OCTOBER 1989

WRL
Technical Note TN-11

Why Aren't Operating Systems Getting Faster As Fast As Hardware?

John Ousterhout

d|i|g|i|t|a|l **Western Research Laboratory** 100 Hamilton Avenue Palo Alto, California 94301 USA

The Western Research Laboratory (WRL) is a computer systems research group that was founded by Digital Equipment Corporation in 1982. Our focus is computer science research relevant to the design and application of high performance scientific computers. We test our ideas by designing, building, and using real systems. The systems we build are research prototypes; they are not intended to become products.

There is a second research laboratory located in Palo Alto, the Systems Research Center (SRC). Other Digital research groups are located in Paris (PRL) and in Cambridge, Massachusetts (CRL).

Our research is directed towards mainstream high-performance computer systems. Our prototypes are intended to foreshadow the future computing environments used by many Digital customers. The long-term goal of WRL is to aid and accelerate the development of high-performance uni- and multi-processors. The research projects within WRL will address various aspects of high-performance computing.

We believe that significant advances in computer systems do not come from any single technological advance. Technologies, both hardware and software, do not all advance at the same pace. System design is the art of composing systems which use each level of technology in an appropriate balance. A major advance in overall system performance will require reexamination of all aspects of the system.

We do work in the design, fabrication and packaging of hardware; language processing and scaling issues in system software design; and the exploration of new applications areas that are opening up with the advent of higher performance systems. Researchers at WRL cooperate closely and move freely among the various levels of system design. This allows us to explore a wide range of tradeoffs to meet system goals.

We publish the results of our work in a variety of journals, conferences, research reports, and technical notes. This document is a technical note. We use this form for rapid distribution of technical material. Usually this represents research in progress. Research reports are normally accounts of completed research and may include material from earlier technical notes.

Research reports and technical notes may be ordered from us. You may mail your order to:

Technical Report Distribution
DEC Western Research Laboratory, UCO-4
100 Hamilton Avenue
Palo Alto, California 94301 USA

Reports and notes may also be ordered by electronic mail. Use one of the following addresses:

Digital E-net:	`DECWRL::WRL-TECHREPORTS`
DARPA Internet:	`WRL-Techreports@decwrl.dec.com`
CSnet:	`WRL-Techreports@decwrl.dec.com`
UUCP:	`decwrl!wrl-techreports`

To obtain more details on ordering by electronic mail, send a message to one of these addresses with the word ‘‘`help`’’ in the Subject line; you will receive detailed instructions.

Why Aren't Operating Systems Getting Faster As Fast As Hardware?

John Ousterhout

October, 1989

Abstract

This note evaluates several hardware platforms and operating systems using a set of benchmarks that test memory bandwidth and various operating system features such as kernel entry/exit and file systems. The overall conclusion is that operating system performance does not seem to be improving at the same rate as the base speed of the underlying hardware.

Western Research Laboratory 100 Hamilton Avenue Palo Alto, California 94301 USA

1. Introduction

This technical note contains results from several benchmark programs that I ran recently on machines at DEC's Western Research Laboratory and at the University of California at Berkeley. The benchmarks are mostly ''micro-benchmarks'', meaning that each one measures a particular hardware or operating system feature (such as memory-to-memory copy speed or kernel entry-exit). Micro-benchmarks like these can identify particular strengths and weaknesses of systems, but they may not give a good indication of overall system performance. The benchmarks also include one ''macro-benchmark'', which exercises a variety of file system features and gives a better idea of overall operating system speed.

My goal in running the benchmarks was partly to compare different hardware platforms and operating systems, but mostly to understand whether or not operating system performance will scale with the base performance of hardware platforms. The answer for today's RISC workstations and RISC mainframes (including machines from DEC, Sun, and MIPS Computer) seems to be ''no'': when comparing slower older machines to newer RISC machines, the speedup in most of the benchmarks is much less than the difference in raw hardware speed.

The benchmarks suggest at least two possible factors that contribute to non-scalability of operating systems. The first is memory bandwidth, which has not scaled to match processor speed in faster machines. The second factor is file systems, some of which require synchronous disk I/Os in common situations. The synchronous I/O requirements limit the performance of operating systems when processors get faster but disks don't.

2. Hardware

I used five hardware platforms for the benchmarks, which are listed in Table 1. Table 1 also includes an abbreviation for each platform, which is used in the rest of this note, an approximate MIPS rating, and an indication of whether the machine is based on a RISC processor or a CISC processor. The MIPS ratings are my own estimates, and are intended to give a rough idea of the base integer performance provided by each platform. The main use of the MIPS ratings is to establish an expectation level for benchmarks. For example, if operating system performance scales with base system performance, then a DS3100 should run the various benchmarks about 1.5 times as fast as a Sun4 and about seven times as fast as a Sun3.

Hardware	Abbreviation	RISC/CISC	MIPS
MIPS M2000	M2000	RISC	20
DECstation 3100	DS3100	RISC	13
Sun-4/280	Sun4	RISC	9
VAX 8800	8800	CISC	6
Sun-3/75	Sun3	CISC	1.8
Microvax II	MVAX2	CISC	0.9

Table 1: Hardware Platforms

All of the machines were generously endowed with memory. As far as I know, no significant paging occurred in any of the benchmarks. In the file-related benchmarks, the relevant files all fit in the main-memory buffer caches maintained by the operating systems.

3. Operating Systems

I used four operating systems for the benchmarks: Ultrix, SunOS, RISC/os, and Sprite. Ultrix and SunOS are the DEC and Sun derivatives of Berkeley's 4.2 BSD UNIX, and are similar in many respects. RISC/os is MIPS Computer Systems' operating system for the M2000 machine. It appears to be a derivative of System V with some BSD features added. Sprite is an experimental operating system developed by my research group at U.C. Berkeley [3]; although it provides the same user interface as BSD UNIX, the kernel implementation is completely different. In particular, Sprite's file system is radically different from that of Ultrix and SunOS, both in the ways it handles the network and in the ways it handles disks. Some of the differences will be seen in the benchmark results.

The version of SunOS used for Sun4 measurements was 4.0, whereas version 3.5 was used for Sun3 measurements. SunOS 4.0 incorporates a major restructuring of the virtual memory system and file system; for example, it maps files into the virtual address space rather than keeping them in a separate buffer cache. This difference will also be reflected in some of the benchmark results.

4. Kernel Entry-Exit

The first benchmark measures the cost of entering and leaving the operating system kernel. It does this by repeatedly invoking the *getpid* kernel call. *Getpid* does nothing but return the caller's process identifier. Table 2 shows the average time for this call on different platforms and operating systems.

Configuration	Time (microseconds)	MIPS-Relative Speed
M2000 RISC/os 4.0	18	0.54
DS3100 Sprite	26	0.49
DS3100 Ultrix 3.1	25	0.60
8800 Ultrix 3.0	28	1.15
Sun4 SunOS 4.0	32	0.68
Sun4 Sprite	32	0.58
Sun3 Sprite	92	1.0
Sun3 SunOS 3.5	108	1.0
MVAX2 Ultrix 3.0	207	0.9

Table 2: Getpid kernel call time

The third column in the table is labeled ‘‘MIPS-Relative Speed’’. This column indicates how well the machine performed on the benchmark, relative to its MIPS-rating in Table 1 and to the Sun3 times in Table 2. Each entry in the third column was computed by taking the ratio of the Sun3 time to the particular machine’s time, and dividing that by the ratio of the machine’s MIP rating to the Sun3’s MIP rating. For the UNIX-derivative operating systems (Ultrix, SunOS, and RISC/os) I used the Sun3 SunOS time; for Sprite I used the Sun3 Sprite time. For example, the MIPS-relative speed for the M2000 is (108/18)/(20/1.8) = 0.54. A MIPS-relative speed of 1.0 means that the given machine ran the benchmark at just the speed that would be expected based on the Sun3 times and the MIPS ratings from Table 1. A MIPS-relative speed less than one means that the machine ran this benchmark more slowly than would be expected from its MIPS rating, and a figure larger than 1 means the machine performed better than might be expected.

For the RISC machines, the MIPS-relative speeds in Table 2 are only about .5-.7. This indicates that the cost for entering and exiting the kernel has not improved as much in the RISC machines as their basic computation speed.

5. Context Switching

The second benchmark is called *cswitch*. It measures the cost of context switching, plus the time for processing small pipe reads and writes. The benchmark operates by forking a child process and then passing one byte back and forth between parent and child using pipes. Table 3 lists the time for each round-trip between the processes, which includes two context switches and one *read* and one *write* kernel call in each process. As with the getpid benchmark, MIPS-relative speeds were computed by scaling from the Sun3 times and the MIPS ratings in Table 1. Once again, the RISC machines didn’t perform as well as might be expected, except for the DS3100/Ultrix combination.

Configuration	Time (ms)	MIPS-Relative Speed
M2000 RISC/os 4.0	0.30	0.71
DS3100 Ultrix 3.1	0.34	0.96
DS3100 Sprite	0.51	0.65
8800 Ultrix 3.0	0.70	1.0
Sun4 SunOS 4.0	1.02	0.47
Sun4 Sprite	1.17	0.41
Sun3 SunOS 3.5	2.36	1.0
Sun3 Sprite	2.41	1.0
MVAX2 Ultrix 3.0	3.66	1.3

Table 3: Cswitch: echo one byte between processes using pipes.

6. Select

The third benchmark exercises the *select* kernel call. It creates a number of pipes, places data in some of those pipes, and then repeatedly calls *select* to determine how many of the pipes are readable. A zero timeout is used in each *select* call so that the kernel call never waits. Table 4 shows how long each *select* call took, in microseconds, for three configurations. The first configuration used a single pipe that contained no data. The second configuration used 10 pipes, all empty, and the third configuration used 10 pipes all containing data. The last column is MIPS-relative speed again, computed using the ‘‘10 full’’ data. The performance of this benchmark was generally in line with the machines’ MIPS ratings.

The M2000 numbers in Table 4 were surprisingly high for pipes that were empty, but quite low as long as at least one of the pipes contain data. I suspect that RISC/os’s emulation of the *select* kernel call is faulty and is causing the process to wait for 10 ms even if the calling program requested immediate timeout.

Configuration	1 pipe (microseconds)	10 empty (microseconds)	10 full (microseconds)	MIPS-Relative Speed
M2000 RISC/os 4.0	10000	10000	108	0.84
DS3100 Sprite	76	240	226	1.04
DS3100 Ultrix 3.1	81	153	151	0.93
Sun4 SunOS 4.0	104	240	216	0.93
8800 Ultrix 3.0	120	265	310	.98
Sun4 Sprite	126	396	356	0.96
Sun3 Sprite	413	1840	1700	1.00
Sun3 SunOS 3.5	448	1190	1012	1.00
MVAX2 Ultrix 3.0	740	1610	1820	1.11

Table 4: Time for select kernel call.

7. Block Copy

The fourth benchmark uses the *bcopy* procedure to transfer large blocks of data from one area of memory to another. It doesn’t exercise the operating system at all, but different operating systems differ for the same hardware because their libraries contain different bcopy procedures. The main differences, however, are due to the cache organizations and memory bandwidths of the different machines.

The results are given in Table 5. For each configuration I ran the benchmark with two different block sizes. In the first case, I used blocks large enough (and aligned properly) to use *bcopy* in the most efficient way possible, but small enough so that both the source and destination block would fit in the cache (if any). In the second case I increased the transfer size to be larger than the cache size, so that cache misses would occur continuously. In each case several transfers were made between the same source and destination, and the average bandwidth of copying is shown in Table 5.

Configuration	Cached (Mbytes/second)	Uncached (Mbytes/second)	Bytes/instruction
M2000 RISC/os 4.0	39	20	1.0
8800 Ultrix 3.0	22	16	2.7
Sun4 Sprite	11.1	5.0	0.55
DS3100 Sprite	10.2	5.4	0.43
DS3100 Ultrix 3.1	10.2	5.1	0.39
Sun4 SunOS 4.0	8.2	4.7	0.52
Sun3 Sprite	5.6	5.5	3.1
MVAX2 Ultrix 3.0	3.5	3.3	3.7

Table 5: Throughput of bcopy procedure for large blocks.

The last column in Table 5 is a relative figure showing how well each configuration can move large uncached blocks of memory relative to how fast it executes normal instructions. I computed this figure by taking the number from the second column (‘‘Uncached’’) and dividing it by the MIPS rating from Table 1. Thus, for the 8800 the value is (16/6) = 2.7. The most interesting thing to notice is that the CISC machines (8800, Sun3, and MVAX2) have normalized ratings of 2.5-4, whereas the RISC machines have ratings of 0.4-1.0. For the DEC and Sun RISC workstations, faster processors do not appear to have been accompanied by *any* increase in memory bandwidth. Thus, memory-intensive applications are not likely to scale on these machines. In fact, the relative performance of memory copying drops almost monotonically with faster processors, both for RISC and CISC machines.

8. Read from File Cache

This benchmark consists of a program that opens a large file and reads the file repeatedly in 16-kbyte blocks. For each configuration I chose a file size that would fit in the main-memory file cache. Thus the benchmark measures the cost of entering the kernel and copying data from the kernel’s file cache back to a buffer in the benchmark’s address space. The file was large enough that the data to be copied in each kernel call was not resident in any hardware cache. However, the same buffer was re-used to receive the data from each call; in machines with caches, the receiving buffer was likely to stay in the cache. Table 6 lists the overall bandwidth of data transfer, averaged across a large number of kernel calls.

The numbers in Table 6 reflect fairly closely the memory bandwidths from Table 5. The only noticeable difference is that the Sun4 does relatively better in this benchmark due to its write-back cache. Since the receiving buffer always stays in the cache, its contents get overwritten without ever being flushed to memory. For the DS3100, in contrast, the write-through cache causes information in the buffer to be flushed immediately to memory.

Configuration	Mbytes/second	MIPS-Relative Speed
M2000 RISC/os 4.0	15.6	0.45
8800 Ultrix 3.0	10.5	1.02
Sun4 SunOS 4.0	7.5	0.48
Sun4 Sprite	6.8	0.37
DS3100 Ultrix 3.1	4.8	0.21
DS3100 Sprite	4.4	0.16
Sun3 Sprite	3.7	1.0
Sun3 SunOS 3.5	3.1	1.0
MVAX2 Ultrix 3.0	2.3	1.48

Table 6: Bandwidth of reading from the file cache

9. Modified Andrew Benchmark

This is the one large-scale benchmark that I ran. It is a modified version of the Andrew benchmark developed by M. Satyanarayanan for measuring the performance of the Andrew file system [1]. The benchmark operates by copying a directory hierarchy containing the source code for a program, *stat*-ing every file in the new hierarchy, reading the contents of every copied file, and finally compiling the code in the copied hierarchy. In order to make the results comparable between different machines, I modified the benchmark so that it always uses the same compiler. In other words, regardless of which machine is executing the benchmark, the compiler is always the GNU C compiler generating code for a machine called SPUR.

The raw Andrew results are shown in Table 7. The table lists separate times for two different phases of the benchmark. The ‘‘copy’’ phase consists of everything except the compilation (all of the file copying and scanning), and the ‘‘compile’’ phase consists of just the compilation.

I ran the benchmark in both local and remote configurations. ‘‘Local’’ means that all the files accessed by the benchmark were stored on a disk attached to the machine running the benchmark. ‘‘Remote’’ means that as many files as possible were stored on a server machine and accessed over the network. In the Sprite and SunOS measurements, ‘‘remote’’ means that the benchmark was run on a diskless workstation, so absolutely all file accesses were remote. For the Ultrix measurements, temporary files used during compilation may have been stored locally even for the ‘‘remote’’ measurements. However, all of the files in the directory hierarchy being copied, and almost all of the program binaries, were stored remotely. For Ultrix and SunOS, NFS was used for remote accesses. For Sprite, Sprite’s caching network file system was used (see [2] for details). In each of the remote cases, the server was the same kind of machine as the client.

Table 8 gives additional ‘‘relative’’ numbers: the MIPS-relative speed for local operation, and the percentage slow-down experienced when the benchmark ran with a remote disk instead of a local one. No ‘‘Remote Penalty’’ figures are given for the M2000 and 8800 because I wasn’t able to benchmark them in a remote configuration.

Configuration	Copy (seconds)	Compile (seconds)	Total (seconds)
M2000 RISC/os 4.0 Local	13	59	72
DS3100 Sprite Local	22	98	120
DS3100 Sprite Remote	34	93	127
Sun4 Sprite Local	44	128	172
Sun4 SunOS 4.0 Local	46	130	176
Sun4 Sprite Remote	56	128	184
DS3100 Ultrix 3.1 Local	80	133	213
8800 Ultrix 3.0 Local	48	181	229
DS3100 Ultrix 3.1 Remote	115	154	269
Sun4 SunOS 4.0 Remote	108	162	270
Sun3 Sprite Local	52	375	427
Sun3 Sprite Remote	75	364	439
MVAX2 Ultrix 3.0 Local	214	1202	1416
MVAX2 Ultrix 3.0 Remote	298	1409	1707

Table 7: The modified Andrew benchmark

Configuration	MIPS-Relative Speed (Local)	Remote Penalty (%)
M2000 RISC/os 4.O Local	0.53	
8800 Ultrix	0.56	
Sun3 Sprite	1.0	3
DS3100 Sprite	0.49	6
Sun4 Sprite	0.50	7
MVAX2 Ultrix 3.0	0.60	21
DS3100 Ultrix 3.1	0.28	26
Sun4 SunOS 4.0	0.49	53

Table 8: Modified Andrew benchmark, cont'd

There are several interesting results in Tables 7 and 8. First of all, no operating system scaled to match hardware speedups. Second, Sprite comes out consistently faster than Ultrix or SunOS for remote access. Sprite shows hardly any performance degradation for remote access, and for the compilation phase Sprite was faster remote than local (I don't have a good explanation for why remote would be faster than local; at first I assumed that it was an experimental anomaly, but I have seen the effect on several different occasions). In contrast, NFS-based RISC workstations slow down by about 50% relative to local access. It appears to me that the relative penalty for using NFS is increasing as machine speeds increase (for example, the MVAX2 slowed down by only 20% when using NFS instead of a local disk).

The third interesting result of this benchmark is that the DS3100-Ultrix combination is somewhat slower than would have been expected. For example, DS3100-Ultrix-Local is about 70% slower than DS3100-Sprite-Remote, and DS3100-Ultrix-Remote is not much faster than Sun4-SunOS-Remote.

10. Open-Close

I ran two other benchmarks in an attempt to explain the results in Table 7. The first of these is open-close, a benchmark which repeatedly opens and closes a particular file. The results are shown in Table 9 for two cases: a name with only a single element, and one with 4 elements. In the local case, the UNIX derivatives are consistently faster than Sprite. In the remote case Sprite is faster than SunOS, but slower than Ultrix. In any case, this benchmark cannot explain the differences in Table 7.

Configuration	‘‘foo’’ (ms)	‘‘/a/b/c/foo’’ (ms)
DS3100 Ultrix 3.1 Local	0.27	0.41
M2000 RISC/os 4.0 Local	0.32	0.83
Sun4 SunOS 4.0 Local	0.34	0.43
8800 Ultrix 3.0 Local	0.45	0.68
DS3100 Sprite Local	0.82	0.97
Sun4 Sprite Local	1.2	1.4
MVAX2 Ultrix 3.0 Local	2.9	4.7
DS3100 Ultrix 3.1 Remote	3.8	3.9
DS3100 Sprite Remote	4.3	4.4
Sun3 Sprite Local	4.3	5.2
Sun4 Sprite Remote	6.1	6.4
Sun3 Sprite Remote	12.8	16.3
MVAX2 Ultrix 3.0 Remote	36.0	36.9

Table 9: Time to open and close a file

11. Create-Delete

The last benchmark was perhaps the most interesting in terms of identifying differences between operating systems. It also helps to explain the results in Table 7. This benchmark simulates the creation, use, and deletion of a temporary file. It opens a file, writes some amount of data to the file, closes the file, then opens the file for reading, reads the data, closes the file, and finally deletes the file. I tried three different amounts of data: none, 10 kbytes, and 100 kbytes.

Table 10 gives the total time to create, use, and delete the file in each of several hardware/operating system configurations.

Configuration	No data (ms)	10 kbytes (ms)	100 kbytes (ms)
DS3100 Sprite Local	17	34	69
Sun4 Sprite Local	18	33	67
DS3100 Sprite Remote	33	34	68
Sun3 Sprite Local	33	47	130
M2000 RISC/os 4.0 Local	33	51	116
Sun4 Sprite Remote	34	50	71
8800 Ultrix 3.0 Local	49	100	294
Sun3 Sprite Remote	61	73	129
Sun4 SunOS 4.0 Local	66	830	940
DS3100 Ultrix 3.1 Local	80	146	548
MVAX2 Ultrix 3.0 Local	100	197	841
DS3100 Ultrix 3.1 Remote	116	370	3028
MVAX2 Ultrix 3.0 Remote	295	634	2500

Table 10: Time to create, use, and delete a file

This benchmark highlights a basic difference between Sprite and UNIX derivatives. In Sprite, short-lived files can be created, used, and deleted without any data ever being written to disk. Information only goes to disk after it has lived at least 30 seconds. In UNIX and its derivatives, the file system appears to be much more closely tied to the disk. Even with no data written in the file, the UNIX derivatives all required 35-100 ms to create and delete the file, regardless of the performance of the machine. This suggests that the creation and deletion operations are forcing data to disk and waiting for the disk operations to complete.

The create-delete benchmark also helps to explain the poor performance of DS3100 Ultrix on the Andrew benchmark. The basic time for creating an empty file is 60% greater in DS3100-Ultrix-Local than in 8800-Ultrix-Local, and the time for a 100-kbyte file in DS3100-Ultrix-Remote is 45 times as long as for DS3100-Sprite-Remote! The poor performance relative to the 8800 may perhaps be due to slower disks (RZ55's on the DS3100's); the poor remote performance is probably due to NFS's writing policy, which requires new data to be written through to disk when the file is closed. Note that DS3100-Ultrix-Remote achieves a write bandwidth of only about 30 kbytes/sec. This is almost twice as slow as I measured on the same hardware running an earlier version of Ultrix (3.0), and also about twice as slow as I have measured previously on Sun machines running SunOS and NFS.

Lastly, Table 10 exposes some suprising behavior in SunOS 4.0. Note that the time for a file with no data is 66 ms, but the time for 10 kbytes is 830 ms! This surpised me, so I also tried data sizes of 2-9 kbytes at 1-kbyte intervals. The SunOS time stayed in the 60-80ms range until the file size increased from 8 kbytes to 9 kbytes; at this point it jumped up to the 800-ms range.

12. Conclusions

In almost every benchmark the faster machines ran more slowly than I would have guessed from raw processor speed. In some cases, like getpid and cswitch, I don't have a good explanation for the discrepancy (the additional registers in the RISC machines cannot account for the difference all by themselves, for example). However, some of the benchmarks highlight issues for both hardware designers and operating systems people to think about.

On the hardware side, memory bandwidth has been allowed to slip relative to processor speed. If this trend continues, future machines (particularly workstations where cost considerations may tempt designers to skimp on memory system performance) are likely to be limited in performance by overall memory bandwidth. A fast cache may reduce the need for low-latency main memory, but it doesn't eliminate the need for high bandwidth in the main memory.

On the software side, operating system designers need to decouple file system performance from disk performance. Operating systems derived from UNIX use caches to speed up reads, but they require synchronous disk I/O for operations that modify files. If this coupling isn't eliminated, a large class of file-intensive programs will receive little or no benefit from faster hardware. Of course, delaying disk writes may result in information loss during crashes; the challenge for operating system designers is to maintain reliability while decoupling performance.

A final consideration is in the area of network protocols. In my (biased) opinion, the assumptions inherent in NFS (statelessness and write-through-on-close, in particular) represent a fundamental performance limitation. If users are to benefit from faster machines, either NFS must be scrapped (my first choice), or NFS must be changed to be less synchronous.

13. References

[1] John H. Howard, Michael L. Kazar, Sherri G. Menees, David A. Nichols, M. Satyanarayanan, Robert N. Sidebotham, and Michael J. West. Scale and Performance in a Distributed File System. *ACM Transactions on Computer Systems* 6(1):51-81, February, 1988.

[2] Michael N. Nelson, Brent B. Welch, and John K. Ousterhout. Caching in the Sprite Network File System. *ACM Transactions on Computer Systems* 6(1):134-154, February, 1988.

[3] John K. Ousterhout, Andrew R. Cherenson, Fred Douglis, Michael N. Nelson, and Brent B. Welch. The Sprite Network Operating System. *IEEE Computer* 21(2):23-36, February, 1988.

WRL Research Reports

‘‘Titan System Manual.’’
Michael J. K. Nielsen.
WRL Research Report 86/1, September 1986.

‘‘Global Register Allocation at Link Time.’’
David W. Wall.
WRL Research Report 86/3, October 1986.

‘‘Optimal Finned Heat Sinks.’’
William R. Hamburgen.
WRL Research Report 86/4, October 1986.

‘‘The Mahler Experience: Using an Intermediate Language as the Machine Description.’’
David W. Wall and Michael L. Powell.
WRL Research Report 87/1, August 1987.

‘‘The Packet Filter: An Efficient Mechanism for User-level Network Code.’’
Jeffrey C. Mogul, Richard F. Rashid, Michael J. Accetta.
WRL Research Report 87/2, November 1987.

‘‘Fragmentation Considered Harmful.’’
Christopher A. Kent, Jeffrey C. Mogul.
WRL Research Report 87/3, December 1987.

‘‘Cache Coherence in Distributed Systems.’’
Christopher A. Kent.
WRL Research Report 87/4, December 1987.

‘‘Register Windows vs. Register Allocation.’’
David W. Wall.
WRL Research Report 87/5, December 1987.

‘‘Editing Graphical Objects Using Procedural Representations.’’
Paul J. Asente.
WRL Research Report 87/6, November 1987.

‘‘The USENET Cookbook: an Experiment in Electronic Publication.’’
Brian K. Reid.
WRL Research Report 87/7, December 1987.

‘‘MultiTitan: Four Architecture Papers.’’
Norman P. Jouppi, Jeremy Dion, David Boggs, Michael J. K. Nielsen.
WRL Research Report 87/8, April 1988.

‘‘Fast Printed Circuit Board Routing.’’
Jeremy Dion.
WRL Research Report 88/1, March 1988.

‘‘Compacting Garbage Collection with Ambiguous Roots.’’
Joel F. Bartlett.
WRL Research Report 88/2, February 1988.

‘‘The Experimental Literature of The Internet: An Annotated Bibliography.’’
Jeffrey C. Mogul.
WRL Research Report 88/3, August 1988.

‘‘Measured Capacity of an Ethernet: Myths and Reality.’’
David R. Boggs, Jeffrey C. Mogul, Christopher A. Kent.
WRL Research Report 88/4, September 1988.

‘‘Visa Protocols for Controlling Inter-Organizational Datagram Flow: Extended Description.’’
Deborah Estrin, Jeffrey C. Mogul, Gene Tsudik, Kamaljit Anand.
WRL Research Report 88/5, December 1988.

‘‘SCHEME->C A Portable Scheme-to-C Compiler.’’
Joel F. Bartlett.
WRL Research Report 89/1, January 1989.

‘‘Optimal Group Distribution in Carry-Skip Adders.’’
Silvio Turrini.
WRL Research Report 89/2, February 1989.

‘‘Precise Robotic Paste Dot Dispensing.’’
William R. Hamburgen.
WRL Research Report 89/3, February 1989.

‘‘Simple and Flexible Datagram Access Controls for Unix-based Gateways.’’
Jeffrey C. Mogul.
WRL Research Report 89/4, March 1989.

‘‘Spritely NFS: Implementation and Performance of Cache-Consistency Protocols.’’
V. Srinivasan and Jeffrey C. Mogul.
WRL Research Report 89/5, May 1989.

‘‘Available Instruction-Level Parallelism for Superscalar and Superpipelined Machines.’’
Norman P. Jouppi and David W. Wall.
WRL Research Report 89/7, July 1989.

‘‘A Unified Vector/Scalar Floating-Point Architecture.’’
Norman P. Jouppi, Jonathan Bertoni, and David W. Wall.
WRL Research Report 89/8, July 1989.

‘‘Architectural and Organizational Tradeoffs in the Design of the MultiTitan CPU.’’
Norman P. Jouppi.
WRL Research Report 89/9, July 1989.

‘‘Integration and Packaging Plateaus of Processor Performance.’’
Norman P. Jouppi.
WRL Research Report 89/10, July 1989.

‘‘A 20-MIPS Sustained 32-bit CMOS Microprocessor with High Ratio of Sustained to Peak Performance.’’
Norman P. Jouppi and Jeffrey Y. F. Tang.
WRL Research Report 89/11, July 1989.

‘‘Leaf: A Netlist to Layout Converter for ECL Gates.’’
Robert L. Alverson and Norman P. Jouppi.
WRL Research Report 89/12, July 1989.

‘‘The Distribution of Instruction-Level and Machine Parallelism and Its Effect on Performance.’’
Norman P. Jouppi.
WRL Research Report 89/13, July 1989.

‘‘Long Address Traces from RISC Machines: Generation and Analysis.’’
Anita Borg, R.E.Kessler, Georgia Lazana, and David W. Wall.
WRL Research Report 89/14, September 1989.

‘‘Link-Time Code Modification.’’
David W. Wall.
WRL Research Report 89/17, September 1989.

WRL Technical Notes

‘‘TCP/IP PrintServer: Print Server Protocol.’’
Brian K. Reid and Christopher A. Kent.
WRL Technical Note TN-4, September 1988.

‘‘TCP/IP PrintServer: Server Architecture and Implementation.’’
Christopher A. Kent.
WRL Technical Note TN-7, November 1988.

‘‘Smart Code, Stupid Memory: A Fast X Server for a Dumb Color Frame Buffer.’’
Joel McCormack.
WRL Technical Note TN-9, September 1989.

‘‘Why Aren’t Operating Systems Getting Faster As Fast As Hardware?’’
John Ousterhout.
WRL Technical Note TN-11, October 1989.

‘‘Mostly-Copying Garbage Collection Picks Up Generations and C++.’’
Joel Bartlett.
WRL Technical Note TN-12, October 1989.

Table of Contents

List of Tables