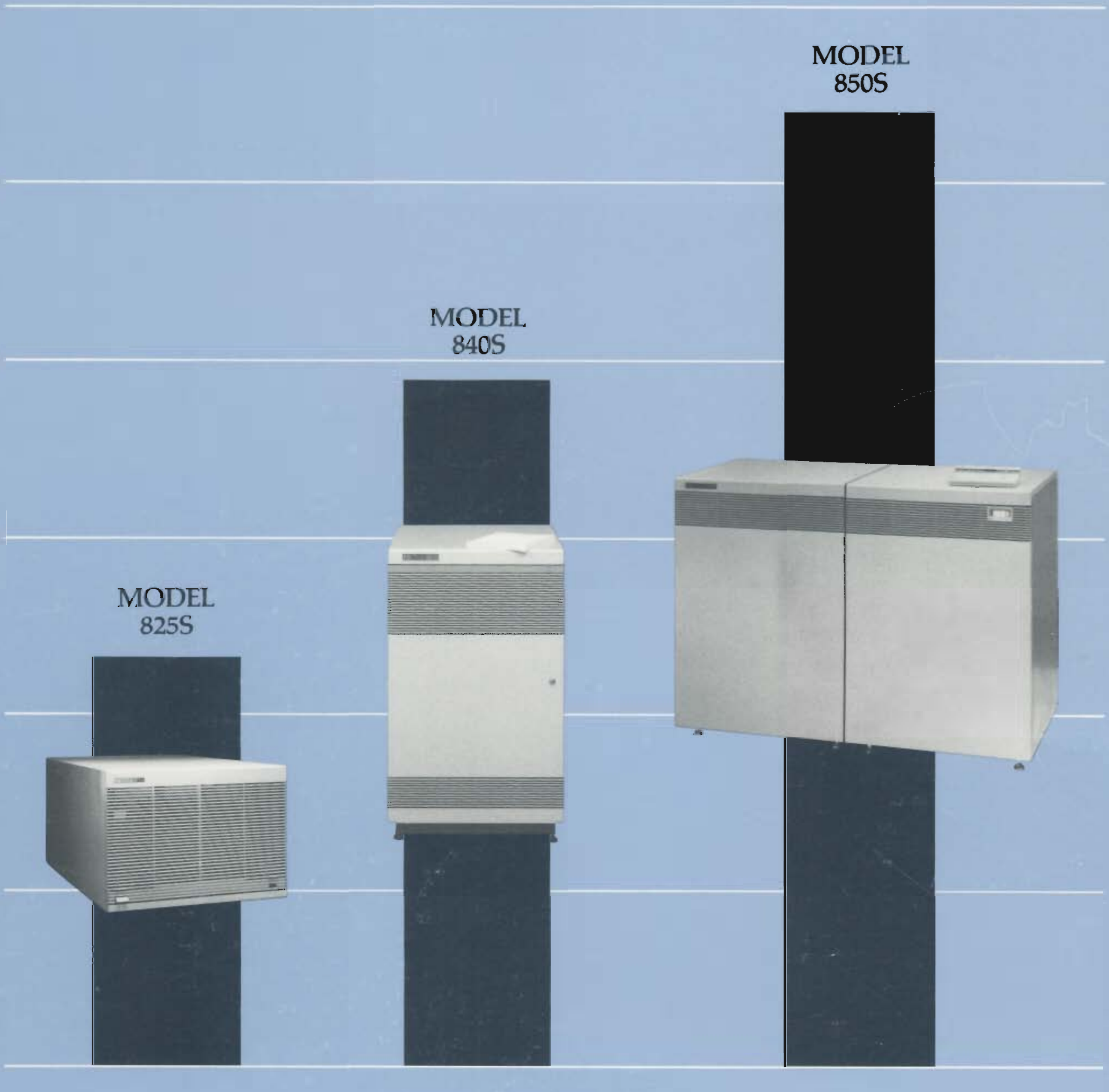
HP 9000 Series 800

Performance Brief

HEWLETT PACKARD



Includes Whetstone, Linpack, Multiuser, AIM, and others

HEWLETT
PACKARD

HP 9000 Series 800
Performance Brief

Section 1: Introduction

Objective and Performance Highlights

When performance is critical to the productivity and success of your business, selecting the right system is of utmost importance. The right system means having the right performance for the right price. It must offer an affordable solution to your application needs and grow as your business grows.

The Series 800 systems -- the Model 825S, 840S and 850S -- are the first three HP-UX members of the HP Precision Architecture family. The Series 800 are high end UNIX[1] systems offering unequaled price/performance ratios. Enhanced by the real-time functionality included in HP-UX, they are logical extensions to both the HP 1000 and HP 9000 families.

Definition of HP-UX

HP-UX has passed compliance tests imposed by the System V Verification Suite (SVVS), which was issued by AT&T to verify conformance with the UNIX System V Interface Definition (SVID). HP-UX on the Series 800 systems is also a superset of AT&T's UNIX System V, Release 2.0 (excluding hardware dependencies) and conforms to the base system plus kernel extensions of SVID (Issue 2, Volume 1). Thus portability of applications developed on other SVID compliant systems is assured. The HP-UX operating system also incorporates features of U.C. Berkeley release 4.2 BSD. Finally, HP has significantly extended HP-UX to include real-time functionality, high performance file access, device I/O programming, Native Language Support (internationalization) and more.

In sum, HP-UX is an exceptionally powerful, standards-based operating system which offers high capacity, high performance, and excellent transportability of applications between heterogenous systems.

[1] UNIX is a registered trademark of AT&T in the USA and other countries.

Objective of the Brief

The objective of this document is to provide performance information on the Series 800, Models 825S, 840S and 850S that will provide assistance in evaluating and choosing between these systems for a particular application.

The brief presents the performance of the Series 800 systems in the areas listed below.

* System throughput performance
* CPU performance benchmarks
* UNIX system competitive benchmarks
* Real-time performance measurements
* Database performance
* Data acquisition measurements
* Network Services performance
* NS impact on real-time response

Appendices detail the system configurations and benchmark tests.

Series 800 Performance Overview

In all the performance measurements in this brief, the following consideration should be kept in mind: While the Model 825S has a faster clock speed than the Model 840S (12.5 MHz versus 8 MHz), the cache of the Model 825S is much smaller than that of the 840S (16 KBytes versus 128 KBytes). The penalty for a cache miss is relatively large in a RISC-based machine, so the faster clock speed of the Model 825S will dominate in compute intensive workloads (such as Dhrystone or CAD-type workloads) where the full benefit of a large cache is not utilized. On the other hand, the very large and fast cache memory of the Model 840S will become extremely important within multiuser environments, during execution of operating system code, and in any situation where relatively worse locality of reference is experienced (such as certain real-time tests and throughput benchmarks).

Table 1.1. Precision Architecture MIPS: Relative Ratings by Environment

Environment	Benchmark	825S	840S	850S
Single User				
Floating pt.	HPSpice-mosamp2	0.75	1.0	1.57
Non-floating pt.	HILO	0.83	1.0	1.46
Seven users				
Floating pt.	HPSpice-greycode	0.88	1.0	1.54
Non-floating pt.	HILO	0.88	1.0	1.60
General (32 Users)	Program Development	0.68	1.0	1.57
Process Control (16 users)	YEWCOM Benchmark	0.82	1.0	1.53

MIPS Analysis

MIPS is a measure which provides a value for the number of (millions of) instructions executed by the CPU in one second. Some vendors announce optimistic MIPS ratings casually as a lump sum statistic, but *a single MIPS rating does not account for overall efficiency of the instructions or overall system performance*. It is a measure of workload-dependent raw CPU speed and is one of the many performance measures that should be considered. Series 800 systems are enhanced RISC machines, so MIPS ratings approach the ideal of one instruction per clock cycle (in the 8 MegaHertz Model 840S, one second divided by 125 nanoseconds equals a maximum of 8 million instructions per second). HP uses *actual measured* MIPS rather than "potential" MIPS for rating Series 800 machines, which yields an average of approximately 4.5 MIPS for the Model 840S.

HP has analyzed CPU performance by major environment for Series 800 machines. The relative MIPS numbers in Table 1.1 above were derived by taking 20 traces of benchmarks selected to be representative of each environment.

In summary, for a general multiuser workload, the MIPS rating of the Model 825S is approximately 0.7 times that of the Model 840S; the MIPS rating of the Model 850S is approximately 1.6 times that of the Model 840S.

Multiuser benchmarks are designed to measure real working environments.

CPU benchmarks measure integer, floating point and programming environment performance.

Competitive benchmarks provide further evidence that the Series 800 systems are high performing UNIX systems.

Real-time benchmarks show that typical real-time performance of the Series 800 systems is close to that of the HP 1000 A900 -- an ability unprecedented by any other UNIX system vendor. Real-time capability is a complex issue which cannot be adequately explained in one sentence. Please see the HP-UX Real-Time section in this brief.

Database benchmarks show the results of 10 types of transactions for Series 800 systems.

Data acquisition benchmarks measure performance in this real-time market environment.

Network Services benchmarks measure Network File Transfer between HP 9000 Models 840S and 320 and the HP 1000 A900, and Remote File Access between the HP 9000 Models 840S and 320.

NS Impact on Real-Time Performance: Interrupt response and process dispatch times as impacted by NS services are measured.

Section 2: Multiuser/Application Performance

System Throughput

The benchmarks in this section have been designed to characterize overall performance such as will be seen by a typical customer.

For the following three benchmarks the systems are loaded using a multiuser synthetic workload. A multiuser environment is simulated by running multiple background tasks ("users") which are generated at a single terminal.

The graphs which follow illustrate the transaction rate as the number of users increases. A transaction here is defined as "any command that causes a prompt". Some transactions will take longer than others. The important issue is that the transaction mix should be an accurate representation of the simulated environment and that the same environment is used when comparing different systems The Transaction Rate curve gives a system view of the total work being done and will increase with each added user (or task) up to a knee, and then decrease. As the knee is approached, the system resources are used more completely. Beyond the knee these resources are used less efficiently resulting in a reduction of total work being accomplished by the system.

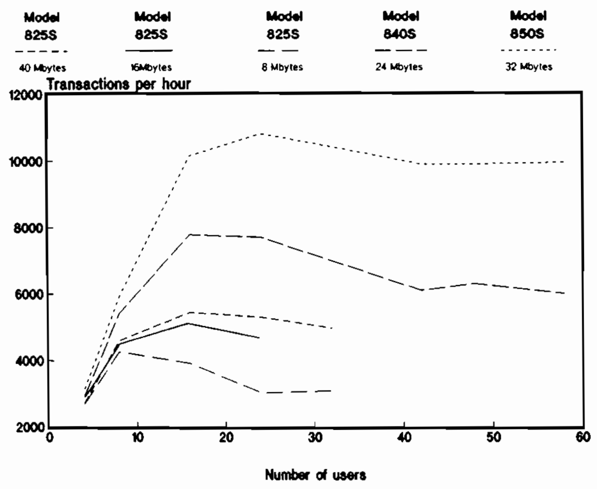
The phenomena described above are characteristic of all multiuser and multitasking systems. The shape and location of the knee depends on the application and the performance characteristics of the system.

Program Development

The program development application graphed in Figure 2.1 uses a 930-line, 20,000 byte FORTRAN program derived from the B1D Whetstone benchmark. The automated sequence for each "user" is:

* Compile the program with errors
* Edit (using "vi") to correct all but one error
* Compile again (with error)
* Edit to correct error
* Successfully compile and link the program
* Execute the program
* Execute several commands from the shell, including ls, pwd, and date

Figure 2.1. Program Development Environment



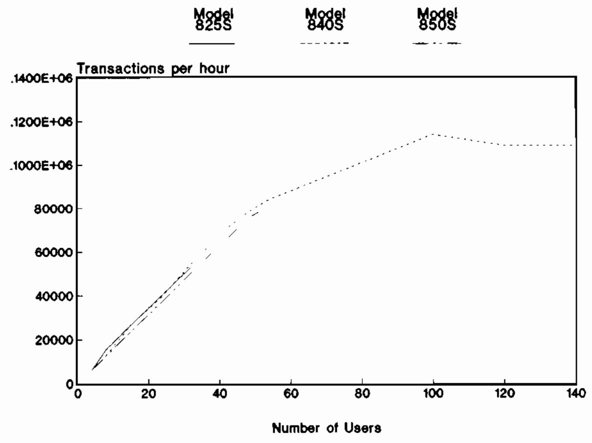
Editing Environment

This interactive editing application uses the full-screen "vi" editor to add and delete text from a large file. The automated sequence for each "user" is:

* List file in the directory
* Enter "vi"
* Read in file
* Add text to file
* Delete text from file
* Move around in file
* Write to file

Figure 2.2 shows the results for up to 32 users on the Model 825S and up to 128 users on the Model 840S. Data for more than 51 users is not yet available for the Model 850S. Only about 7% throughput degradation is seen between the peak performance (100 users) and performance for the maximum supported number of users (128) on the Model 840S.

Figure 2.2. Editing Environment



SPICE Benchmark

SPICE is a standard circuit simulator written in FORTRAN. Performance listed in Table 2.1 and graphed in Figure 2.3 is measured using two versions of the program (HPSpice and Berkeley Spice) simulating MOSAMP2, UA741 and Greycode circuits.

Figure 2.3. SPICE Benchmark (Relative Throughput)

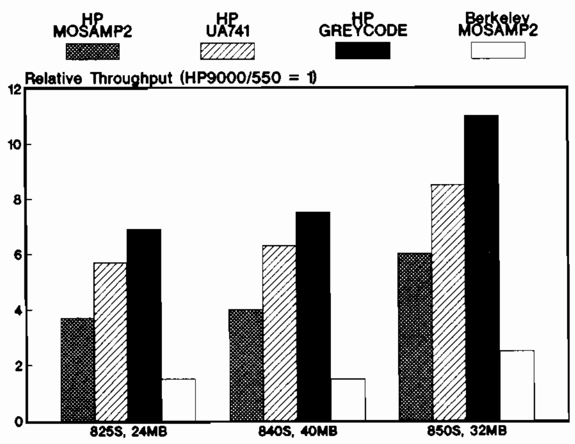


Table 2.1. SPICE Benchmark (Elapsed Seconds)

System	HPSPICE UA741	HPSPICE GREYCODE
Model 825S	25	393
Model 840S	23	366
Model 850S	16	238
	HPSPICE MOSAMP2	**BERKELEY MOSAMP2**
Model 825S	38	72
Model 840S	32	67
Model 850S	21	44

CAE/CAD Benchmarks

These benchmarks are real application programs. They provide performance data for engineering design and analysis tasks that are commonly performed on workstations. Figure 2.4 and Table 2.2 summarize the results.

The **CSIM** benchmark uses a circuit simulator program with 2941 lines of C source code. It takes as input files a flattened BDL file which is used to build the connectivity lists -- and a test vector file containing the list of vectors to be tested.

PLAGEN uses a PLA (Programmable Logic Array) generator which reads a file of boolean equations, constructs a truth table and minimizes it. The input files used for the benchmark are 70 and 145 lines long.

ROUTER simulates the computationally intensive parts of an actual router. It uses a 64-bit bitmap and the same underlying algorithm as a real router. For this test the control loop is set at 30 floods and repeatedly performs the following: Initialize cost queue, create a routing window, create a source and a target, flood inside the window, clear away the flood.

Figure 2.4. CAE/CAD Benchmarks

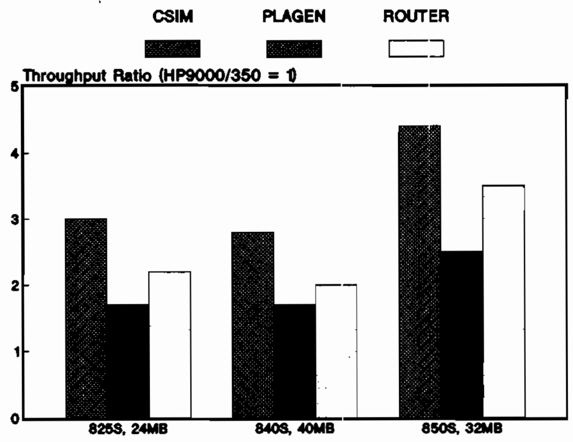


Table 2.2. CAE/CAD Benchmarks (User Seconds)

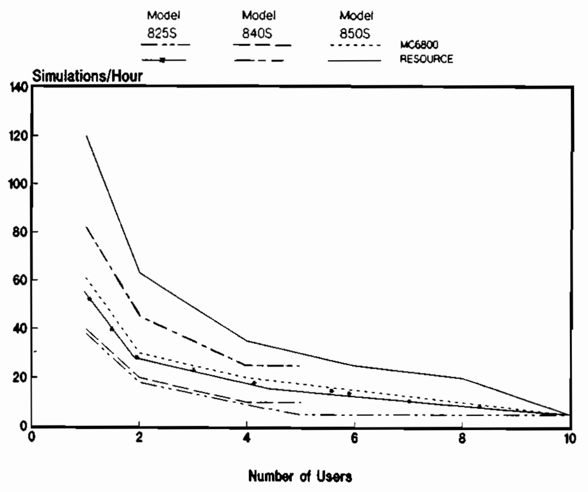
System	CSIM	PLAGEN	ROUTER
Model 825S	27.3	44.5	64.8
Model 840S	30.4	46.8	71.5
Model 850S	19.2	30.3	41.8

HILO Benchmark

This benchmark was designed as a performance test for HILO -- a logic simulator sold by HP as part of the EE-CAD workbench. Four basic circuits are simulated: ISOR, MC6800, RESOURCE, and WCS. Figure 2.5 shows the results of MC6800 (which uses a higher level modeling construct) and RESOURCE (which uses mainly modeling primitives).

The benchmark is run in batch mode with 1 through 10 concurrent simulations (Users) for each circuit. The results shown in Figure 2.5 are the average simulations per hour of 3 runs per circuit simulation. As the results show, simulating a circuit with HILO uses most or all of the CPU. HILO does not use floating point.

Figure 2.5. HILO Benchmark



Section 3: CPU Performance Benchmarks

Series 800 Systems Excel in Industry Benchmarks

This section presents the results of several benchmarks, published in computer industry magazines and journals. Raw CPU performance as measured in this section is the area where the clock speed of the Model 825S will allow it to achieve better results than the Model 840S because these benchmarks perform tests within a single user environment.

Whetstone

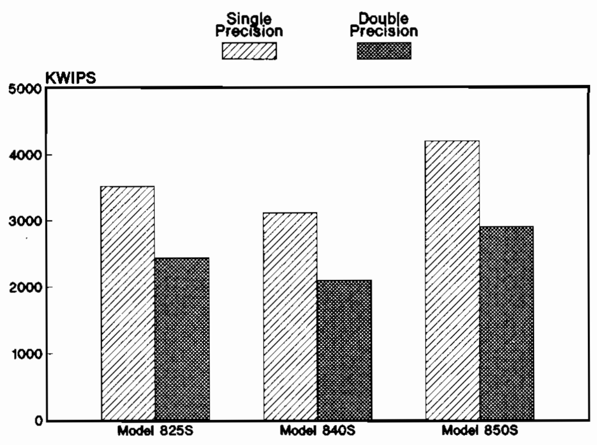
The Whetstone benchmarks were written by the National Physical Laboratory in England. The single and double precision benchmarks graphed in Figure 3.1, are referred to in the industry as the Whetstone B1 and B1D benchmarks, respectively. These FORTRAN programs were derived from an analysis of one thousand ALGOL 60 programs as an attempt to represent an average program instruction mix. The many different operations represented include floating point and integer calculations, transcendental functions, array manipulation, and conditional jumps.

The measurement is expressed in terms of an entity known as a Whetstone instruction. One loop through the program represents one million Whetstone instructions. The Whetstone results below are expressed in KWIPS (Thousands of Whetstone Instructions Per Second).

Table 3.1. Whetstone Benchmark in KWIPS

System	Single-B1 KWIPS	Double-B1D KWIPS
Model 825S	3521	2433
Model 840S	3115	2092
Model 850S	4202	2907

Figure 3.1. Whetstone Benchmark in KWIPS



Dhrystone

The Dhrystone benchmark simulates a high-level C programming environment. The program contains 100 statements (1 dhrystone) which represent a balance of statement type, operand type and operand access. The program contains 53% assignments, 32% control statements, and 15% function calls. There are two significant versions of the C Dhrystone benchmark -- known as 1.0 and 1.1. Two widely known technical flaws exist in the 1.0 benchmark -- one line of code is missing (the benchmark was originally written in Ada), and some results computed by the benchmark are never used, so code can be completely removed by an optimizing compiler. The missing line of code is included in 1.1, which is shown here. Additionally, the instructions for the benchmark indicate that only very basic optimizations can be performed on the code. Figure 3.2 illustrates the results of Table 3.2 of the Dhrystone 1.1 benchmark in Dhrystones per second for two separate cases: with registers declared prior to execution and without registers declared (see next page). Version 1.1 typically runs about 15% slower than version 1.0.

Figure 3.2. Dhrystone 1.1 Benchmark (Dhrystones per Second)

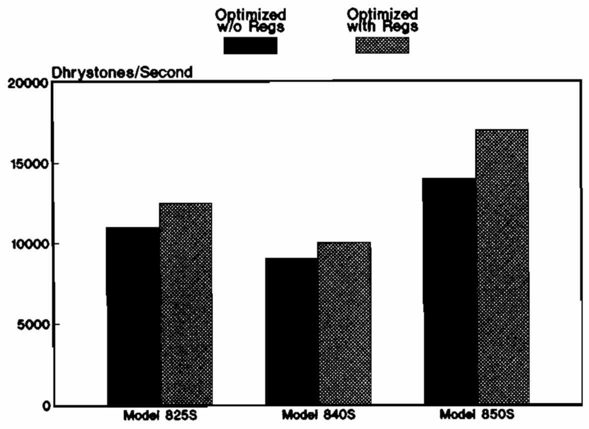


Table 3.2. Dhrystone 1.1 Benchmark (Dhrystones per Second)

System	Optimized (w/o Regs)	Optimized (with Regs)
Model 825S	11111	12639
Model 840S	8835	9920
Model 850S	13774	15576

Linpack

This floating point intensive benchmark solves a dense system of linear equations of order 100 in a FORTRAN environment. Linpack is the most widely accepted benchmark of CPU floating point performance. The results are measured in thousands of floating point operations per second (KFLOPS) for single and double precision. The Linpack benchmarks allow the *BLAS* subroutines (basic linear algebra subprograms) to be re-coded in Assembly. Figure 3.3 graphs the results in Table 3.3 of the Models 825S, 840S and 850S with, and without BLAS subroutines recoded in Assembly.

Figure 3.3. Linpack Benchmarks in KFLOPS

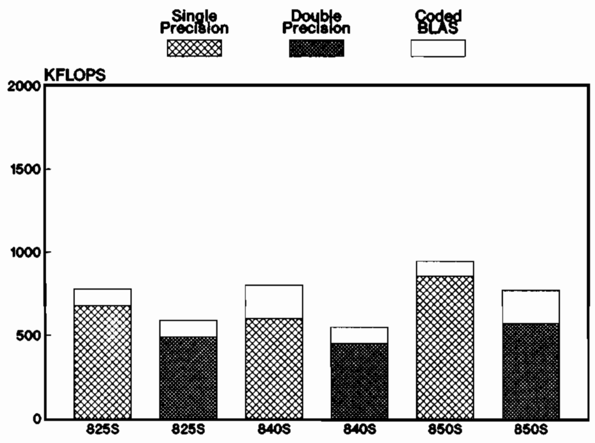


Table 3.3. Linpack Benchmark in KFLOPS

System	Single Precision	
	Fortran BLAS	Assembly BLAS
Model 825S	624	680
Model 840S	601	738
Model 850S	858	869
	Double Precision	
	Fortran BLAS	Assembly BLAS
Model 825S	491	537
Model 840S	450	542
Model 850S	572	715

Section 4: Competitive Benchmarks

Benchmarks for UNIX Systems

Aim Benchmark Suite II

This section focuses on the ability of the Series 800 systems to perform the Aim Benchmark Suite II. Test results of Aim Benchmarks have been published in various trade journals for many UNIX systems and may be used to contrast the performance of the Series 800 systems with other computers in their class. Aim Technology also offers tools for analyzing and reporting measurement results to aid in the comparison of selected UNIX systems under evaluation.

Summary of Results -- Suite II

The list of benchmarks contained in Suite II total 37. The results in Figures 4.1 through 4.6, and Table 4.1 are grouped into the seven categories listed below. See Appendix A for Aim Technology's description of the tests.

* Arithmetic Instruction Times
* Memory Loop Access Times
* Input/Output Rates
* Array Subscript References
* Function References
* Process Forks
* System Kernel Calls

The Aim Benchmark results reflect the high performance of HP's Precision Architecture and further validate the excellent price/performance ratios of the Series 800 systems. The following discusses several points worth noting as the data are interpreted.

The **Arithmetic Instruction Timers** (Figure 4.1) exercise arithmetic operations for 32 loops so that a multiple of 512 operations are measured per test.

The **Memory Loop Access** tests (Figure 4.2) examine RAM. Memory read, write and copy times are measured in short (2 bytes), long (4 bytes) and character (1 byte) modes. Each test moves a multiple of 512 bytes and measures the maximum RAM-to-RAM rates.

Input/Output tests include disc bandwidth, pipe throughput (Figure 4.3) and TTY output timers. In the disc tests, disc files are written, read and copied using 10 KByte chunks. No processing of files is done; I/O calls only are used. The pipe test pipes information between cooperating programs. There are two TTY tests which fork one and two subprocesses, each writing n-blocks (512 bytes) to a single TTY. The second test measures the capability to output to two ports simultaneously.

The Series 800 machines obtain substantially faster TTY speeds at higher baud rates. These tests are limited by the 9600 baud rate of the terminal and do not fully illustrate the true potential of the Series 800 systems in this area. Series 800 systems perform the pipe tests well, but still better performance can be obtained using HP-UX's shared memory facility.

The **Array Subscript** tests (Figure 4.4) time nested short and long subscript references. An inner loop is used, so that 512 references are executed for each n.

Function Reference tests (also Figure 4.4) call functions with zero, one and two parameters. Each function is called a multiple of 512 times.

The **Fork Timer** (Figure 4.5) measures how fast it takes to fork off n-subprocess and as such, measures the upper limit on how fast shell scripts can respond. Applications can achieve even faster performance in this area by using vfork (provided in HP-UX).

The **System Kernel Call** tests (Figure 4.6) perform kernel functions a multiple of 512 times. It is important to note that when comparing systems, some implementations treat an argument of zero to sbrk as a special case and bypass the system call. This results in an abnormally good rating in this area and invalidates the intended use of the benchmark. The Series 800 systems do NOT bypass the system for this condition.

Overall

The Series 800 optimizing C compiler was used in performing the tests. As a result, some benchmarks (such as add_float and add_double) in the suite were invalidated (i.e. the optimizer recognized unused code and removed it). In doing so, the intended purpose of the test was also removed. For these particular tests the reported results are obtained from the unoptimized execution of the benchmarks. Such results are flagged with a star (*).

AIM Disc Benchmark Discussion

The AIM disc benchmarks show unrepeatable results on the Series 800 systems. They use an adaptive timing method to decide how much "work" to measure. This works well in most cases. For example, the speed of multiplication is relatively independent of the number of multiplications done.

On HP's implementation of the file system, however, this assumption of invariance does not hold. We can write 100 KBytes more than 100 times faster than we can write 10 MBytes.

The method that AIM uses to determine how much to measure is highly dependent on small timing variations. As a result, the disc measurement may measure access of as little as .5 MBytes or as much as 8 MBytes. This explains why successive runs of AIM will give consistent results for most measurements, but wildly varying ones for the file system measurements. These results are flagged with a dagger (†).

The new disc benchmarks described in the Data Acquisition Section of this brief avoid these problems by writing a consistent amount of data (4 MBytes). Thus they give repeatable results.

The results of the AIM Benchmarks are graphed on pages 10 and 11, and tabulated on page 12.

Figure 4.1. Arithmetic Instruction Times

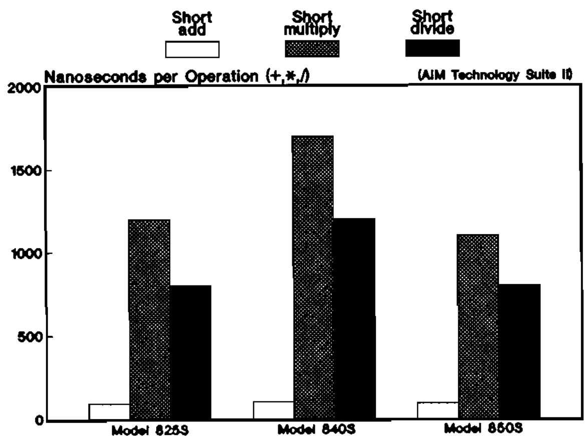


Figure 4.2. Memory Loop Access

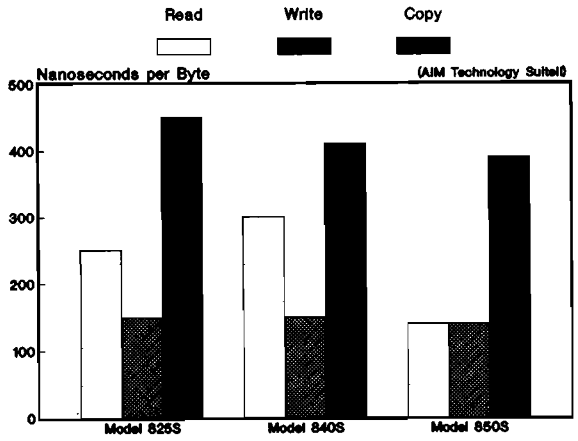


Figure 4.3. Pipe Throughput

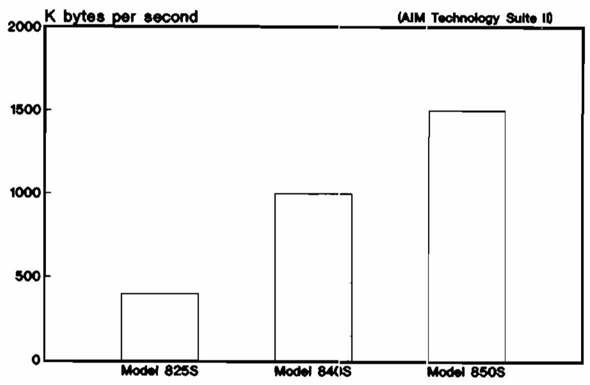


Figure 4.5. Process Forks

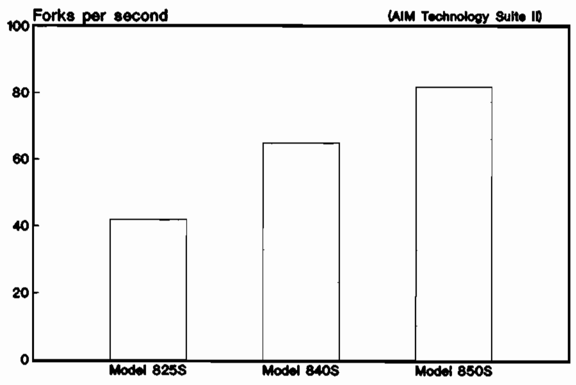


Figure 4.4. Array and Function References

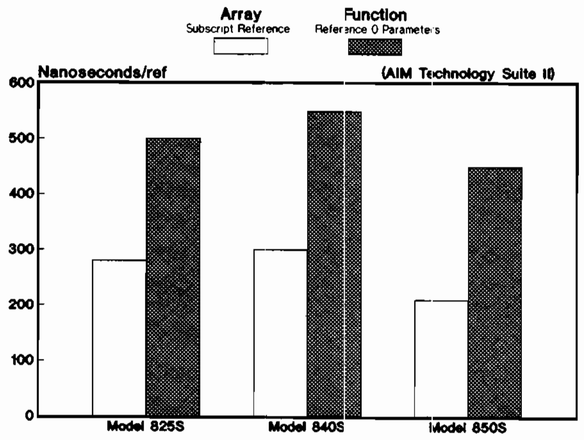


Figure 4.6. System Kernel Calls

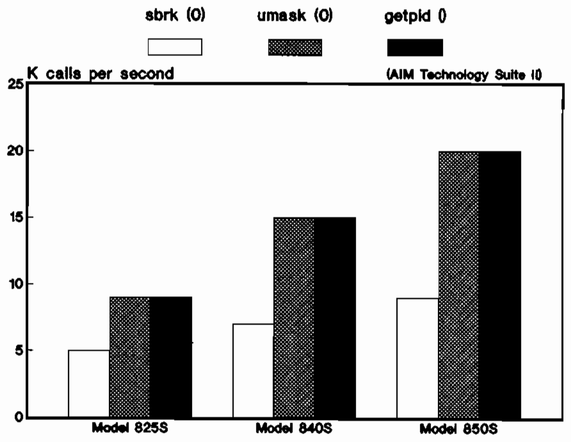


Table 4.1. Aim Benchmark Suite II Results

Arithmetic Instruction Times (nanoseconds per operation)												
	825				840				850			
Operation	Short	Long	Float	Dbl	Short	Long	Float	Dbl	Short	Long	Float	Dbl
+ Add	119	113	30*	30*	172	163	42*	42*	107	102	26*	26*
* Multiply	1200	1310	1800	1500	2000	2000	2000	2000	1100	1180	1600	1400
/ Divide	840	847	2750	2600	1000	1000	3000	1000	751	752	2500	2300

Memory Loop Access Times (nanoseconds per bytes)									
	read	write	copy	read	write	copy	read	write	copy
Char type	267	164	463	295	164	424	147	145	413
Short type	83	80	234	186	83	213	74	72	108
Long type	43	41	117	75	42	108	38	36	104

Input/Output Rates (K bytes/second)									
	read	write	copy	read	write	copy	read	write	copy
Disk†	NA	NA	NA	606	374	282	NA	NA	NA
Pipe			455			1036			1471
TTY 1		959			958			954	
TTY 1 + 2		2000			1916			2000	
RAM 1-byte			2159			2359			2418
RAM 4-byte			8543			9285			9590

Array Subscript References (nanoseconds)						
Type	short[]	long[]	short[]	long[]	short[]	long[]
	271	271	293	293	242	242

Function references (nanoseconds/ref)									
No. params	fn()	fn(i)	fn(i,i)	fn()	fn(i)	fn(i,i)	fn()	fn(i)	fn(i,i)
	512	516	466	542	542	671	459	458	418
Forks/sec	44			66			84		

System Kernel Calls (calls-per-second and microseconds per call)						
	calls/sec	usec/call	calls/sec	usec/call	calls/sec	usec/call
getpid():	9K	110	15K	66	20K	50
sbrk():	5K	219	8K	121	9K	114
creat/close:	45 pairs	22222/pr	45 pairs	22222/pr	45 pairs	22222/pr
umask(0):	9K	107	15K	65	20K	50

* The optimizing compiler trivializes the float and double add benchmarks so times reported are incorrect.

† Disk throughput numbers depend on the characteristics of the file system used and can vary widely for different configurations. Block size was 8K and frag size was 1K for this test which is not optimal.

BYTE Benchmarks

Table 4.2 shows performance as measured by a suite of benchmarks published by BYTE magazine *(Benchmarking UNIX Systems*, August 1984) designed to measure:

* Pipe implementation and efficiency (pipes)
* Kernel overhead involved in executing a system call (scall)
* User function overhead (fcalla, fcalle)
* CPU speed and C compiler efficiency (sieve)
* Disc throughput (disc write and disc read a 512x256 byte file)
* General purpose shell (sh tst.sh)
* Multitasking with a variable number of background processes (sh tst.sh 1-6)
* Long integer increments (loop)

Table 4.2. BYTE Benchmarks (Elapsed Seconds)

Test	825S	840S	850S
pipes	2.1	1.1	0.8
scall	2.7	1.7	1.3
fcalla	0.1	0.1	0.1
fcalle	0.1	0.1	0.1
sieve	0.1	0.19	0.1
dwrite	0.9	1.0	0.6
dread	0.2	0.5	0.1
sh tst.sh	3.4	3.1	2.7
sh multi.sh 1	3.7	3.3	2.9
sh multi.sh 1 2	6.6	6.4	5.9
sh multi.sh 1 2 3	10.0	9.6	8.5
sh multi.sh 1 2 3 4	13.0	12.5	11.3
sh multi.sh 1 2 3 4 5	16.0	16.7	13.8
sh multi.sh 1 2 3 4 5 6	20.1	19.8	18.2
loop	0.3	0.3	0.2

Kernel Micro Benchmarks

This collection of benchmarks is designed to measure various system operations in UNIX Systems. They are described in *Measuring and Improving the Performance of 4.2BSD* by Sam Leffler, et. al., 1984. Below is a summary of the tests. Results are in Table 4.3 on page 14.

syscall: perform 100,000 getpid system calls
csw: perform 100,000 context switches using signals
signocsw: send 10,000 signals to self
pipeself4: send 10,000 4-byte messages to self
pipeself512: send 10,000 512-byte messages to self
pipediscard4: send 10,000 4-byte messages to a child, who discards
pipediscard512: send 10,000 512-byte messages to a child, who discards
pipeback4: exchange 10,000 4-byte messages with a child
pipeback512: exchange 10,000 512-byte messages with a child
forks0: fork-exit-wait 1K times
forks1K: sbrk(1024), fault pages, fork-exit-wait 1K times
forks100k: sbrk(102400), fault pages, fork-exit-wait 1K times
vforks0: vfork-exit-wait 1K times
vforks1K: sbrk(1024), fault pages, vfork-exit-wait 1K times
vforks100K: sbrk(102400), fault pages, vfork-exit-wait 1K times
execs0null: fork-exec "null job"-exit-wait 1K times
execs0null (1K env): execs0null above, but 1K environment added
execs1Knull: sbrk(1024), fault pages, fork-exec "null job"-exit-wait 1K times
execs1Knull (1K env): execs1Knull above, but 1K environment added
execs100Knull: sbrk(102400), fault pages, fork-exec "null job"-exit-wait 1K times
vexecs0null: vfork-exec "null job"-exit-wait 1K times
vexecs1Knull: sbrk(1024), fault pages, vfork-exec "null job"-exit-wait 1K times
vexecs100Knull: sbrk(102400), fault pages, vfork-exec "null job"-exit-wait 1K times
execs0big: fork-exec "big job"-exit-wait 1K times
execs1Kbig: sbrk(1024), fault pages, fork-exec "big job"-exit-wait 1K times
execs100Kbig: sbrk(102400), fault pages, fork-exec "big job"-exit-wait 1K times
vexecs0big: vfork-exec "big job"-exit-wait 1K times
vexecs1Kbig: sbrk(1024), fault pages, vfork-exec "big job"-exit-wait 1K times
vexecs100Kbig: sbrk(102400), fault pages, vfork-exec "big job"-exit-wait 1K times

The **System Call** benchmark (also in AIM Suite II) provides a quantitative measure of the performance of the system call mechanism.

The **Context Switch** tests measure how fast the system can switch from one process to another. One process signals another, then waits to be signalled back. The second process receives the signal, then signals back. Twice the result for the one process case (*signocsw*) can be deducted from the two process case (*csw*) to get the time spent in the scheduler.

Pipes are buffered by the OS. Writes to a pipe whose buffer space has been exhausted will block until some data has been read out of the pipe. This leaves space for the pending write, which may then complete normally. Each test is run with two different message sizes to determine if message size has an effect on system throughput. *Pipeself* reads its own messages to exclude the overhead of context switching and rescheduling from the measurement results. In *pipeback*, savings normally realized by OS buffering are lost, since the write/wait sequence forces a context switch. *Pipediscard* attempts to measure the efficiency of buffering by putting a fast reader at the pipe's receiving end.

The **Exec** tests use two different jobs to gauge the cost of overlaying a larger one with a smaller one and vice versa. The *null job* and the *big job* programs differ only in the size of their data segments, 1 KByte versus 256 KBytes. In all cases the text segment of the parent is larger than that of the child.

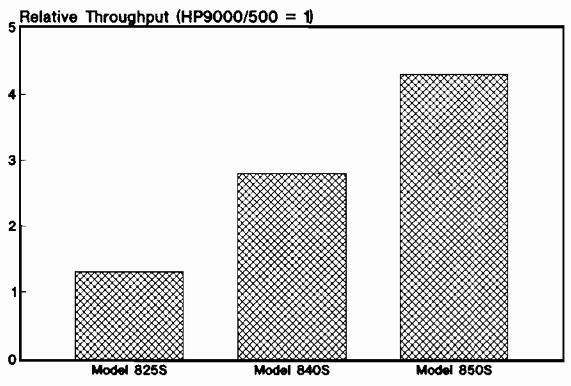
I/O Call Benchmark

This benchmark tests the speed of HP-UX system call interface and the CPU doing common UNIX system I/O systems calls. The benchmark performs 1000 iterations of the following: create a file, close it, open it back up, do a 500 byte write and three 100 byte reads using *lseek* to return to the beginning of the file before each write and each set of reads. Elapsed time for the systems are as follows: Model 825S -- 4.0 seconds; Model 840S -- 1.9 seconds; Model 850S -- 1.2 seconds.

Table 4.3. Kernel Micro Benchmarks (Elapsed Minutes)

Test	825S	840S	850S
syscall	0:09	0:07	0:05
csw	0:48	0:25	0:17
signocsw	0:18	0:10	0:07
pipeself4	0:15	0:07	0:04
pipeself512	0:16	0:08	0:05
pipediscard4	0:13	0:07	0:05
pipediscard512	0:17	0:08	0:05
pipeback4	0:43	0:19	0:13
pipeback512	0:50	0:22	0:14
forks0	0:18	0:11	0:08
forks1K	0:18	0:10	0:07
forks100K	1:05	0:50	0:39
vforks0	0:13	0:08	0:05
vforks1K	0:14	0:07	0:05
vforks100K	0:13	0:08	0:05
execs0null	1:08	0:49	0:45
execs0null (1K env)	1:13	1:07	0:45
execs1Knull	1:07	0:45	0:45
execs1Knull (1K env)	1:10	1:07	0:44
execs100Knull	1:52	1:29	1:07
vexecs0null	1:07	0:45	0:38
vexecs1Knull	1:10	0:50	0:39
vexecs100Knull	1:06	0:45	0:38
execs0big	1:07	0:45	0:48
execs1Kbig	1:07	0:45	0:46
execs100Kbig	1:52	1:30	1:29
vexecs0big	1:07	0:45	0:45
vexecs1Kbig	1:07	0:50	0:45
vexecs100Kbig	1:08	0:48	0:45

Figure 4.8. I/O Call Benchmark



UNIX System Benchmark

This benchmark, taken from the net, measures activities commonly done during program development in a UNIX system environment. The results from Table 4.4 are graphed in Figure 4.7. The C program used (bench.c) in the benchmark is as follows:

```
#include <stdio.h>
#define COUNT 30
#define r(x) ("qwertyuiopasdfghjklzxcvbnm" \
    [pos(x)%26])
#define pos(x) ((x)>0 ? (x) : 5-(x))

main()
{
  int i, j, k;
  for (i=0; i<COUNT; ++i)
  {
    for (j=0; j<COUNT; ++j)
    {
      for (k=0; k<COUNT; ++k)
        printf("%c%c%c%c%c%c\n",r(k),\
              r(j+k), r(j-k), r(i+k), r(i-k),\
              r(i+j+k));
    }
  }
}
```

The benchmark consists of timing (/bin/time) these commands:

```
cc -O bench.c
a.out > foo
cat foo > /dev/null
cat foo > bar
rm bar a.out bench.c
cat foo | cat > /dev/null
cat foo | cat | cat | cat | cat >/dev/null
sort foo > /dev/null
ex foo <<'EOF' > /dev/null
nroff -me /dev/null
nroff -ms /dev/null
nroff -man /dev/null
wc foo
```

Figure 4.7. UNIX System Benchmark

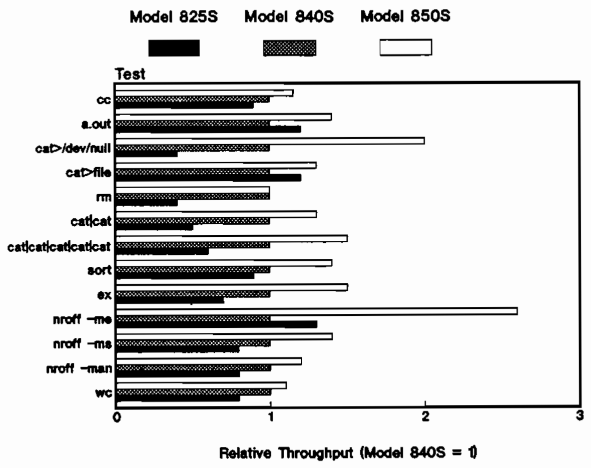


Table 4.4. UNIX System Benchmark (Elapsed Seconds)

Test	825S	840S	850S
cc	6.2	4.7	4.0
a.out	4.6	5.5	3.8
cat > /dev/null	0.4	0.2	0.1
cat > file	1.0	1.2	0.9
rm	1.5	0.4	0.4
cat | cat	0.8	0.4	0.3
cat | cat | cat | cat | cat	2.1	1.2	0.8
sort	7.1	6.5	4.4
ex	6.4	4.8	3.2
nroff -me	3.5	4.5	1.8
nroff -ms	2.5	2.1	1.9
nroff -man	1.6	1.3	1.2
wc	1.0	0.8	0.7

Section 5: HP-UX Real-Time Performance

When Responding to Real-Time Events Is Critical

This section illustrates the real-time capabilities of the Series 800 systems. These measurements contrast the real-time performance of the HP 1000/A900, and HP 9000 Models 825S and 840S. Where available, results for the HP 9000 Model 850 and HP 1000/A600 are included. HP-UX is unique among UNIX systems in providing fast, dependable real-time response. The data reflects measurement techniques that are to a great degree independent of the application, yet convey meaningful information about the system.

The four groups of measurements that will be discussed are: Interrupt Response Performance, I/O Performance, Real-Time Services and Scheduling, and Disc I/O Performance.

HP-UX Real-Time Features

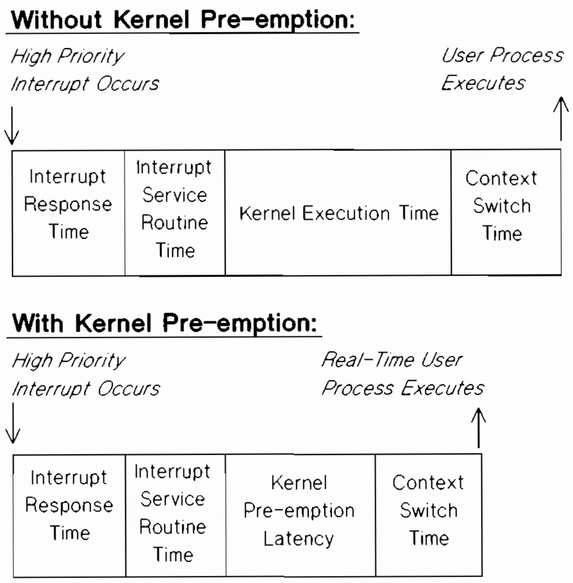
A key measure of real-time operating system performance is how quickly a waiting process can be dispatched in response to some event (such as I/O completion). This "process dispatch latency" includes I/O driver interrupt response time, kernel execution time, and context switch time.

I/O driver interrupt response refers to the time it takes for a driver to service the data on an I/O interface card. If that driver schedules a program to process the data, the time before that program begins execution is the kernel execution time.

In RTE, when an I/O interrupt arrives during operating system ("kernel") execution, the kernel routine will complete execution before executing the driver to service that interrupt. RTE users must write a "privileged driver" to shorten this I/O interrupt response time and are then guaranteed I/O interrupt response times in the 100 microsecond range.

In UNIX systems, all I/O drivers are "privileged" in the RTE sense, because they run with greater importance than kernel routines. Series 800 HP-UX "kernel preemption" allows a real-time priority user process to suspend kernel routines (kernel execution time) and schedule the real-time process. Figure 5.1 shows the elements of process dispatch latency.

Figure 5.1. Process Dispatch Latency



Without Kernel Pre-emption:
High Priority Interrupt Occurs
User Process Executes
Interrupt Response Time
Interrupt Service Routine Time
Kernel Execution Time
Context Switch Time
With Kernel Pre-emption:
High Priority Interrupt Occurs
Real-Time User Process Executes
Interrupt Response Time
Interrupt Service Routine Time
Kernel Pre-emption Latency
Context Switch Time

The tests in this section perform measurements on these and other areas of real-time response.

Noteworthy HP-UX real-time features include:

* *Real-time process scheduling*
* *Kernel preemption*
* *Process and file locking*
* *Elimination of linear searches*
* *Powerfail recovery*
* *File space pre-allocation.*

Real-Time Process Scheduling

The HP-UX process scheduler differentiates between real-time and time-shared processes. The scheduler will always dispatch a process with a real-time priority before a process with a time-shared priority. Real-time process priorities are non-changing, unlike time-shared processes. Suspending one process to execute another is referred to as context switching, and is an important area to examine in differentiating between general purpose and real-time operating systems.

For example, a system used in a control application includes a separate "alarm program" which waits idle until an error or "alarm condition" is detected. Automated applications dictate that this alarm program be kept ready to run at all times, with a minimum of delays. Upon detection of an alarm condition, the operating system must cease executing any lower priority program and context switch to this alarm program. Control over delays caused by current activity, program swapping to disc, dispatch latency, etc., requires that a real-time operating system be used.

Kernel Preemption

In traditional UNIX systems a process executing its own "user" code can be preempted immediately, but if the kernel is executing in behalf of a process, such as when the user process makes a system call, that process surrenders the CPU only voluntarily (e.g. to sleep for a resource). Thus the kernel may execute for a significant period of time before giving the CPU to another process. This period of time is called "preemption latency" and can be unacceptable in a real-time system. HP has greatly reduced this time by enabling a user process (a real-time priority process only) to preempt the kernel.

Process Locking

This feature prevents paging or swapping of a process, so it can be guaranteed immediate execution when it becomes runnable.

File Locking

A region of a file or the entire file may be locked. Region locking of a file is "advisory" -- that is, it is accomplished with semaphores. If the "enforcement" mode of the (entire) file is set by a process, all other processes attempting to access any part of the file will return an error or wait.

Elimination of Linear Searches

Time consuming linear table searches in the kernel (e.g. proc table) were replaced by hash table lookup methods. This reduces kernel preemption latency even further by speeding up the kernel.

Powerfail recovery

When HP-UX detects a power failure, the CPU state and cache data are flushed out to battery backed-up memory. If power is restored within about 15 minutes, I/O devices are reset, I/O transactions going on at the time of failure are restarted, CPU and cache states are restored, and a signal is sent to every process informing it of the power failure.

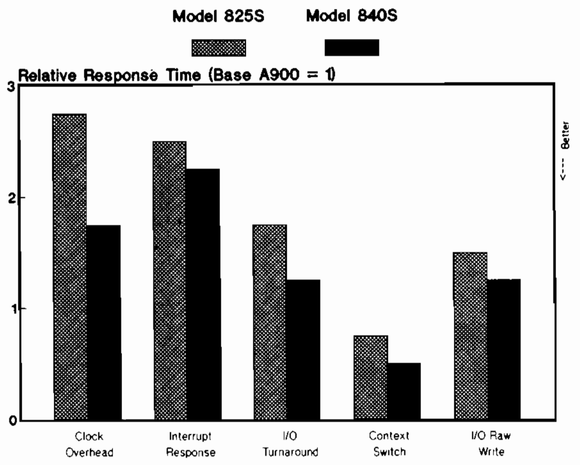
File space pre-allocation

In standard UNIX systems, file system blocks are allocated dynamically for every write operation. HP-UX can pre-allocate file system space for real-time applications to avoid this overhead during activities such as high speed continuous data acquisition.

Real-Time Response Summary

Figure 5.2 is based on Table 5.1 (page 19).

Figure 5.2. Real-Time Response Summary



The summary metrics below are derived using weighted averages (lower numbers mean better performance). The 17 factor ratio used is obtained by averaging the results of the tests within each category and then averaging the means of the five categories. This ratio represents an attempt to characterize the overall real-time performance of the Series 800 systems when compared to the HP 1000 A900.

Table 5.1. Real-Time Response Summary

Response: Ratios to an A900			
DSD Test Num.	**Weight**	**825S**	**840S**
Clock Overhead	0.2	1.69	1.73
Interrupt Resp.			
#1b-to Driver	0.0667	2.85*	2.67
#1c-to User	0.0667	2.91*	1.84
#2 -Latency	0.0667	2.05*	2.67
I/O Turnaround (Driver)			
#3 -HPIB	0.0667	2.82	1.44
#4 -PIC/AFI	0.0667	2.27	1.7
#5 -DISC	0.0667	0.57	0.23
Context Switch			
#7 -Sem. Clear	0.0667	0.68	0.37
#8 -Mess. Pass (1 byte)	0.0667	0.54	0.32
#9 -Mess. Pass (80 bytes)	0.0667	0.54	0.37
I/O Turnaround (User)			
#10-HPIB (1B)	0.0286	2.76	1.6
#11-HPIB (80B)	0.0286	2.58	1.57
#12-PIC/AFI (1B)	0.0286	1.45	0.68
#13-PIC/AFI (80B)	0.0286	1.48	0.83
#14-DISC (512B)	0.0286	1.00	1.00
#15-DISC (8KB)	0.0286	1.00	1.00
#16-DISC (32KB)	0.0286	1.00	1.00
Summary	1.00	1.68	1.34

* The Model 825S Interrupt Response Latency estimates were made by taking a Model 840S trace and feeding it through the simulator for the Model 825S.

Interrupt Response Performance

Interrupt response of HP-UX is a measure of the time required to receive an interrupt from the I/O device, cease the current activity, and enter the I/O driver or the user code to acknowledge the interrupt just received.

Maximum Time To Driver

This test, (see Table 5.2 on page 20) is designed to measure the time required for the system to recognize and begin to process an external interrupt.

HP 1000 test notes: The system workload included nine programs that were used to force swapping and cause memory fragmentation. Virtual Memory Array (VMA) reads and writes were executed, 20 to 30% of System Available Memory (SAM) was randomly used, and the spooling system was stressed. A simple program was written to perform EXEC reads from a parallel interface (HP 12006). A signal generator was used to signal the device flag (DVFLG) handshake. A logic analyzer (HP 64000) was used to measure time from the interrupt request signal (INTRQ) to the entry into the interface driver (ID.50).

Series 800 Test notes: This test was run on a busy system. Measurements were taken for four levels of "busy":

Level 0: No workloads on the system.
Level 1: The program Development Script (see pg. 3) with 16 users.
Level 2: Level 1 plus a program writing continuously to an infinitely fast HPIB instrument.
Level 3: Level 2 plus a tape archive program (tar) continuously writing to tape.
Level 4: Level 3 plus continuous network file transfer.

A program was written to make one byte reads to a parallel (AFI) interface. A special test device was used to signal the read completion AFI interface. A logic analyzer was used to measure the time from the external signal to the execution of the first instruction in the device adapter manager (DAM).

Table 5.2. Maximum To Driver (Test #1b)

System	Min (us)	Max (ms)	Median (us)	Mean (us)
1000/A900	82.0	2.589	104	129
Model 825S				
Level 0	87.0*	0.237*	172*	NA
Level 3	237*	1.162*	296*	NA
Model 840S				
Level 0	173	0.379	195	198
Level 1	191	1.266	267	278
Level 2	123	1.155	263	291
Level 3	140	1.632	278	303
Level 4	110	1.688	269	311

* These estimates were made from tracing the Model 840S then feeding results through the simulator for the Model 825S.

Maximum Time To User

This test is designed to measure the time required for the system to recognize an interrupt and schedule the process waiting for that event. See Table 5.3 for results.

The same test notes apply for this test as in the "Maximum to Driver" test except the logic analyzer was used to measure the time the flag that causes a DMA completion interrupt was set to the execution of the first instruction of the user's code. The user was running at real-time priority 0 (the most important).

Interrupt Response Latency

Interrupt response latency was measured between trap cell execution and execution of the first line of the interface (physical) driver. Table 5.4 shows the results.

HP 1000 test notes: The measurement was taken on a busy system to maximize the OS overhead time, i.e. the driver was in a driver partition, the driver was not mapped into the driver partition, and the DVT/IFT in $VCTR was not already pointing at the DVT/IFT for the device.

Table 5.3. Maximum To User (Test #1c)

System	Min (us)	Max (ms)	Median (us)	Mean (us)
1000/A900	576	5.99	635	710
Model 825S				
Level 0	912*	2.665*	1053*	NA
Level 3	1132*	5.905*	1848*	NA
Model 840S				
Level 0	655	1.991	736	774
Level 1	686	4.989	1187	1278
Level 2	776	3.560	1188	1288
Level 3	791	8.039	1170	1298
Level 4	791	9.147	1198	1305

* These estimates were made from tracing the Model 840S then feeding results through the simulator for the Model 825S.

Table 5.4. Interrupt Response Latency (Test #2)

System	Min (us)	Max (us)	Median (us)	Mean (us)
1000/A600	157	213	175	181
1000/A900	70	131	84	88
Model 825S	87*	237*	172*	NA
Model 840S	206	401	225	227

* These estimates were made from tracing the Model 840S then feeding results through the simulator for the Model 825S.

Series 800 test notes: The measurement is the time from interrupt acknowledgement to Device Adapter Manager (DAM) entry. Interrupt acknowledgement means the CPU is physically interrupted and vectors through a trap cell (or interrupt vector). The logic analyzer measurement was taken in two places -- the time from interrupt to identifying the external interrupt, and the time from the external interrupt recognition to entry to the DAM. The time reported is the sum of the two measurements.

I/O Turnaround (Driver)

I/O turnaround is the time from driver entry due to an I/O completion to initiation of the next I/O request in a situation where further requests are pending. This number varies from driver to driver and is measured for the disc, HPIB, and 16-bit parallel interface drivers. These numbers give an indication of the driver architecture and I/O system efficiency. The average value indicates what system overhead to expect on a per-I/O basis. The extreme values show the bound on continuous data acquisition (as from a synchronous device). (See Tables 5.5, 5.6, and 5.7.)

HP 1000 test notes: The test program issues multiple Class I/O writes to the device. To ensure that the I/O request does not complete before the next can get queued a buffer length of 1000 bytes was used. The test was run using both buffered and unbuffered Class I/O. The timings were the same for buffered and unbuffered Class I/O; therefore only one number is reported. The measurements start with the execution of the first line of the driver and stop with the execution of STC 2 (start of DMA).

Series 800 test notes: In order to assure multiple pending requests, a process will be created that forks itself multiple times and proceeds to post requests against the device of interest. The logic analyzer measured the time from the execution of the first line of the DAM to the start of the next DMA request in the Channel Adapter Manager (CAM).

Table 5.5. HPIB To Nonexistent Instrument (Test #3)

System	Min (ms)	Max (ms)	Median (ms)	Mean (ms)
1000/A600	1.988	2.074	1.989	1.996
1000/A900	0.968	1.034	0.977	0.977
Model 825S	2.448	4.911	2.757	2.929
Model 840S	1.211	6.84	1.41	1.497

Table 5.6. PIC/AFI Card With Loopback (Test #4)

System	Min (us)	Max (us)	Median (us)	Mean (us)
1000/A600	NA	NA	NA	NA
1000/A900	692	805	700	701
Model 825S	1459	7081	1586	1840
Model 840S	736	6172	1195	1198

Table 5.7. DISC (7935) (Test #5)

System	Min (ms)	Max (ms)	Median (ms)	Mean (ms)
1000/A600	7.625	9.679	8.528	8.234
1000/A900	3.798	7.391	5.703	5.192
Model 825S	2.93	14.25	3.27	3.75
Model 840S	0.914	8.718	1.317	1.485

Semaphore Clear, Context Switch

Context switching in general requires three actions in most multitasking operating systems: Process the event which is causing the context switch, suspend the currently executing process, and begin executing the higher priority process.

Releasing a semaphore or sending a message from a lower priority process to a higher priority process illustrates the time required to perform all three steps of a context switch.

This test measures the amount of OS overhead involved in dispatching a process suspended on a locked semaphore -- the time required to process the semaphore unlock request and to dispatch the suspended program once that semaphore is unlocked. This measurement is made with three constraints: no external interrupts (other than Time Base Generator); the process is already in memory; the process to be redispatched is the highest priority process. The logic analyzer measured the time from the execution of the call (in the user process) to the OS to unlock the semaphore to the execution of the first line of code in the suspended process. Results are in Table 5.8 (page 22).

Table 5.8. Semaphore Clear With Context Switch (Test #7)

System	Min (us)	Max (us)	Median (us)	Mean (us)
1000/A600	1872	2018	1875	1898
1000/A900	901.2	993	905	912
Model 825S	573	2171	334	378
Model 840S	334	1279	386	411

Message Passing With Context Switch

These tests measure the amount of OS overhead required to dispatch a process suspended waiting for a message. Messages of a single byte (Table 5.9), and of 80 bytes (Table 5.10) are reported. This measurement is made with three constraints: no external interrupts (other than Time Base Generator); both processes are already in memory; the process waiting for the message is the highest priority process. The logic analyzer measured the time from the call to the OS to send the message (in the user process) to the execution of the first line of code in the suspended process.

Table 5.9. Single Byte Message Passing Time (Test #8)

System	Min (ms)	Max (ms)	Median (ms)	Mean (ms)
1000/A600	3.063	3.222	3.068	3.122
1000/A900	1.647	1.749	1.656	1.67
Model 825S	0.813	3.22	0.887	1.053
Model 840S	0.531	1.479	0.534	0.556

Table 5.10. 80 Byte Message Passing Time (Test #9)

System	Min (ms)	Max (ms)	Median (ms)	Mean (ms)
1000/A600	3.245	3.405	3.251	3.292
1000/A900	1.767	1.868	1.776	1.79
Model 825S	0.897	1.583	0.952	1.048
Model 840S	0.661	1.91	0.664	0.69

I/O Turnaround (User)

These tests (Tables 5.11 through 5.17) measure the time it takes for an I/O request to be processed as seen from the user program, from user I/O request to the next line of code in the user's application.

The measurements were made under these four constraints: The program doing the I/O was the only program executing; no other external interrupts occurred (except Time Base Generator); multiple tests were run for a fixed set of buffer sizes; dummy devices were used on the HPIB and PIC tests to simulate a very high speed device.

HP 1000 test notes: With RTE-A there are two forms of I/O, buffered and unbuffered. Buffered I/O allows the user program to continue executing while the driver is processing the I/O request. Unbuffered I/O forces the user program to wait until the I/O is complete before it can continue execution. Each test was run using both buffered and unbuffered I/O, except the "raw" disc tests. They were only run in an unbuffered mode, which is the normal mode in RTE-A. The buffered and unbuffered times were the same when the measurement was taken; therefore, only one number is reported.

Series 800 test notes: The logic analyzer measured the time from the start of the write request in the user process to the execution of the next line of code in the user program. In HP-UX there is no direct equivalent to buffered I/O in RTE. Buffering, if available, is driver dependent. It is more of a blocking scheme than a buffering scheme. None of the drivers used in these tests support any form of buffering. Therefore, only unbuffered writes were measured. All writes were to raw devices.

Tables 5.15, 5.16, and 5.17 show "raw" disc writes, i.e. measure direct I/O to the disc drive without going through the file system. Many real-time applications use direct disc I/O for writing data directly to the disc and avoiding the overhead of the file system. When a large amount of data is to be managed at high speeds and must be stored on disc for later analysis, direct disc I/O is useful. In writing to the raw device, the programmer takes responsibility for data format, location, directories, and benefits from the speed and predictability of direct disc I/O. Disc optimization techniques such as disc sector skipping would result in greater performance than reported in this area. We have observed throughput measures of 700 KBytes per second for large (64 KBytes) array writes for the Model 840S.

For these "raw" I/O tests, the head was prepositioned to eliminate the seek overhead and sufficient sequential writes were performed to fill the analyzer.

Table 5.11. HPIB Instrument: 1 Byte (Test #10)

System	Min (ms)	Max (ms)	Median (ms)	Mean (ms)
1000/A600	3.389	3.703	3.395	3.444
1000/A900	1.625	1.833	1.627	1.641
Model 825S	4.16	5.10	4.497	4.557
Model 840S	2.563	3.57	2.61	2.689

Table 5.12. HPIB Instrument: 80 Bytes (Test #11)

System	Min (ms)	Max (ms)	Median (ms)	Mean (ms)
1000/A600	3.59	3.90	3.599	3.651
1000/A900	1.717	1.934	1.719	1.734
Model 825S	4.222	7.534	4.435	4.681
Model 840S	2.65	3.653	2.70	2.737

Table 5.13 PIC/AFI: 1 Byte (Test #12)

System	Min (ms)	Max (ms)	Median (ms)	Mean (ms)
1000/A600	3.101	3.335	3.121	3.155
1000/A900	1.435	1.603	1.437	1.449
Model 825S	2.003	3.482	2.09	2.269
Model 840S	0.945	1.982	0.971	1.013

Table 5.14. PIC/AFI: 80 Bytes (Test #13)

System	Min (ms)	Max (ms)	Median (ms)	Mean (ms)
1000/A600	3.117	3.335	3.121	3.168
1000/A900	1.449	1.62	1.452	1.463
Model 825S	2.06	3.93	2.156	2.348
Model 840S	1.058	2.197	1.206	1.255

Table 5.15. Disc (7935): 512 Bytes (Test #14)

System	Min (ms)	Max (ms)	Median (ms)	Mean (ms)
1000/A600	22.49	29.12	22.72	22.82
1000/A900	22.61	29.09	22.73	22.83
Model 825S	19.09	31.81	22.52	22.82
Model 840S	21.97	27.53	22.42	22.82
Model 850S	16.42	27.27	22.6	22.58

Table 5.16. Disc (7935): 8 KBytes (Test #15)

System	Min (ms)	Max (ms)	Median (ms)	Mean (ms)
1000/A600	29.71	36.5	29.9	31.45
1000/A900	29.81	36.38	29.92	31.45
Model 825S	28.77	39.1	29.7	31.17
Model 840S	29.67	27.53	30	31.45
Model 850S	27.17	36.58	29.8	31.25

Table 5.17. Disc (7935): 32 KBytes (Test #16)

System	Min (ms)	Max (ms)	Median (ms)	Mean (ms)
1000/A600	56.96	63.78	57.31	59.06
1000/A900	57.12	63.73	57.23	59.09
Model 825S	53.68	89.94	57.84	59.04
Model 840S	55.99	63.92	57.40	59.13
Model 850S	54.44	76.57	57.57	59.11

File System Read/Write Times

HP-UX provides a fast file system with performance improvements attained through the use of disc caching, optimized disc layout, and large block sizes.

For real-time applications, disc I/O performance can be characterized in two ways; file system and direct disc I/O. File system performance is key to achieving a higher level of system performance for most real-time applications, such as manufacturing area management systems, large communications applications, process control, and OEM real-time systems,

Tables 5.18 through 5.29 measure sequential reads and writes, and random reads and writes to the File System. The HP-UX File System gains efficiency by keeping frequently accessed file blocks in the disc cache. Additionally, file system writes are performed asynchronously which allows the user's process to continue computational tasks in parallel.

Both sequential read and write measures show the speed difference due to disc caching. The 512 byte and 8 KByte writes are performed in disc cache, while the 32 KByte transfers require a disc access every time the disc cache is filled. This occasional disc access accounts for the significant difference for the 32 KByte writes.

Random file system measurement shows similar results. The use of a large disc cache increases the probability that the desired segment of the file is in the cache, thus eliminating a disc access.

These tests are designed to measure the amount of time required for a data space on disc to be read or written to. In real-time applications users would tend to open the file (and leave it open) prior to actual data gathering. Consequently, the overhead required to open the file was ignored. For standardization the following constraints were used: Only the program doing the I/O was active; no other external interrupts occurred (except Time Base Generator); a clean file system was used in which a file of the required size was preallocated; all random tests used the same predetermined sequence of random numbers for both RTE and HP-UX.

HP 1000 test notes: All tests use an 8 KByte buffer (Data Control Block). The random read test was performed using an 8 KByte and also using an 8 KByte plus one disc block DCB. This measured the impact of an RTE-A file system attribute on access time. The system time was taken, 100 operations were performed, the last buffer was posted to disc on write tests, and the system time was retaken.

In RTE Type 2 (see note) files are always in update mode. Thus if the block containing the record is not in the DCB a read will be done to get the block into the DCB before the data is moved from the users buffer into the DCB. When the record size is the same or larger than the DCB size, two or more disc accesses will be performed for every record. Every record in a type 3 (see note) file is 2 words longer than the user requested record length. These two words are appended to every record by the file system. As a result, the RTE-A Type 3 file test programs did not allocate a file large enough to hold all the records, thereby causing an extent to be created.

FILE TYPES: *RTE distinguishes between file types, while HP-UX does not. The HP-UX file system considers all ascii, data, and executable files to be "byte streams". RTE type 1 and 2 files are fixed length record files, all other file types are variable length record files.*

Series 800 test notes: All tests were done to a file set up on a disc partitioned with an 8 KByte block size. Buffer sizes of 512 bytes, 10 KByte bytes and 32 KBytes were used; one hundred sequential or random read or write operations were performed from the user process. The *fsync* time for write benchmarks is reported separately; for random file access, an *lseek* call was used to position to the proper location in the file. In this case, the time measured was from the lseek call to the completion of the read/write call.

Sequential File System Writes

Table 5.18. 512 Byte Buffer (Test #17)

(To Type 2 File:)

System	Min (us)	Max (ms)	Median (us)	Mean (ms)
1000/A600	923.6	34.78	926.3	4.703
1000/A900	348.8	40.97	385.5	4.696

(To Type 3 File:)

System	Min (us)	Max (ms)	Median (us)	Mean (ms)
1000/A600	1300	93.07	1304	4.866
1000/A900	453.8	95.73	487.6	3.582
Model 825S	882	57.37	979	4.073
Model 840S	447	51.59	469	3.324
Model 850S	76	53.61	492	3.825

fsync following 100 512 byte writes on the Model 840S: 375.7 ms

Table 5.19. 8 KByte Buffer (Test #18)

(To Type 2 File:)

System	Min (ms)	Max (ms)	Median (ms)	Mean (ms)
1000/A600	52.01	80.9	52.33	60.51
1000/A900	48.41	82.46	75.05	70.59

(To Type 3 File:)

System	Min (ms)	Max (ms)	Median (ms)	Mean (ms)
1000/A600	29.92	132.2	29.94	32.81
1000/A900	29.8	130.0	29.95	32.59
Model 825S	4.68	190.2	5.9	9.03
Model 840S	2.2	239	2.8	5.4
Model 850S	1.14	251.4	1.8	4.59

fsync following 100 8 KBytes writes on the Model 840S: 1.7 sec

Table 5.20. 32 KByte Buffer (Test #19)

(To Type 2 File:)

System	Min (ms)	Max (ms)	Median (ms)	Mean (ms)
1000/A600	213.1	264.1	235.5	242.0
1000/A900	257.7	308.6	280.5	281.6

(To Type 3 File:)

System	Min (ms)	Max (ms)	Median (ms)	Mean (ms)
1000/A600	46.74	223.6	124.2	126.1
1000/A900	50.02	221.2	124.2	126.1
Model 825S	106.9	200.5	118.4	119.1
Model 840S	95.5	178	97.3	103.8
Model 850S	81.8	230.1	98.5	107.4

fsync following 100 32 KBytes writes on the Model 840S: 364.4 ms

Sequential File System Reads

Table 5.21. 512 Byte Buffer (Test #20)

(To Type 2 File:)

System	Min (us)	Max (ms)	Median (us)	Mean (ms)
1000/A600	919.7	41.56	920.4	3.037
1000/A900	365.2	27.77	399.6	1.849

(To Type 3 File:)

System	Min (us)	Max (ms)	Median (us)	Mean (ms)
1000/A600	952.0	51.05	955.2	3.752
1000/A900	380.1	32.53	406.4	2.178
Model 825S	720	44.65	805	1.445
Model 840S	333	36.24	358	1.344
Model 850S	300	32.97	426	1.388

Table 5.22. 8 KByte Buffer
(Test #21)

(To Type 2 File:)

System	Min (ms)	Max (ms)	Median (ms)	Mean (ms)
1000/A600	29.91	36.53	29.94	31.5
1000/A900	29.91	36.18	30.47	31.5

(To Type 3 File:)

System	Min (ms)	Max (ms)	Median (ms)	Mean (ms)
1000/A600	29.73	52.08	29.94	31.68
1000/A900	29.92	55.2	30.48	31.68
Model 825S	11.74	77.88	18.82	19.2
Model 840S	14.4	52.84	19.56	19.41
Model 850S	12.6	51.3	18.95	19.3

Table 5.23. 32 KByte Buffer
(Test #22)

(To Type 2 File:)

System	Min (ms)	Max (ms)	Median (ms)	Mean (ms)
1000/A600	123.8	130.5	124.2	125.9
1000/A900	123.9	130.5	124.8	125.9

(To Type 3 File:)

System	Min (ms)	Max (ms)	Median (ms)	Mean (ms)
1000/A600	123.8	152.5	124.2	126.18
1000/A900	123.9	130.1	124.8	125.92
Model 825S	65.5	205	73.9	77.02
Model 840S	69.19	157.4	74.11	75.05
Model 850S	67.7	156.3	73.7	75.2

Random Access File System Writes

Table 5.24. 512 Byte Buffer
(Test #23)

(To Type 2 File:)

System	Min (us)	Max (ms)	Median (ms)	Mean (ms)
1000/A600	1496	83.65	51.42	47.75
1000/A900	580.4	67.73	48.81	41.74
Model 825S	1085	46.3	1.196	4.4
Model 840S	411	34.22	0.477	2.91
Model 850S	140	39.93	0.542	3.31

fsync following 100 512 byte random file writes on the Model 840S: 476 ms

Table 5.25. 8 KByte Buffer
(Test #24)

(To Type 2 File:)

System	Min (ms)	Max (ms)	Median (ms)	Mean (ms)
1000/A600	42.15	91.8	59.34	63.11
1000/A900	37.57	88.54	75.56	73.24
Model 825S	3.55	720	4.4	25
Model 840S	1.662	536	2.3	23.44
Model 850S	1.3	695.7	2.2	23.9

fsync following 100 8 KBytes random file writes on the Model 840S: 723.6 ms

Table 5.26 32 KByte Buffer
(Test #25)

(To Type 2 File:) System	Min (ms)	Max (ms)	Median (ms)	Mean (ms)
1000/A600	209.0	275.8	241.8	244.2
1000/A900	4.52	316.7	287.7	281.3
Model 825S	4	254	132	133
Model 840S	13	619	122	119
Model 850S	NA	NA	NA	NA

fsync following 100 32 KBytes writes on the Model 840S: 1.85 sec

Random Access File System Reads

Table 5.27 512 Byte Buffer
(Test #26)

(To Type 2 File:) System	Min (us)	Max (ms)	Median (ms)	Mean (ms)
1000/A600	1494	42.25	29.9	28.67
1000/A900	695	39.74	29.0	25.99
Model 825S	859	55.07	0.929	2.867
Model 840S	302	48.78	0.355	1.554
Model 850S	251	53.76	0.477	2.33

Table 5.28 8 KByte Buffer
(Test #27)

(To Type 2 File:) System	Min (ms)	Max (ms)	Median (ms)	Mean (ms)
1000/A600	24.92	52.37	37.46	38.79
1000/A900	22.31	46.79	35.59	34.79
Model 825S	3.194	105.5	39.3	34.36
Model 840S	1.0	45.99	23.7	20.2
Model 850S	0.851	73.75	23.8	20.92

Table 5.29. 32 KByte Buffer
(Test #28)

(To Type 2 File:) System	Min (ms)	Max (ms)	Median (ms)	Mean (ms)
1000/A600	118.9	146.1	133.5	133.98
1000/A900	118.8	145.7	130.5	131.1
Model 825S	15.9	150.9	89.5	84.2
Model 840S	11.6	140.3	92.9	80.0
Model 850S	NA	NA	NA	NA

Section 6: Database Benchmarks

FASTTRACK Benchmark

FASTTRACK is a benchmark developed at HP, used for measuring the performance of various DBMSs. It consists of ten transactions that are run against a simple database. The ten transactions are executed in random order. They are assumed to be representative of typical applications, but the numbers presented here can only provide general indications of relative performance. Accurate performance data can only be measured on an application by application basis.

Table 6.1 shows relative performance of the relational model (HPSQL) DBMS for the Series 800 systems. Table 6.2 shows comparative performance of the network model IMAGE/1000-II on an A900 and the two modes of operation for HPIMAGE on a Model 840S (native and translator -- described below).

The ten transactions can be characterized as follows:

Transactions 1-3 are short read-only transactions.
Transaction 4 is a somewhat longer read-only transaction. A number of b-tree searches are made in the network model. A join between two small tables occurs in HPSQL.
Transaction 5 is a long read-only transaction. Network model: many hashed accesses are made. HPSQL: a larger join between 3 tables occurs.
Transactions 6 and 8 cause updates.
Transaction 7 deletes, then rolls back (undoes) the delete.
Transaction 9 updates, then rolls back (undoes) the update.
Transaction 10 is a fairly long read-only transaction. Mostly serial accesses are made in the network model; all records for one table are returned in HPSQL.

Figure 6.1 summarizes results for Series 800 processors shown in Table 6.1.

Figure 6.1. Series 800 FASTTRACK HPSQL Benchmark

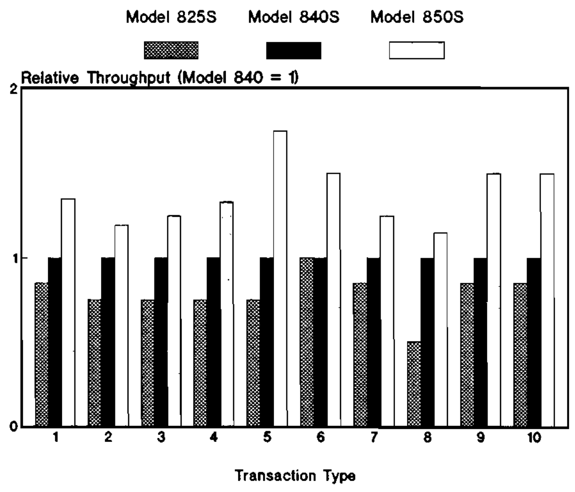


Table 6.2 shows the results of a variety of FASTTRACK runs. The "Translator" is a package that runs on ALLBASE/HP-UX, and which maps IMAGE/1000 calls to HPIMAGE/HP-UX calls for easier migration. The "set locking" and "page locking" refer to different granularities -- the former locks an entire dataset, while the latter locks 4K data pages. Since page locking is the default, somewhat more overhead is involved in dataset locking. Finally, the difference between "optimized" and "unoptimized" IMAGE/1000 is that "unoptimized" is a version of IMAGE/1000 FASTTRACK that includes begin-transaction, end-transaction and lock calls, which are all HPIMAGE/HPUX idioms; the "optimized" version does not include these calls, and is therefore closer to a typical IMAGE/1000 application.

Table 6.1. FASTTRACK HPSQL Benchmark

	Avg. Elapsed Time (millisecs)	Avg. CPU Time (millisecs)
Model 825S		
TRAN 1	225	149
TRAN 2	230	149
TRAN 3	166	143
TRAN 4	372	249
TRAN 5	11883	11650
TRAN 6	315	200
TRAN 7	1332	844
TRAN 8	1126	615
TRAN 9	3969	2741
TRAN 10	283	241
Model 840S		
TRAN 1	189	110
TRAN 2	179	112
TRAN 3	128	109
TRAN 4	339	178
TRAN 5	8951	8678
TRAN 6	313	149
TRAN 7	1068	599
TRAN 8	801	433
TRAN 9	3350	2154
TRAN 10	243	196
Model 850S		
TRAN 1	147	68
TRAN 2	159	72
TRAN 3	101	68
TRAN 4	265	116
TRAN 5	5011	4582
TRAN 6	222	90
TRAN 7	880	386
TRAN 8	788	274
TRAN 9	2410	1231
TRAN 10	174	124

Table 6.2. FASTTRACK IMAGE Benchmark

A900 IMAGE/1000-II (Elapsed Milliseconds)	**Optimized**	**Unoptimized**
TRAN 1	50	120
TRAN 2	80	250
TRAN 3	170	190
TRAN 4	520	540
TRAN 5	48200	48200
TRAN 6	80	190
TRAN 7	1130	1850
TRAN 8	5350	5350
TRAN 9	5520	7760
TRAN 10	1760	1760
840S IMAGE/1000 Translator	**Page Locks**	**Set Locks**
TRAN 1	73	147
TRAN 2	117	216
TRAN 3	192	258
TRAN 4	661	702
TRAN 5	18322	18744
TRAN 6	307	330
TRAN 7	667	627
TRAN 8	4119	4118
TRAN 9	1488	1495
TRAN 10	1299	1418
840S HPIMAGE	**Page Locks**	**Set Locks**
TRAN 1	77	75
TRAN 2	90	133
TRAN 3	183	193
TRAN 4	526	581
TRAN 5	11561	11814
TRAN 6	195	268
TRAN 7	431	546
TRAN 8	2964	3071
TRAN 9	1111	1372
TRAN 10	824	851

Section 7: Data Acquisition Performance

The following benchmarks are designed to measure performance in a data acquisition environment typical in the real-time market. The benchmarks demonstrate time to service external interrupts from instruments, and disc access performance.

YEWCOM Benchmark

This benchmark is based on a YEW (Yokogawa Electric Works) application programs and is part of a typical process control application which makes good use of the real-time operating system (see Figure 7.1). It does not fully simulate this application, but parts of it -- the DB file update and display processes -- are representative of process control applications. In these two areas the YEWCOM benchmark puts a simulated process control workload on the system. Disc and terminal throughput on the system are the measures that YEWCOM makes available to compare A-Series and HP-PA systems.

The programs were written in FTN7X on RTE-A and called several FMP routines and EXEC routines. These programs were executed on RTE-A and then were rewritten in "C" using UNIX system calls and the native HP-UX real-time functions. Figure 7.1 diagrams the benchmark.

There are two parts to the benchmark. The first is a scanning function that reads 3000 data points per minute from simulated input and writes the gathered data into a disc file (32 bytes per point). When the benchmark is run there is one scanning process reading data and storing it in a database. The second part is a display function which is scheduled upon the operators request. It initializes the CRT and sends formatted data to the screen. Multiple display processes can be looking at the database.

Figure 7.1. YEWCOM Benchmark Flow

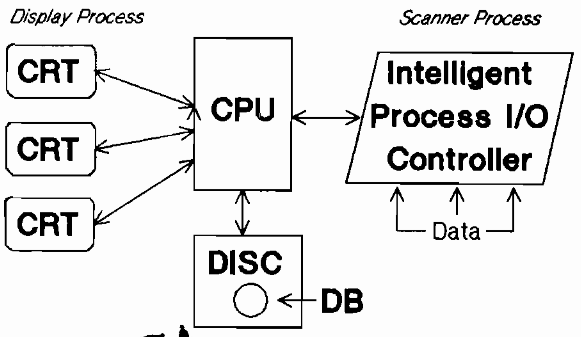


Figure 7.2 graphs the results in table 7.1. Two display transactions are being tracked: "Page 1" causes 7 sequential reads; "Page 5" causes 36 random reads.

Figure 7.2. YEWCOM Benchmark

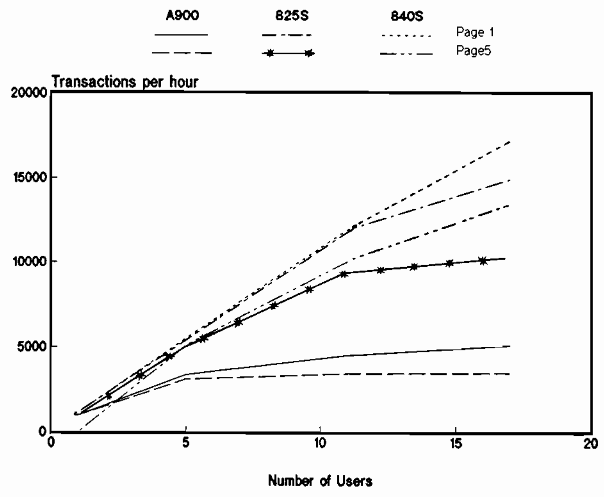


Table 7.1. YEWCOM Benchmark

Trans/hr.	1 User	5 Users	11 Users	17 Users
A900				
Page 1	980	3352	4496	5080
Page 5	976	3092	3416	3448
Model 825S				
Page 1	1132	5500	11908	14900
Page 5	1008	5016	9392	10244
Model 840S				
Page 1	1128	5588	12080	17136
Page 5	1032	5292	11456	13032

Disc Access Performance

These file system tests write to a new file system with a block size of 8 KBytes. The results are specified by disc and amount of system memory.

File System Writes

Three types of file system writes are measured. Each writes a total of 4 MBytes of data in 8 KByte writes (see Table 7.2).

Sequential Write with no fsync is used to measure the time to perform writes by the program. At the end of that time, the buffer cache will contain data of up to 10% of system memory (10% is the default, the buffer cache size is configurable). This data will automatically be posted to disc by a daemon process (syncer). These times will vary from run to run more than the other write tests and will be positively correlated to system memory size.

Sequential Write with fsync: The fsync call is made in the program after all the writes have been done, forcing the blocks in the buffer cache to be posted to disc before the program completes, rather than waiting a few seconds for syncer to do it. Generally this is unnecessary.

Synchronous Sequential Writes: After each write, the program waits until the buffer is actually on the disc before beginning the next write. This wait has substantial negative impact on performance. Fsync is not used with synchronous writes.

Table 7.2. File System Write (KBytes/sec)

File System Write (KB/sec)			
System	**Sequential no fsync**	**Sequential fsync**	**Synchronous Sequential**
Model 825S (40 MBytes)			
HP7935	545	410	195
Model 840S (8 MBytes)			
HP7914	410	385	270
Model 840S (24 MBytes)			
HP7935	636	414	247
Model 850S (40 MBytes			
HP7935	638	417	271

RAW Disc Writes

The raw disc benchmark reported here wrote 4 MBytes in groups of 64 KBytes. Note that raw throughput is not affected by the buffer cache, since raw disc accesses bypass the cache. Unlike the file system writes, the raw disc will only correctly handle file accesses which are a multiple of the block size of the disc (1K).

File System Reads

Two kinds of reads are measured. Again, each read totals 4 MBytes executed in 8 KBytes blocks.

Sequential Read measures the time to perform reads by the program. The buffer cache is not able to help much since we never look at the same block twice. Also the file system read-ahead has little chance to help out because there is no opportunity for overlap, since very little else (such as computation or writing) is being done.

Gaussian Read demonstrates one type of performance speedup provided by the buffer cache. It models accesses to a file where some locations in the file are more likely to be referenced than others. Again, 512 8 KByte reads are done from a 4 MByte file, but they are drawn from a Gaussian distribution about the center of the file. This distribution has a standard deviation of 20 * 8KB = 160KB.

Table 7.3. Raw Disc Write/File System Read (KBytes/sec)

System	Raw Write	Sequential Read	Gaussian Read
Model 825S (40 MBytes			
HP7935	607	422	758
Model 840S (8 MBytes)			
HP7914	710	390	880
Model 840S (24 MBytes)			
HP7935	767	422	978
Model 850S (40 MBytes)			
HP7935	728	423	1029

Section 8: Network Services

This section describes performance of certain Network Services on the HP 9000 Series 800 (NS/9000 Series 800). Performance data is provided for the following situations:

* Network File Transfer (NFT) to and from the HP 9000 Models 840S and 320 and the HP 1000 A900. The NFT file copying program *dscopy* allows you to copy files to and from nodes on your network.

* Remote File Access (RFA) to and from the HP 9000 Models 840S and 320. RFA allows you to access file systems on remote computers as if they were appended to your local file system.

NFT and RFA Performance

Test Measurement

The measurements below (see Figures 8.1 through 8.8) illustrate the elapsed time for each file transfer, and the average throughput per transfer. Some measurements show the CPU percentage used by the producer and consumer of the file.

The HP-UX *time* command measured the elapsed time for each file transfer. For NFT, the *dscopy* program moved the files from the producer system to the consumer system. For RFA, the HP-UX *cp* command copied files from the producer system to the consumer system over an RFA connection.

The HP-UX performance tool VMSTAT measured the CPU utilization. Samples were taken once per second for three minutes of continuous file transfer.

Measurements were taken on Model 840S nodes only.

System Configuration

Each Model 840S had 16 MBytes of RAM and a 7935 disc drive.

Each Model 320 had 4 MBytes of RAM, with 550400 Bytes allocated to networking, and four 7945 disc drives.

Each Model A900 had 3 MBytes of RAM, with 302 KBytes allocated to networking, and a 7914 disc drive.

The only workload on each system was the performance test.

Measurement Summary

Each file transfer was measured in terms of elapsed time per transfer; the 840S to 840S tests also measured the CPU usage during repeated transfers. The set-up time per transfer is the average time required to transfer a 0-byte file. The throughput per file transfer can be determined by dividing the file size by the elapsed time.

Figure 8.1. 840S to 840S NFT Elapsed Time

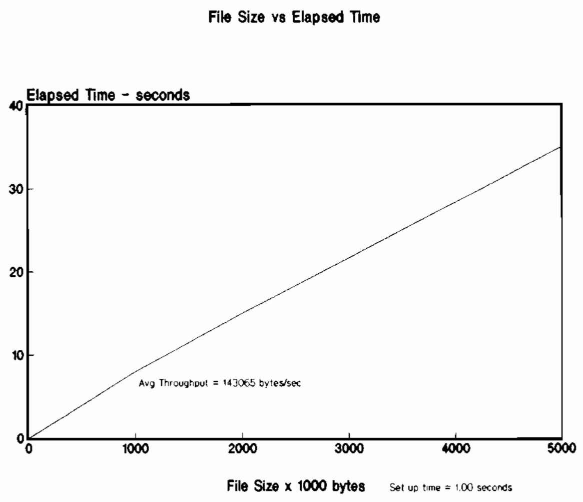


Figure 8.3. 840S to 840S RFA Elapsed Time

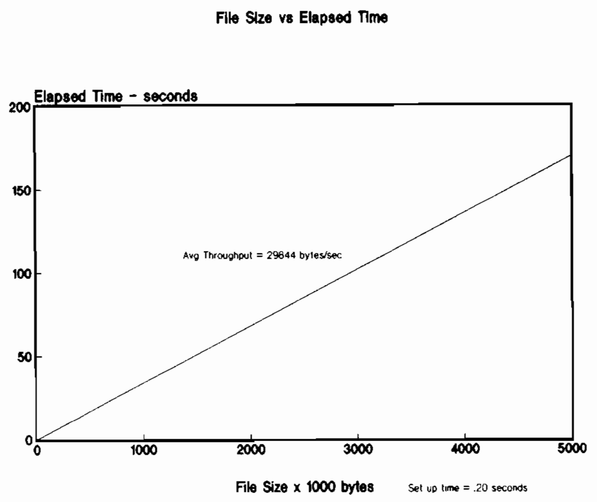


Figure 8.2. 840S to 840S NFT CPU Usage

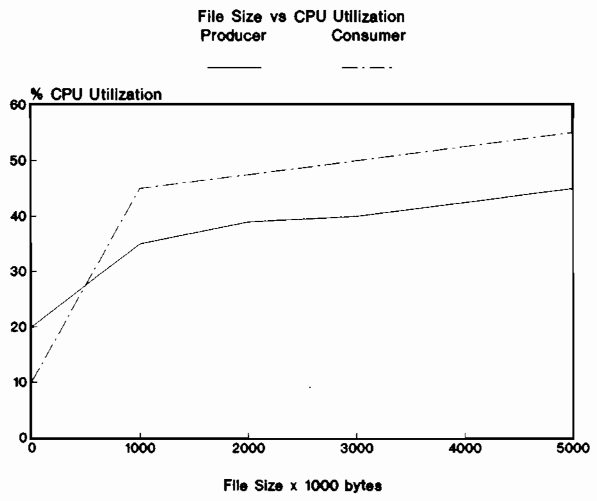


Figure 8.4. 840S to 840S RFA CPU Usage

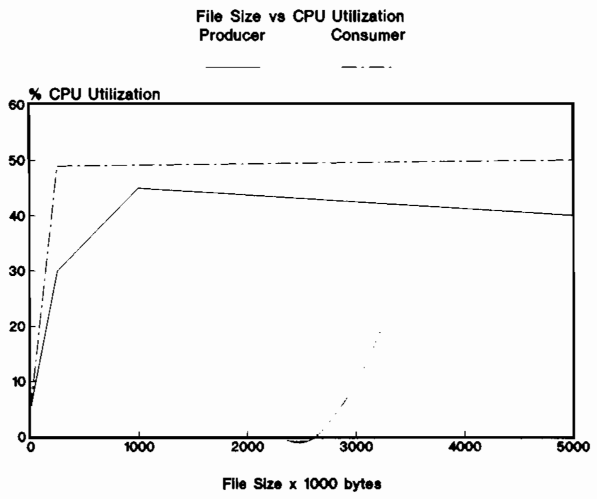


Figure 8.5. 840S to/from 320 NFT Elapsed Time

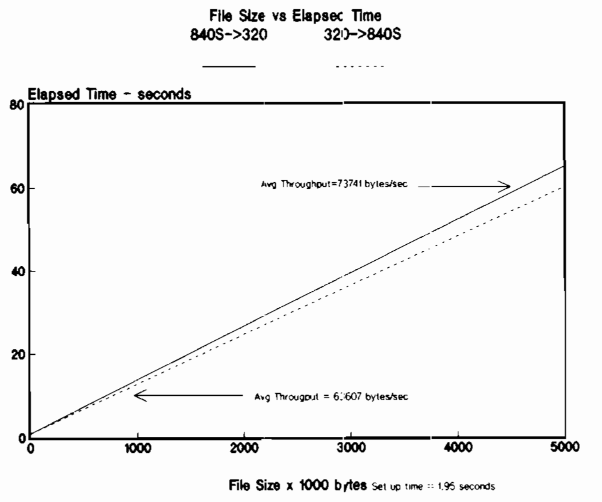


Figure 8.7. 840S to A900 NFT Elapsed Time

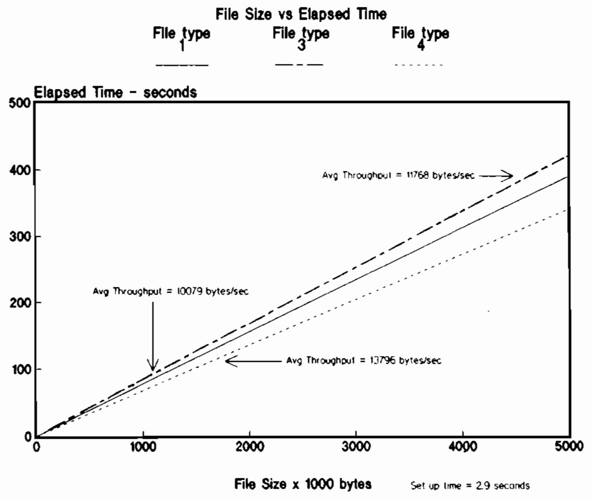


Figure 8.6. 840S to/from 320 RFA Elapsed Time

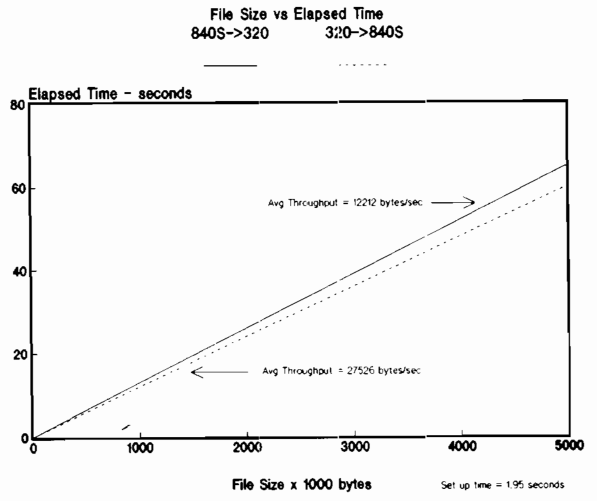
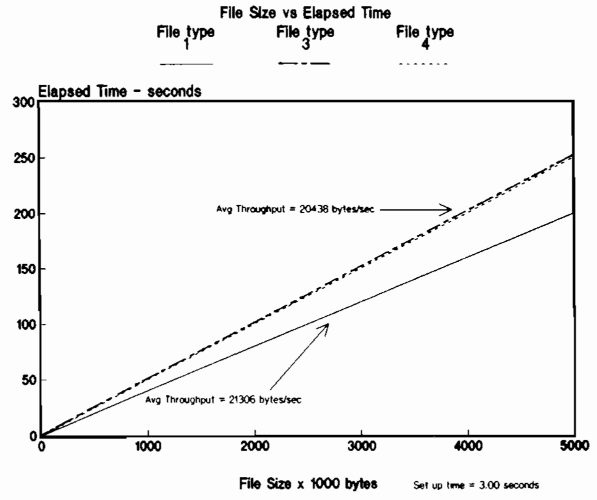


Figure 8.8. A900 to 840S NFT Elapsed Time



Section 9: NS Impact on Real-Time Performance

This section illustrates the effect of network file transfer (NFT) to and from an HP 1000 A900 to a Model 840S, on the real-time performance of the Model 840S. Refer to section 5 for a general description of real-time HP-UX performance. The measurements in this section compare the Model 840S's real-time response in the following situations:

* Varying degrees of system load with NS not configured.
* Varying degrees of system load with NS configured; files copied between a Model 840S and an HP 1000 A900 system using the NFT *dscopy* command.

All measurements were made on the Model 840S. Two groups of measurements define the situations:

* Interrupt Response Time (to the driver).
* Process Dispatch Time.

System and Network Load

The measured levels of system and network activity are defined below:

* **Test 0**: No workload; NS not configured.
* **Test 3**: 16 interactive program development sessions; a program writing continuously to an infinitely fast HPIB instrument; plus a tape archive (*tar*) continuously writing to tape; NS not configured.
* **Test 8**: Test 3 plus NS configured; one *dscopy* file transfer from the Model 840S to the HP 1000 A900.
* **Test 9**: Test 3 plus NS configured; one dscopy file transfer from the HP 1000 A900 to the Model 840S.
* **Test 10**: Test 3 plus NS configured; six dscopy file transfers to and from the Model 840S and the HP 1000 A900 (three each direction).
* **Test 11**: Test 0 plus NS configured; six dscopy file transfers to and from the Model 840S and the HP 1000 A900 (three each direction).

System Configuration

Each HP 1000 Model A900 had 302 KBytes allocated to networking.

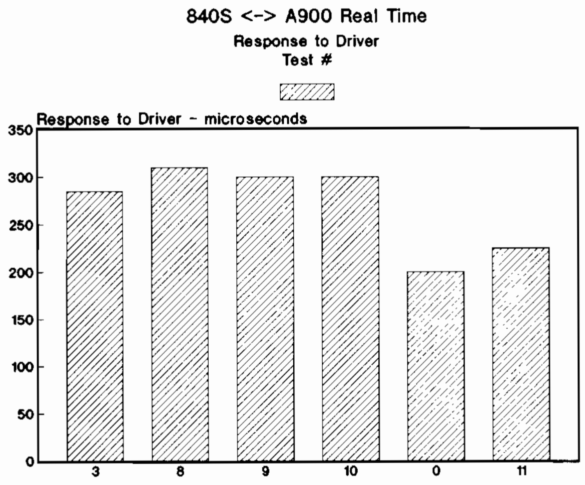
Real-Time Benchmark Methodology

A logic analyzer was set up to measure the time from external signal, which was caused by a CIO interrupt generator, to the execution of the first instruction in the Device Adapter Manager (DAM). This measured interrupt acknowledge time. The test was repeated measuring the time from the external signal to user-code entry, to measure process dispatch time.

Interrupt Response Time (to Driver)

Interrupt response time to the driver measures the time required to receive an interrupt, cease the current activity, save part of the current user process state, and begin execution of the first instruction in the device adapter manager (DAM). Figure 9.1 compares Model 840S interrupt response times to the driver for the activity levels defined above. All figures in the graph are mean times. The maximum observed interrupt response time in these test cases was 2.593 milliseconds. Response time will vary according to your workload and applications.

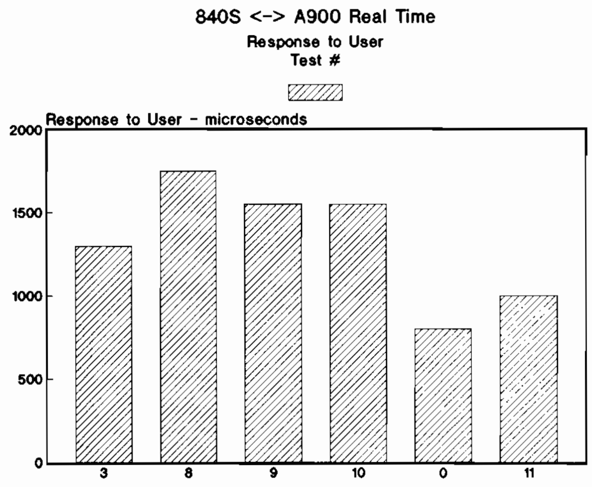
Figure 9.1. Mean interrupt response time (to driver).



Process Dispatch Time

A key measure of real-time operating system performance is how quickly a waiting process can be dispatched in response to some external event. Figure 9.2 illustrates the mean process dispatch times for the levels of network and system activity described above. The maximum observed process dispatch time for these tests was 21.8 milliseconds. Response time will vary according to workload and applications.

Figure 9.2. Mean process dispatch time.



NS Effect on Real-Time Performance: Conclusion

As expected, NS can affect real-time performance. However, with careful planning you can minimize the impact of NS on real-time response time. Here are two concrete steps you can take to decrease NS effect on real-time performance:

* When planning the network in a real-time HP-UX environment, minimize the number of nodes on the network to decrease network traffic and increase response time.
* After you have precisely defined your real-time environment, determine the maximum number of NFT connections you will use at one time. Add three to the number of connections; when your network is up, execute that number of NFT connections simultaneously. This activity will allocate the maximum amount of memory you will need for your NS operations. With memory already allocated, you save microseconds on each subsequent NS operation.

NOTE

The data in this section described only the effect of 840S/A900 NFT on Model 840S real-time performance. NFT between Series 800 machines may further degrade real-time response because of the higher speed of the two processors.

Appendix A

Aim Suite II Description

The descriptions of the Aim Suite II benchmarks in this section are provided by by Aim Technology.

Test Number	Tests	Unit/Per
01	getpid()	syscall/sec
02	sbrk()	syscall/sec
03	create-close calls	syscall/sec
33	umask(0) calls	syscall/sec
04	process forks	forks/sec
05	tty1 write	bytes/sec
36	tty1 + tty2 write	bytes/sec
06	disc write	bytes/sec
07	disc read	bytes/sec
08	disc copy	bytes/sec
09	pipe copy	bytes/sec
10	add long	add/sec
11	add short	add/sec
12	add float	add/sec
34	add double	add/sec
13	array ref short [short]	ref/short
14	array ref long [long]	ref/long
15	call funct()	calls/sec
16	call funct(int)	calls/sec
17	call funct(int,int)	calls/sec
18	ram read short	bytes/sec
19	ram read long	bytes/sec
20	ram read char	bytes/sec
21	ram write short	bytes/sec
22	ram write long	bytes/sec
23	ram write char	bytes/sec
24	ram copy short	bytes/sec
25	ram copy long	bytes/sec
26	ram copy char	bytes/sec
27	multiply short	mults/sec
28	multiply long	mults/sec
29	multiply float	mults/sec
35	multiply double	mults/sec
30	divide short	divs/sec
31	divide long	divs/sec
32	divide float	divs/sec
37	divide double	divs/sec

Tests 01, 02, 03, and 33 deal with system calls, statements which ask the UNIX system to perform a kernel function. These five system calls, getpid(), sbrk(), creat(), close(), and umask(), are easy to loop on and each is tested by being called a multiple of 512 times. The resulting response is measured in system calls completed per second.

Test number 04 is a fork timer. It measures how fast it takes to fork off n-subprocesses and, as such, measures the upper limit on how fast shell scripts can respond. Results are in forks created per second.

Tests 05 and 36 examine TTY write speed. Test 05 forks a subprocess which writes n-blocks (512 Bytes) to a single TTY port at 9600 Baud. Test 36 forks two subprocesses, each writing n-blocks (512 Bytes) to a TTY port; this test measures capability to output to two ports simultaneously. The results of both tests are in bytes per second.

Tests 06, 07, and 08 are disc bandwidth timers. In the tests, disc files are written, read and copied using 10 KBytes. No processing of files is done; I/O calls only are used. The three tests measure bytes processed per second.

Test number 09 is a pipe throughput timer. Pipes pass information between information between cooperating programs. The test results are in bytes per second.

Tests 10, 11, 12, 34, and 27, 28, 29, 35, and 30, 31, 32, and 37 are arithmetic timers. They test addition, multiplication and division in the short, long, floating point and double modes. In each test, the arithmetic operations are exercised by 32 loops so that a multiple of 512 operations are measured per call. The results are in number of operations per second.

Tests 13 and 14 are array reference timers. In the tests, nested short and long subscript references are timed. An inner loop is used to scale up so that 512 references are executed for each n. Results are in references per second.

Tests 15, 16, and 17 time function calls. In the tests, functions are called with no parameters, one parameter and two parameters. Each function is called a multiple of 512 times. Results are in calls completed per second.

Finally, tests 18, 19, 20, 21, 22, 23, 24, 25, and 26 examine RAM. In this series of tests, memory read, write and copy times are measured in short (2 bytes), long (4 bytes) and character modes (1 byte). Each test moves a multiple of 512 bytes and measures the maximum RAM-to-RAM rates. Results are in bytes per second.

Appendix B

Test System Configurations

The specific system configurations used to produce the benchmark results are detailed below.

HP 9000 Model 825S:

System was configured with:
40 MBytes memory
lockable memory: 31340544
available memory: 34813952

Addr 0: HPIB card to 3 7935 discs
(one disc configured as swap)
Addr 1: MUX card (console)
Addr 2: HPIB card to Tape Drive
Addr 3 - Addr 7: MUX cards
Addr 8: Lan0

HP 9000 Model 840S:

System was configured with:
24 MBytes memory
lockable memory: 18622464
available memory: 2052308

For all but real-time tests:

Addr 0: HPIB card to 3 7935 discs
Addr 1: MUX card (console)
Addr 2: HPIB card to Tape Drive
Addr 3 - Addr 11: MUX cards
Addr 13: Lan0

For real-time tests:

Addr 0: HPIB card to 2 7935 discs
Addr 1: MUX card (console)
Addr 2: HPIB card to Tape Drive
Addr 4: Lan0 to other Model 840S
Addr 5: Parallel card
Addr 6: HPIB card to nonexistent instrument
Addr 7 - Addr 11: MUX cards

HP 9000 Model 850S:

System was configured with:
32 MBytes memory

Addr 0: HPIB card to 3 7935 discs
(one disc configured as swap)
Addr 1: MUX card (console)
Addr 2: HPIB card to Tape Drive
Addr 3 - Addr 11: MUX cards
Addr 13: Lan0

HP 1000 A900:

System was configured with:

3 MBytes memory

1 7935H disc drive
1-5 8 channel MUX interfaces
2 HPIB interfaces for instruments
O/S - RTE-A
1 Parallel interface -- standard interface

HEWLETT
PACKARD

Printed in U.S.A. 5/87
5954-9903