a support and development fee for use of their benchmarks. Prices in 1995 were [6]:

Benchmark Type	Price
CPU intensive integer benchmarks)	$ 425
CPU intensive floating point benchmarks	$ 575
Integer and floating point benchmarks	$ 900
System level file server (NFS) workload	$ 1200
UNIX software development workloads	$ 1450

Even the basic integer or floating point benchmarks are far too much for the average graduate student wishing to simply explore workstation performance.

To overcome some of the difficulties in running standard benchmarks, we present *Touchstone*, a lightweight benchmark for processor performance. Touchstone is process that increments a variable in a tight loop for a fixed period of time. The value that it reports represents the power of the processor; the higher the number, the more powerful the processor while the lower the number, the less powerful the processor. Touchstone's beauty is its simplicity. At its core, Touchstone uses only 2 lines of high-level code, a branch and an add. These commands are found on all systems and in all languages. Its simplicity makes it easy to port while its basic functionality provides a surprisingly accurate measure of system performance.

Increment instruction benchmarks, such as Touchstone, are not new. Historically, processors were the most expensive and the most used system components. Initially, computers had very few instructions, the most frequent of which was addition. Thus, the computer that added faster was considered a better performer. As the number and complexity of instructions grew, the addition benchmark was no longer sufficient as the only means of measuring processor performance. This remains true today, and so Touchstone should not replace existing benchmarks. However, rather than being abandoned, Touchstone should be used as a quick, easy way to give an approximate estimate of processor performance.

We have two hypotheses about the usefulness of Touchstone:

> *Hypothesis*: Touchstone is a good indicator of processor performance under other benchmarks.
>
> *Hypothesis*: Touchstone is easier to port than other benchmarks.

The rest of this paper is laid out as follows: Section 2 describes experiments we ran to test our hypotheses; Section 3 analyzes our experimental data and tests our hypotheses; and Section 4 summarizes our conclusions and presents other possible uses for Touchstone.

2 Experiments

> `touchstone` *noun* - a test or criterion for determining the quality or genuineness of a thing. – *Webster's 7th dictionary, on-line.*

In order to test our first hypothesis, *Touchstone is a good indicator of processor performance under other benchmarks*, we need to correlate Touchstone with one benchmark from each of the four categories presented in Section 1. This Section describes our experiments to gather data for these correlation computations.

The four benchmarks we selected are: `gcc`, `LINPACK`, `SPEC` (`int` and `fp`) and `quicksort`. To strengthen our results, we perform correlations on nine different computer workstations. The workstations are: SGI Crimson, SGI Indigo 2 Extreme, Sun Sparc 10, Sun IPX, Sun Sparc 2, Intel 80486-33, SGI Personal Iris, Sun Sparc 1 and Intel 80386-20. Table 1 summarizes the systems.

The next 5 subsections describe the experiments involving Touchstone and each benchmark.

2.1 Touchstone

We ran Touchstone on the 9 platforms listed above. Since the count Touchstone reports is sensitive to other processes running concurrently, we postulate that lightly loaded machines a must for accurate measurements. We performed the experiments on machines with a minimum of other processes. In order to facilitate finding when time-shared machines are little used, we developed a network script to monitor processor load. Figure 1 depicts the sample output from one of our script runs. In this case, the script helped us avoid running Touchstone during the 3 a.m. system

Workstation	Operating System	Processor Speed	RAM
SGI Crimson	Irix 5.2	100 Mhz	160 Mbytes
SGI Indigo 2 Extreme	Irix 5.2	100 Mhz	32 Mbytes
Sun Sparc 10	SunOs 4.1.4	33 Mhz	64 Mbytes
SGI Personal Iris	Irix 4.01	20 Mhz	32 Mbytes
Sun IPX	SunOs 4.1.4	40 Mhz	40 Mbytes
Sun Sparc 2	SunOs 4.1.4	40 Mhz	30 Mbytes
Sun Sparc 1	SunOs 4.0.2	20 Mhz	20 Mbytes
Intel 80486	Linux 1.2.13	33 Mhz	8 Mbytes
Intel 80386	Linux 1.2.13	20 Mhz	8 Mbytes

Table 1: Experimental platforms. Nine different systems from three different manufacturers were used for the experiments. The 386, 486 and Sparc 1 machines represent low-end workstations. The Iris, IPX and Sparc 2 represent mid-range workstations. The Crimson, Indigo and Sparc 10 represent high-end workstations.

backups. Verifying that the Unix system load after the Touchstone runs was 1 assured us that no other processes had been running.

We ran Touchstone for 200 seconds to amortize start-up costs. To determine the number 200, we ran Touchstone for increasing times and recorded the standard deviation of its counts. Figure 2 depicts the standard deviation of a series of Touchstone runs versus the length of the run. The standard deviations level out just before 200 seconds. At this point, the standard deviation is only 0.001% of the mean. Thus, we chose one 200 second Touchstone run as one iteration. Pilot tests indicated that five iterations for each machine were needed to achieve reasonable confidence intervals.

Touchstone can increment a floating point variable or an integer variable. We call Touchstone with a floating point variable `TouchFp` and Touchstone with an integer variable `TouchInt`. The Touchstone values we measured are presented in Table 2.

2.2 Quicksort

We implemented a standard quicksort algorithm, as found in most typical computer algorithms books. To ensure that poor choices for the partitioning pivot did not influence sorting times, we sorted the same sequence of randomly generated numbers on each platform. We measured only the sorting time, not the time to read in the numbers, using the wall-clock. The pseudo-code follows:

```
load(array, array_size)
start_time = gettime()
quicksort(array, 1, array_size)
end_time = gettime()
```

Since wall-clock times are sensitive to other running processes, we ran the quicksort experiments on lightly loaded machines, as described in Section 2.1.

The Quicksort times we measured are presented in Table 2.

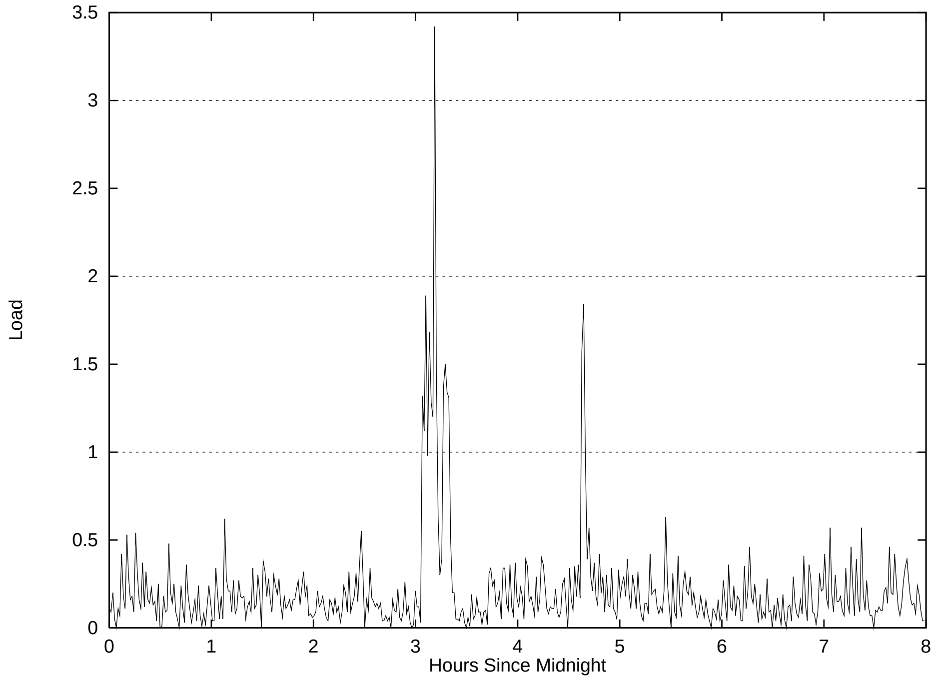

Figure 1: Processor Load at Night. The horizontal axis is hours since midnight. The vertical axis is the number of processes in the ready queue. From this graph, it appears that backups are sometime around 3 a.m. for this system. This time should be avoided for running Touchstone experiments.

2.3 GCC

The Gnu C compiler (`gcc`) is a product of the Free Software Foundation. For reasons not related to its price, it is often preferred by many for system development. The latest `gcc` source code is available from any of the gnu ftp sites, such as `anonymous@prep.ai.mit.edu:pub/gnu/gcc-*.tar.gz` and `anonymous@ftp.uu.net:packages/gnu/gcc-*.tar.gz`.

We ran experiments on lightly loaded machines as we described in Section 2.1. The input for `gcc` was the `LaTeX` source. The `LaTeX` document preparation system is a special version of Donald Knuth's TeX program. `LaTeX` adds to `TeX` a collection of commands that simplify typesetting by letting the user concentrate on the structure of the text rather than on formatting commands [12]. `LaTeX` is available for most workstation platforms. Compilation time was measured using the Unix `time` command.

The gcc times we measured are presented in Table 2.

2.4 LINPACK

LINPACK was developed by Jack Dongarra of Argonne National Laboratory in 1983 [9]. It consists of a number of programs that solve dense systems of linear equations. LINPACK attempts to represent mechanical engineering applications on workstations. LINPACK programs can be characterized as having a high percentage of floating-point additions and multiplications. The benchmark

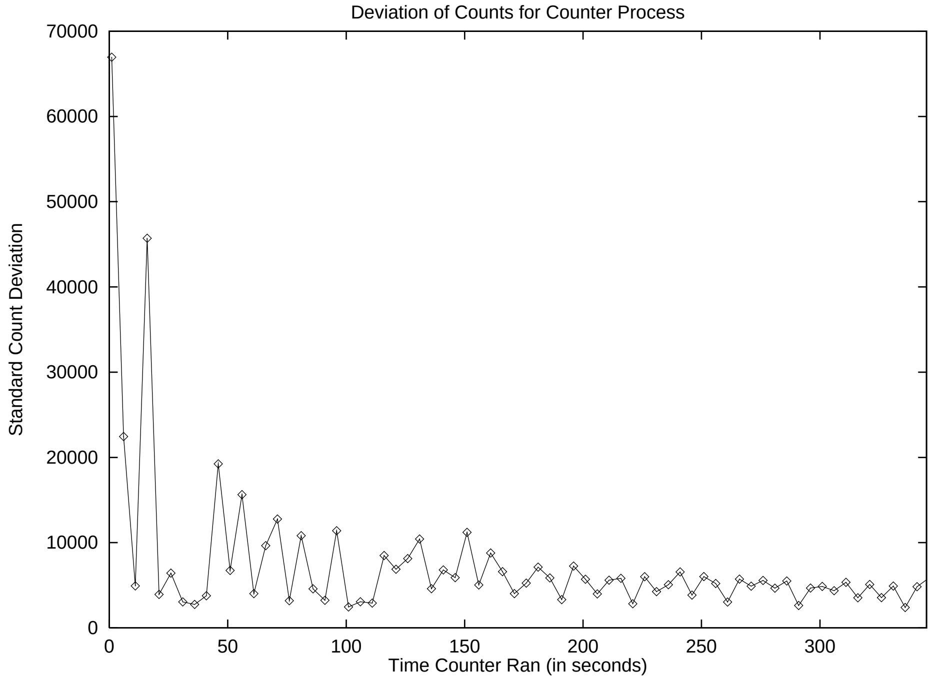

Figure 2: Standard Deviation of Touchstone Values. The horizontal axis is the number of seconds that Touchstone ran. The vertical axis is the standard deviation of the runs. The standard deviation appears to reach a steady state just before 200 seconds. At this point, it is 0.001% of the mean.

results are reported in Millions of Floating Point Operations Per Second (MFLOPS) [11].

The LINPACK benchmark consists of three parts:

1. *LINPACK Benchmark.* The standard LINPACK benchmark is written entirely in Fortran and must be run with no changes to the source code. The problem size is 100x100. It uses an old algorithm that has low compute intensity and is not optimized in terms of memory bandwidth.
2. *Towards Peak Performance, Best Effort.* The best effort benchmark seeks to achieve the maximum performance on the workstation by allowing modifications to the benchmark code. The problem size is 1000x1000. The best implementations currently use blocked solvers that make efficient use of memory with high compute intensity. However, there are limits on algorithm selection and use of assembly language to improve performance. The LINPACK documentation provides examples of this type of solution.
3. *A Look at Parallel Processing.* The parallel benchmark uses a problem size of `NxN` with `N` selected by the user. The best implementations use high compute intensity algorithms scaled to a size where interprocessor communication cost is minimal compared to the computation cost.

The LINPACK benchmark results as reported in [9] are presented in Table 2.

Machine	TouchInt	TouchFp	Mflops	SPECint	SPECfp	Quicksort	Gcc
Crimson	9008886	5479664	16	58.3	63.4	0.84	-
Indigo	8802542	5367132	15.0	57.6	60.3	0.84	50.5
Sparc10	7044849	3445895	8.9	40.0	41.1	1.87	209.1
IPX	3506958	1702699	4.1	21.8	21.5	3.78	279.8
Sparc2	3580454	1709879	4.0	21.8	18.2	4.48	278.3
486	4096334	1230640	3.0	17.5	15.5	5.17	695.6
Iris	3721732	1331571	2.1	22.4	24.4	3.51	199.8
Sparc1	1737108	663700	1.4	9.5	7.5	9.31	530.7
386	638547	1730[1]	0.16	2.15	2.15	23.89	1590.1

Table 2: Experiment results. TouchInt and TouchFp are 200 second Touchstone values we measured when the increment variable is an integer variable or a floating point variable, respectively. Mflops are Millions of Floating Point Operations Per Second, as recorded by LINPACK. SPECint and SPECfp are integer and floating point intensive benchmarks recorded by the SPEC benchmark suite. Quicksort and Gcc are the run times in seconds for our Quicksort and Gcc experiments.

2.5 SPEC

The Systems Performance Evaluation Cooperative (SPEC) is a nonprofit corporation formed by leading computer vendors to develop a standardized set of benchmarks. Founders, including Apollo/Hewlett-Packard, DEC, MIPS and Sun, have agreed on a set of real programs and inputs that all will run. The founders of this organization believe that the user community will benefit greatly from an objective series of application-oriented tests, which can serve as common reference points and be considered during the evaluation process. The benchmarks are meant for comparing processor speeds. The spec numbers are the ratio of the time to run the benchmarks on a reference system and the system under test [6].

There are currently two suites of compute-intensive SPEC benchmarks, measuring the performance of processor, memory system, and compiler code generation. `SPECint` and `SPECfp` represent results from integer intensive and floating point intensive benchmarks, respectively. They normally use UNIX as the portability vehicle, but they have been ported to other operating systems as well. The percentage of time spent in operating system and I/O functions is generally negligible.

The SPEC benchmark results as reported in [1] are presented in Table 2.

3 Analysis

In this section we analyze our experimental data from Section 2 and test the hypotheses presented in Section 1.

3.1 Touchstone as an Indicator of Processor Performance

To test our first hypothesis, *Touchstone is a good indicator of processor performance under other benchmarks*, we determine the correlation between Touchstone and each benchmark. `LINPACK`

Benchmark	A	Confidence Interval	B	Confidence Interval	R^2
LINPACK	2.94e-06	[2.74e-06, 3.15e-06]	-7.75e-01	[-1.39e+00, -1.55e-01]	0.99
SPECint	6.64e-06	[5.93e-06, 7.35e-06]	-3.21e+00	[-7.07e+00, 6.56e-01]	0.98
SPECfp	1.10e-05	[9.91e-06, 1.20e-05]	2.73e+00	[-3.99e-01, 5.86e+00]	0.98
Quicksort	1.41e-07	[1.06e-07, 1.76e-07]	-2.12e-01	[-4.02e-01, -2.22e-02]	0.89
Gcc	1.89e-09	[8.56e-10, 2.93e-09]	-2.76e-03	[-7.77e-03, 2.25e-03]	0.67

Table 3: Touchstone linear regression parameters. Parameters `A` and `B` are from the equation `y = Ax + B`. Intervals are 95% confidence intervals around the parameters `A` and `B`.

and `SPECfp` both primarily test floating point performance, so we compare them with `TouchFp` (Touchstone incrementing a floating point variable). Similarly, `SPECint` stresses integer operations, so we compare it with `TouchInt` (Touchstone incrementing a long integer variable). Since our `quicksort` benchmark sorted integers, we compare it with `TouchInt`. `Gcc` contains 99% integer operations [10]. Thus, we correlated `gcc` with `TouchInt`. For benchmarks measured by time, such as `gcc` and `quicksort`, the lower the number the more powerful the processor. Thus, Touchstone values are inversely proportional to the time for `gcc` and `quicksort`.

To quantify our comparisons, we apply a simple linear regression model through a least-squares line fit for each benchmark. We calculate the regression parameters `A` and `B` from the equation: `y = Ax + B`. This gives us a line predicting the correlation between Touchstone and the other benchmarks. In addition, we calculate a 95% confidence interval surrounding each parameter. This gives us a range for performance predictions for different processors having different Touchstone counts. The numeric results are given in Table 3. Figures 3 through 7 depict the results graphically.

Without linear regression, the mean of Touchstone and the other benchmarks can be used for the predicted correlation. The quantity is called the *total sum of squares*. It can be broken into two parts: the variation not explained by the regression and the variation explained by the regression. The fraction of the variation that is explained by the regression is called the *coefficient of determination*, R^2. The "goodness" of a regression is measured by R^2. The higher the value of R^2, the better the regression. If the regression is perfect in the sense that all observed values are equal to those predicted by the model, R^2 will be one. On the other hand, if the regression model is completely bad, R^2 will be zero. R^2 values above .8 are said to have a strong correlation, R^2 values above between .5 and .8 are considered having a medium correlation, and R^2 values below .5 have a weak correlation [8]. Table 3 lists the coefficient of determination for Touchstone and each benchmark and Figure 8 gives a graphical representation of the coefficient of determination for each benchmark.

Touchstone correlates strongly with all the benchmarks with the exception of `gcc`. Most likely this is because Touchstone is a processor intensive benchmark while `gcc` reflects the performance of more of the system, including Mbytes of RAM and disk speed.

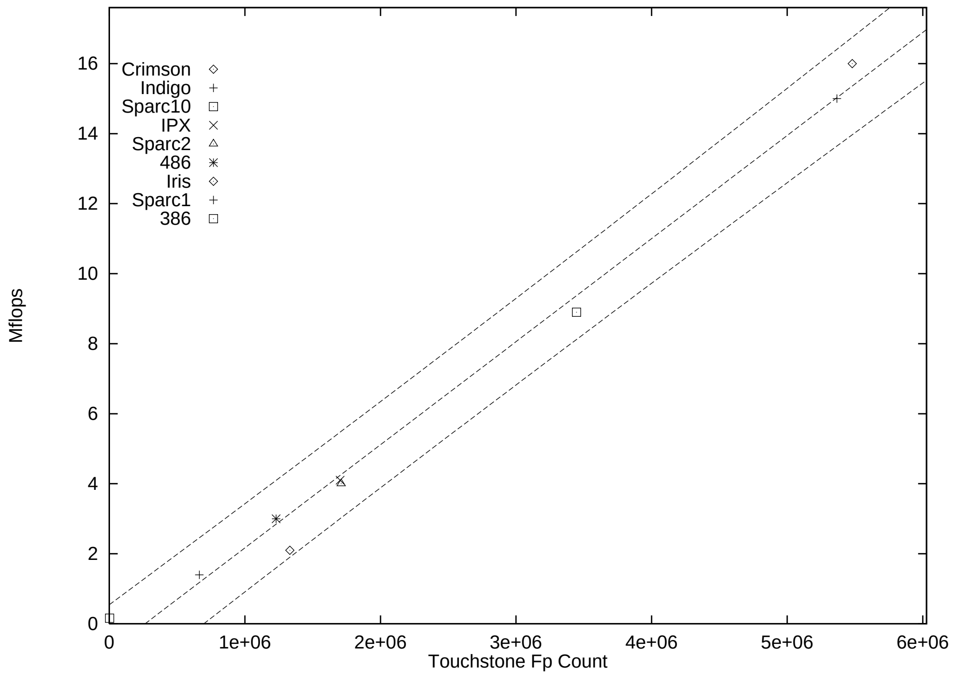

Figure 3: LINPACK Mflops vs. TouchFp. The horizontal axis is the Touchstone count. The vertical axis is the Mflops score. The middle line is the least-squares line fit. The upper and lower curving lines represent the 95% confidence interval around the least-squares line fit.

3.2 Touchstone Portability

To test our second hypothesis, *Touchstone is easier to port than other benchmarks*, we create a portability function that takes into account a level of effort to run each benchmark. Our level of effort is based on the number of user steps required to port the benchmark, the amount of time it took to compile, and the disk space required. We weight each component roughly based on a dollar cost:

- *Change* is the number of line changes necessary to either the source code or the `Makefile` in order to ready the benchmark for a new platform. Change also includes the number of steps required to do an installation and run. We assume each change takes 5 human minutes. We weight human time at $50/hour, so each change costs about $12.
- *Compilation* is the number of minutes to compile the benchmark. We weight machine time at $1 per hour, so each minute of compilation costs about $.16.
- *Space* is the number of Mbytes required to support the benchmark, including source code, compiled code and output data. We weight space at $1 per Mbyte.

The total measure of portability is the weighted sum of each of the above components for each of the 5 operating systems listed in Table 1 (Irix 5.2, SunOs 4.1.4, Irix 4.01, SunOs 4.0.2 and Linux 1.2.13):

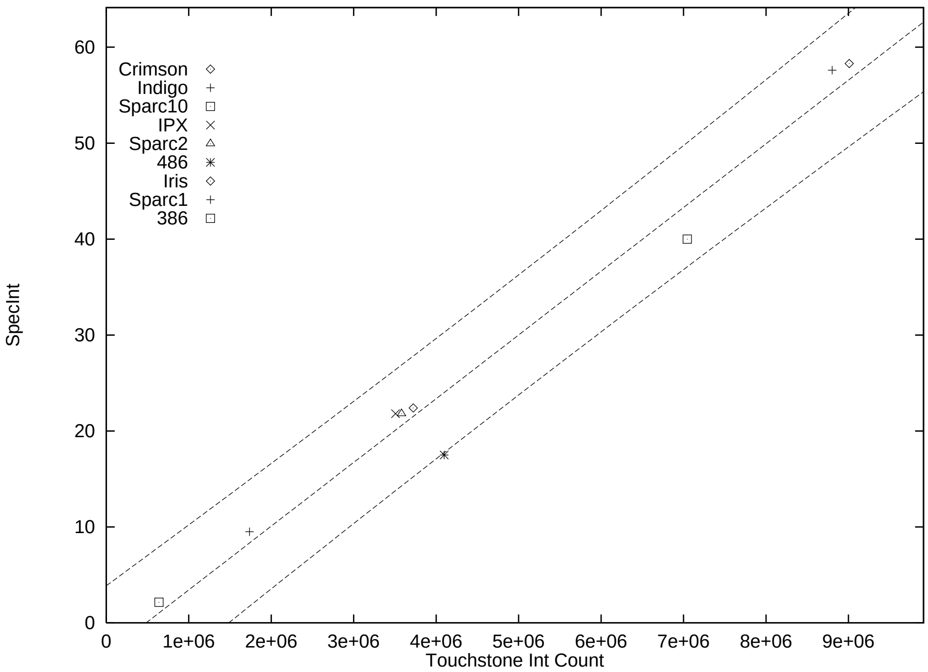

Figure 4: SPECInt vs. TouchInt. The horizontal axis is the Touchstone count. The vertical axis is the SPECint score. The middle line is the least-squares line fit. The upper and lower curving lines represent the 95% confidence interval around the least-squares line fit.

$$Portability = Change \times 12 + Compilation \times .16 + Space \times 1$$

The SPEC benchmarks were unavailable to us because of their prohibitive costs. Their portability was the dollar cost for the integer and floating point benchmarks (see Section 1).

Table 4 summarizes the portability for each benchmark and Figure 9 shows the total portability graphically.

The installation of `gcc` was very resource intensive. The distribution alone required over 20 Mbytes of disk space. Several of the systems we ran our experiments on had disk quotas in effect. On these

Benchmark	Change	Compilation	Space	Portability
Touchstone	5	10	.2	62
Quicksort	5	20	1.5	65
LINPACK	21	30	2.8	260
Gcc	18	400	30	310
SPEC	-	-	-	900

Table 4: A Measure of Portability. "Change" is the number of line changes necessary to ready the benchmark for a new platform. "Compilation" is the number of minutes to compile the benchmark. "Space" is the number of Mbytes required to support the benchmark. "Portability" is a weighted sum of Change, Compilation and Space.

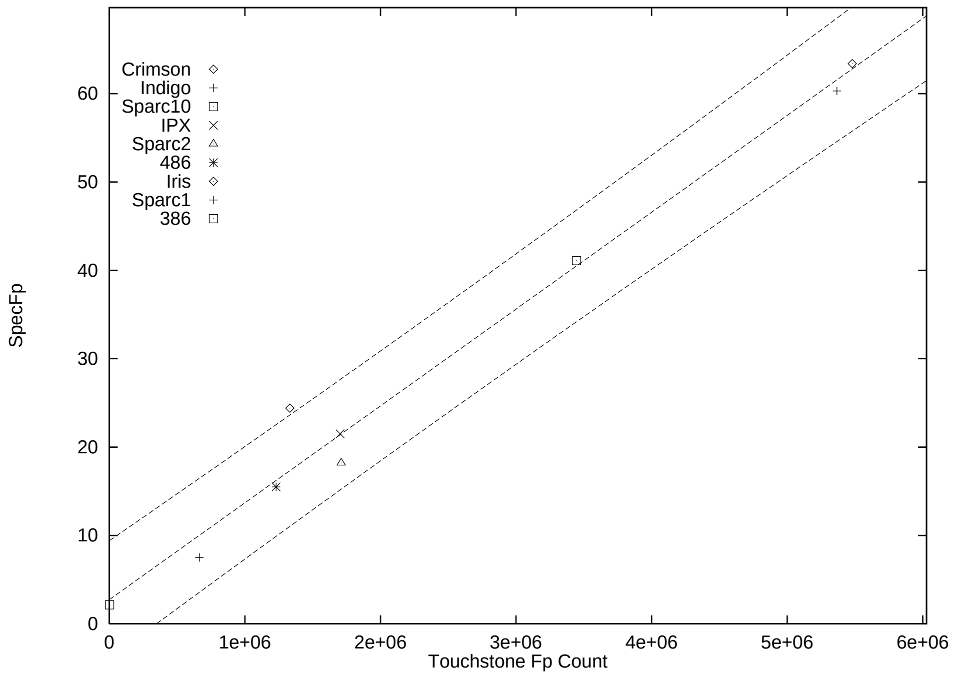

Figure 5: SPECfp vs. TouchFp. The horizontal axis is the Touchstone count. The vertical axis is the SPECfp score. The middle line is the least-squares line fit. The upper and lower curving lines represent the 95% confidence interval around the least-squares line fit.

systems, we had to receive assistance from the system administrators in running our benchmarks.

From the above data, we find Touchstone is the most portable of the benchmarks tested.

4 Conclusion

Benchmarks provide a standard measure of performance that demonstrate system improvements and allow users to make informed system choices. Unfortunately, when benchmark results are unavailable for a system, obtaining and running typical benchmarks can be difficult or impossible.

Touchstone can overcome some of the difficulties in running standard benchmarks. Touchstone is a process that increments a variable in a tight loop for a fixed period of time. This simplicity makes Touchstone easier to port than other benchmarks. The count value that Touchstone reports represents the power of the processor. Surprisingly, this simple measure of performance has a strong correlation with processor performance under other benchmarks. Thus, Touchstone can be used as an quick, accurate easy-to-use indicator of processor performance. Moreover, you can convert Touchstone results into common benchmark results by using the linear equations in Table 3.

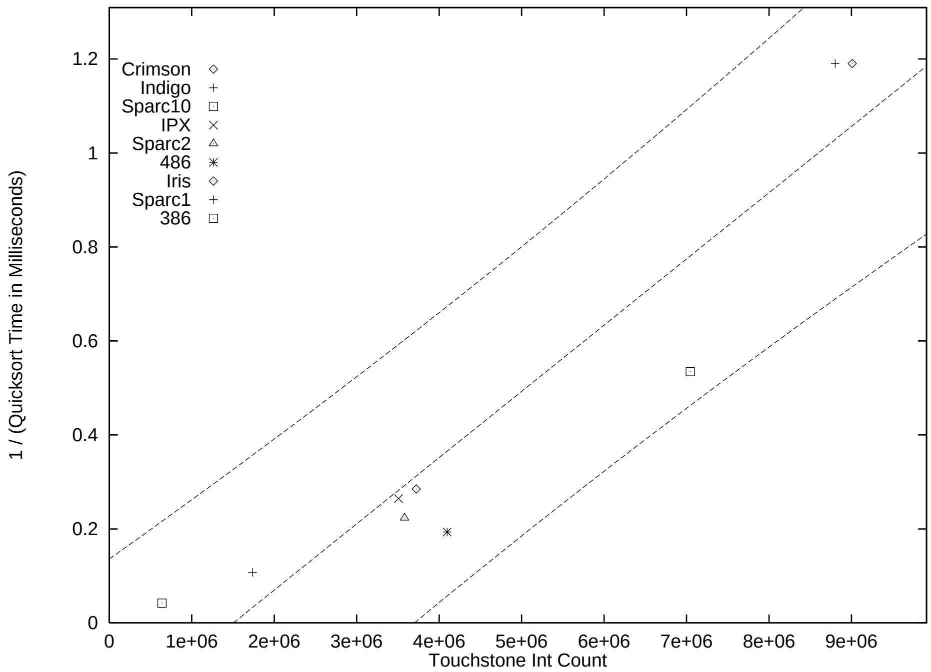

Figure 6: Quicksort vs. TouchInt. The horizontal axis is the Touchstone count. The vertical axis is the 1 over the quicksort score. The middle line is the least-squares line fit. The upper and lower curving lines represent the 95% confidence interval around the least-squares line fit.

4.1 A Grain of Salt

All of the Touchstone measurements should be taken with a grain of salt, for they are largely dependent upon the compiler used. For example, compiling Touchstone on the Sun IPX with `gcc` resulted in a count 1/3 lower than the Touchstone count reported above using `acc`. In addition, compiler optimizations, such as loop unrolling and instruction reordering to exploit instruction slots available after delayed branch instructions, may increase Touchstone counts even more. According to John Larson and David Bailey, such manipulative benchmarking becomes "benchmarketing" when the normally objective evaluation process becomes flawed incomplete, irrelevant, or invalid [13, 2]. Therefore, this weakness in Touchstone is to be expected. Standard benchmarks such as `LINPACK` and `SPEC` always list the compiler names and flags that the benchmark was tested under when reporting benchmark performance. Ultimately, it is up to the user to apply Touchstone properly if a reasonably accurate indicator of processor performance is to be expected.

4.2 Other Applications of Touchstone

It is easy to extend Touchstone to study the effects that other peripherals and other processes on processor performance. This use of Touchstone is depicted in Figure 10. The Touchstone count on the bare machine gives us the processor potential for the machine, depicted by the single column on the far left. We then run Touchstone with the other component we wish to test. The difference in the bare Touchstone count and the component Touchstone count is the component-induced load.

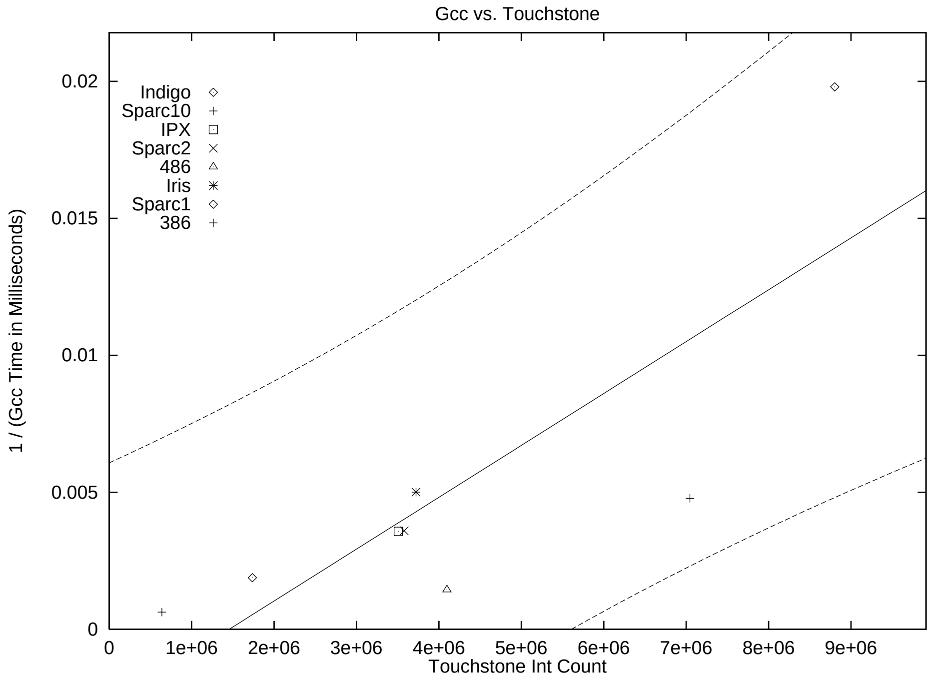

Figure 7: Gcc vs. TouchInt. The horizontal axis is the Touchstone count. The vertical axis is the 1 over the gcc score. The middle line is the least-squares line fit. The upper and lower curving lines represent the 95% confidence interval around the least-squares line fit.

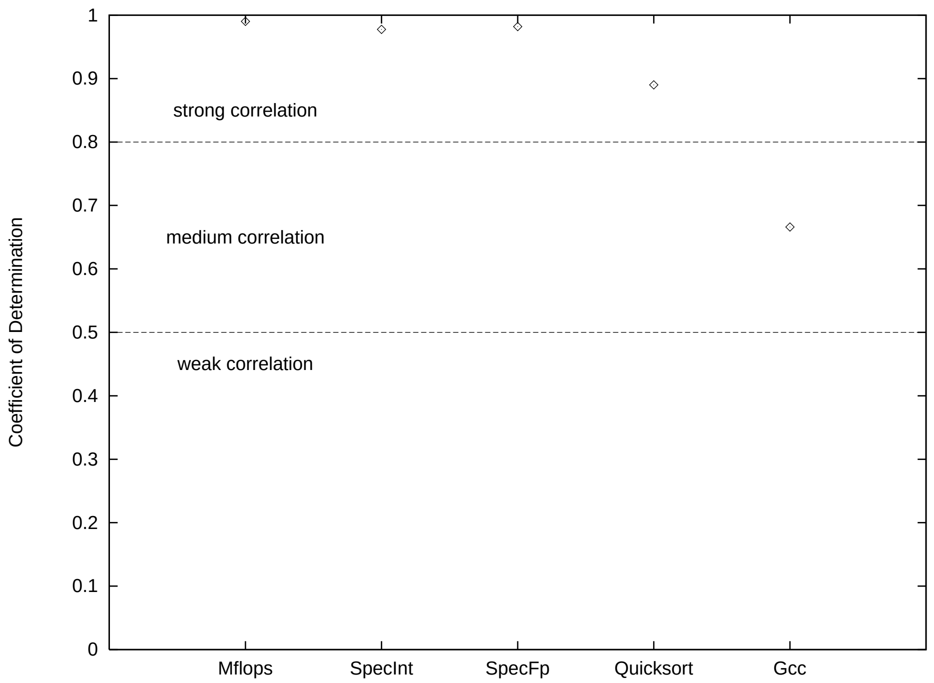

Figure 8: Correlation with Touchstone. The horizontal axis is the coefficient of determination. The vertical axis is the benchmark. The horizontal lines represent the weak, moderate and strong correlation boundaries as indicated.

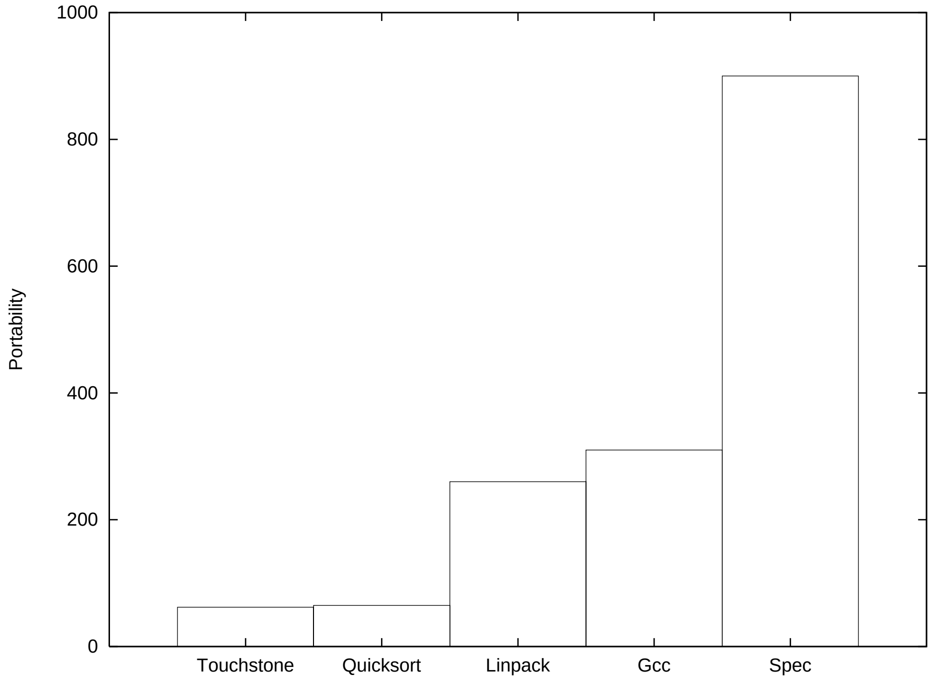

Figure 9: Total Portability. The weighted sum of Change, Compilation and Space is shown for each benchmark.

For example, the middle two columns show the count obtained if two Touchstone processes were run simultaneously. On a Sun IPX, each Touchstone would count to about 0.85 million per second. In the last two columns, the processor load of the `send` process would be the count obtained on a bare machine less the count obtained by the Touchstone process. If the Sun IPX were sending data at a rate of a 1280 byte packet every 160 ms, the count would be about 1.69 million in one second. The difference between the 1.7 million barecount and the 1.69 million send count can be converted into milliseconds of processor load, resulting in a send load of about 1 millisecond per second. As another example, a Touchstone count on a system with a SCSI disk having a faulty parity bit can be compared to a Touchstone run on the same system with a healthy SCSI disk.[2] The difference in the values reported is the processor degradation due to the failing disk.

To verify that Touchstone does indeed accurately report load of processor bound processes with which it runs concurrently, we ran Touchstone with 1, 2, 3 and 4 other Touchstones. We expect the count value to be `(count on bare machine) / (N+1)`, where N is the number of other counters running. Figure 11 depicts the result of this experiment. At all points, the predicted values were within the confidence intervals for the measured values. In fact, we have used Touchstone successfully extensively to measure the processor load of many low-level systems [4, 3, 5]. Note that in instances where processor degradation was very slight, single-user mode was necessary to determine exact performance.

In time sharing systems, Touchstone can also be used to analyze computing power. Touchstone

[2] In fact, we used Touchstone for just this purpose. The ailing disk caused a performance degradation of 20% because of the many interrupts it generated.

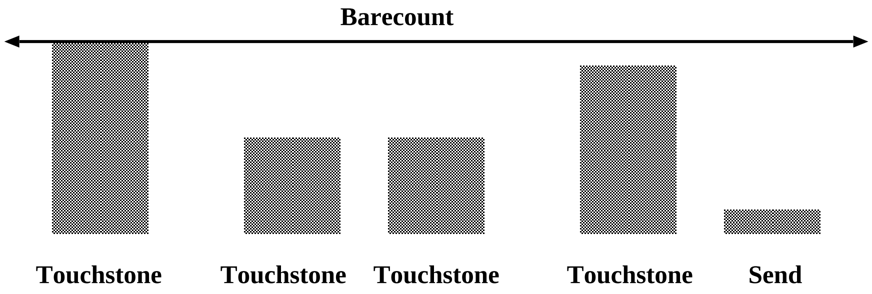

Figure 10: Touchstone to Measure Process Load. Touchstone is run on a quiet machine to get the "barecount" as depicted on the left. Touchstone is then run with the component to be tested, such as another Touchstone (as depicted in the middle) or a process that sends packets (as depicted on the right). The processor load of the component is the barecount less the count obtained by the Touchstone process.

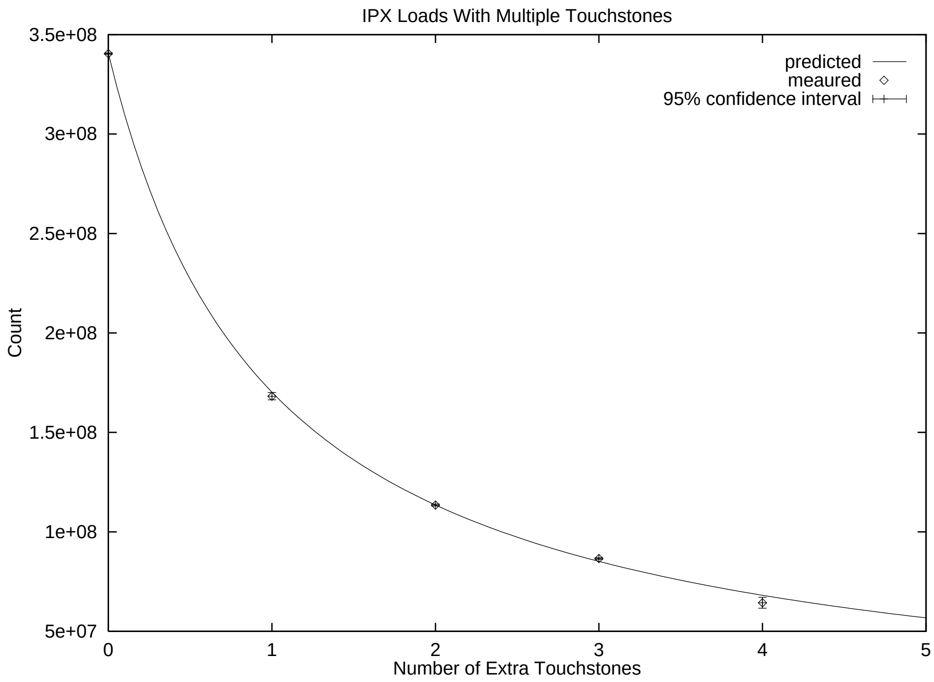

Figure 11: Sun IPX loads with multiple Touchstone processes. The points are from 5 samples each with 95% confidence intervals. The largest interval is 8% of the measured value. The curving line is the predicted value. At all points, the predicted values were within the confidence intervals for the measured values.

can be run several times during the day. The mean value reported indicates the mean available processor power. In this sense, the Touchstone value on a lightly loaded machine will give the peak processor power. To determine processor utilization, Touchstone can be run as a low-priority process, through `nice` or some other means.[3] The value reported represents the amount of "unused" processor and can be used to estimate the number of additional users a time-sharing system may support before the processor becomes the critical resource.

4.3 Future Work

There are several possible areas for future work:

- *Other Workstations.* We ran Touchstone on most of the workstation platforms that were available to us. Future work might involve seeking assistance from other researchers, perhaps even the Internet community at large, in running Touchstone on other workstations.
- *Other Benchmarks.* We correlated Touchstone with one benchmark from each of the benchmark categories presented in Section 1. Future work might involve correlating Touchstone to additional benchmarks within those categories.
- *Multi-Processors.* Touchstone may not give a good indication of performance for machines that have more than one processor. For although a counting loop can certainly be pipelined, splitting the work to multiple processors may be difficult since counting is inherently sequential. Future work may involve research into a counter that utilizes multiple processors.

References

[1] *The Performance Database Server.* 1997. The URL for this document can be found at http://netlib2.cs.utk.edu/performance/html/PDStop.html.

[2] David H. Bailey. How to fool the masses when giving performance results on parallel computers. *Supercomputer*, 8(5):4 – 7, September 1991.

[3] M. Claypool, J. Riedl, J. Carlis, G. Wilcox, R. Elde, E. Retzel, A. Georgopoulos, J. Pardo, K. Ugurbil, B. Miller, and C. Honda. Network requirements for 3D flying in a zoomable brain database. *IEEE JSAC Special Issue on Gigabit Networking*, 13(5), June 1995.

[4] Mark Claypool and John Riedl. Silence is golden? The effects of silence deletion on the CPU load of an audio conference. In *Proceedings of IEEE Multimedia*, Boston, May 1994.

[5] Mark Claypool and John Riedl. A quality planning model for distributed multimedia in the virtual cockpit. In *Proceedings of ACM Multimedia*, pages 253 – 264, November 1996.

[6] Standard Performance Evaluation Corporation. SPEC faq. December 1995. The SPEC Primer is frequently posted to the newsgroup comp.benchmarks. SPEC information can also be found at http://www.specbench.org/.

[7] H.J. Curnow and B.A Wichmann. A synthetic benchmark. *The Computer Journal*, 19(1), 1976.

[3] Pilot tests suggest that a maximally `niced` Touchstone running on a machine with a processor bound process will only count 4% as high as an unloaded Touchstone.

[8] Jay Devore and Roxy Peck. *Statistics – The Exploration and Analysis of Data*. Wadsworth, Inc., second edition edition, 1993.

[9] Jack J. Dongarra. Performance of various computers using standard linear equations software. Technical Report CS-89-85, University of Tennessee, February 1994. To obtain a postscript copy of the report send mail to netlib@ornl.gov and in the message type: send performance from benchmark.

[10] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, Inc., 1990.

[11] Raj Jain. *The Art of Computer Systems Performance Analysis*. John Wiley and Sons, Inc., 1991.

[12] Leslie Lamport. *The Latex Document Preparation System/User's Guide and Reference Manual*. Addison-Wesley Publishing Company, 1985.

[13] John L. Larson. Benchmarks. *Newsletter of IEEE Computer Society Technical Comittee on Supercomputer Applications*, 6(1):3 – 4, June 1992.

[14] F.M McMahon. The livermore fortran kernels: A computer test of numerical performance range. Technical Report UCRL-55745, Lawrence Livermore National Laboratory, University of California, December 1986.

[15] R.P. Weicker. Dhrystone: A synthetic systems programming benchmark. *Communications of the ACM*, 27(10):1013 – 1030, October 1984.